

The L^AT_EX3 Sources

The L^AT_EX3 Project*

November 19, 2011

Abstract

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

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

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

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

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

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

Contents

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

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

31	Defining quarks	43
32	Quark tests	44
33	Recursion	44
34	Internal quark functions	45
VIII The l3token package: Token manipulation		46
35	All possible tokens	46
36	Character tokens	47
37	Generic tokens	50
38	Converting tokens	50
39	Token conditionals	51
40	Peeking ahead at the next token	54
41	Decomposing a macro definition	56
42	Experimental token functions	57
IX The l3int package: Integers		59
43	Integer expressions	59
44	Creating and initialising integers	60
45	Setting and incrementing integers	61
46	Using integers	62
47	Integer expression conditionals	62
48	Integer expression loops	63
49	Formatting integers	64
50	Converting from other formats to integers	66
51	Viewing integers	66
52	Constant integers	67

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

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

98	Sequence conditionals	99
99	Mapping to sequences	100
100	Sequences as stacks	101
101	Viewing sequences	102
102	Experimental sequence functions	102
103	Internal sequence functions	104
XIII	The l3clist package: Comma separated lists	106
104	Creating and initialising comma lists	106
105	Adding data to comma lists	107
106	Using comma lists	108
107	Modifying comma lists	108
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	117
118	Modifying property lists	117
119	Property list conditionals	118

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

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

161	Setting known keys only	160
162	Utility functions for keys	161
163	Low-level interface for parsing key–val lists	161
XXI	The l3file package: File operations	163
164	File operation functions	163
165	Internal file functions	164
XXII	The l3fp package: Floating-point operations	165
166	Floating-point variables	165
167	Conversion of floating point values to other formats	167
168	Rounding floating point values	167
169	Floating-point conditionals	168
170	Unary floating-point operations	169
171	Floating-point arithmetic	169
172	Floating-point power operations	170
173	Exponential and logarithm functions	171
174	Trigonometric functions	171
175	Constant floating point values	172
176	Notes on the floating point unit	172
XXIII	The l3luatex package: LuaTeX-specific functions	174
177	Breaking out to Lua	174
178	Category code tables	175
XXIV	Implementation	176

179	Bootstrap code	176
179.1	Format-specific code	176
179.2	Package-specific code	177
179.3	Dealing with package-mode meta-data	179
179.4	The <code>\pdfstrcmp</code> primitive in \LaTeX	182
179.5	Engine requirements	182
179.6	The \LaTeX 3 code environment	183
180	l3names implementation	184
181	l3basics implementation	194
181.1	Renaming some \TeX primitives (again)	195
181.2	Defining functions	196
181.3	Selecting tokens	197
181.4	Gobbling tokens from input	198
181.5	Conditional processing and definitions	199
181.6	Dissecting a control sequence	204
181.7	Exist or free	206
181.8	Defining and checking (new) functions	208
181.9	More new definitions	210
181.10	Copying definitions	211
181.11	Undefining functions	212
181.12	Defining functions from a given number of arguments	213
181.13	Using the signature to define functions	214
181.14	Checking control sequence equality	216
181.15	Diagnostic wrapper functions	217
181.16	Engine specific definitions	217
181.17	Doing nothing functions	218
181.18	String comparisons	218
181.19	Deprecated functions	218
182	l3expan implementation	220
182.1	General expansion	220
182.2	Hand-tuned definitions	223
182.3	Definitions with the automated technique	226
182.4	Last-unbraced versions	227
182.5	Preventing expansion	228
182.6	Defining function variants	229
182.7	Variants which cannot be created earlier	231

183	l3prg implementation	232
183.1	Primitive conditionals	232
183.2	Defining a set of conditional functions	232
183.3	The boolean data type	232
183.4	Boolean expressions	234
183.5	Logical loops	239
183.6	Switching by case	240
183.7	Producing n copies	242
183.8	Detecting T _E X's mode	245
183.9	Internal programming functions	246
183.10	Experimental programmings functions	247
183.11	Deprecated functions	250
184	l3quark implementation	250
185	l3token implementation	253
185.1	Character tokens	253
185.2	Generic tokens	255
185.3	Token conditionals	256
185.4	Peeking ahead at the next token	264
185.5	Decomposing a macro definition	269
185.6	Experimental token functions	270
185.7	Deprecated functions	271
186	l3int implementation	274
186.1	Integer expressions	274
186.2	Creating and initialising integers	276
186.3	Setting and incrementing integers	277
186.4	Using integers	278
186.5	Integer expression conditionals	278
186.6	Integer expression loops	281
186.7	Formatting integers	282
186.8	Converting from other formats to integers	287
186.9	Viewing integer	291
186.10	Constant integers	291
186.11	Scratch integers	292
186.12	Registers for earlier modules	292
186.13	Deprecated functions	292

187	l3skip implementation	293
187.1	Length primitives renamed	293
187.2	Creating and initialising <code>dim</code> variables	294
187.3	Setting <code>dim</code> variables	294
187.4	Utilities for dimension calculations	295
187.5	Dimension expression conditionals	296
187.6	Dimension expression loops	297
187.7	Using <code>dim</code> expressions and variables	299
187.8	Viewing <code>dim</code> variables	299
187.9	Constant dimensions	300
187.10	Scratch dimensions	300
187.11	Creating and initialising <code>skip</code> variables	300
187.12	Setting <code>skip</code> variables	301
187.13	<code>skip</code> expression conditionals	301
187.14	Using <code>skip</code> expressions and variables	302
187.15	Inserting skips into the output	302
187.16	Viewing <code>skip</code> variables	303
187.17	Constant skips	303
187.18	Scratch skips	303
187.19	Creating and initialising <code>muskip</code> variables	303
187.20	Setting <code>muskip</code> variables	304
187.21	Using <code>muskip</code> expressions and variables	304
187.22	Viewing <code>muskip</code> variables	305
187.23	Experimental skip functions	305
188	l3tl implementation	305
188.1	Functions	305
188.2	Adding to token list variables	307
188.3	Reassigning token list category codes	308
188.4	Reassigning token list character codes	310
188.5	Modifying token list variables	310
188.6	Token list conditionals	313
188.7	Mapping to token lists	316
188.8	Using token lists	317
188.9	Working with the contents of token lists	318
188.10	The first token from a token list	319
188.11	Viewing token lists	324
188.12	Constant token lists	324
188.13	Scratch token lists	325
188.14	Experimental functions	325
188.15	Deprecated functions	330

189	l3seq implementation	332
189.1	Allocation and initialisation	332
189.2	Appending data to either end	333
189.3	Modifying sequences	334
189.4	Sequence conditionals	335
189.5	Recovering data from sequences	336
189.6	Mapping to sequences	338
189.7	Sequence stacks	340
189.8	Viewing sequences	341
189.9	Experimental functions	342
189.10	Deprecated interfaces	348
190	l3clist implementation	348
190.1	Allocation and initialisation	349
190.2	Removing spaces around items	350
190.3	Adding data to comma lists	351
190.4	Comma lists as stacks	352
190.5	Using comma lists	353
190.6	Modifying comma lists	353
190.7	Comma list conditionals	355
190.8	Mapping to comma lists	356
191	Viewing comma lists	358
191.1	Scratch comma lists	359
191.2	Experimental functions	359
191.3	Deprecated interfaces	362
192	l3prop implementation	363
192.1	Allocation and initialisation	363
192.2	Accessing data in property lists	364
192.3	Property list conditionals	367
192.4	Recovering values from property lists with branching	368
192.5	Mapping to property lists	369
192.6	Viewing property lists	370
192.7	Experimental functions	371
192.8	Deprecated interfaces	372

193	l3box implementation	373
193.1	Creating and initialising boxes	374
193.2	Measuring and setting box dimensions	375
193.3	Using boxes	375
193.4	Box conditionals	376
193.5	The last box inserted	376
193.6	Constant boxes	376
193.7	Scratch boxes	377
193.8	Viewing box contents	377
193.9	Horizontal mode boxes	377
193.10	Vertical mode boxes	379
193.11	Affine transformations	380
193.12	Viewing part of a box	389
193.13	Deprecated functions	390
194	l3coffins Implementation	391
194.1	Coffins: data structures and general variables	391
194.2	Basic coffin functions	393
194.3	Measuring coffins	397
194.4	Coffins: handle and pole management	397
194.5	Coffins: calculation of pole intersections	400
194.6	Aligning and typesetting of coffins	404
194.7	Rotating coffins	408
194.8	Resizing coffins	413
194.9	Coffin diagnostics	415
194.10	Messages	421
195	l3color Implementation	422
196	l3io implementation	423
196.1	Primitives	423
196.2	Variables and constants	423
196.3	Stream management	424
196.4	Deferred writing	429
196.5	Immediate writing	430
196.6	Special characters for writing	430
196.7	Hard-wrapping lines based on length	431
196.8	Reading input	436
196.9	Deprecated functions	437
197	l3msg implementation	437

198	Creating messages	438
198.1	Messages: support functions and text	439
198.2	Showing messages: low level mechanism	440
198.3	Displaying messages	442
198.4	Kernel-specific functions	447
198.5	Expandable errors	451
198.6	Deprecated functions	452
199	l3keys Implementation	453
199.1	Low-level interface	453
199.2	Constants and variables	456
199.3	The key defining mechanism	458
199.4	Turning properties into actions	459
199.5	Creating key properties	464
199.6	Setting keys	467
199.7	Utilities	470
199.8	Messages	471
199.9	Deprecated functions	472
200	l3file implementation	472
201	l3fp Implementation	476
201.1	Constants	476
201.2	Variables	477
201.3	Parsing numbers	480
201.4	Internal utilities	483
201.5	Operations for fp variables	484
201.6	Transferring to other types	489
201.7	Rounding numbers	496
201.8	Unary functions	498
201.9	Basic arithmetic	500
201.10	Arithmetic for internal use	509
201.11	Trigonometric functions	515
201.12	Exponent and logarithm functions	528
201.13	Tests for special values	550
201.14	Floating-point conditionals	550
201.15	Messages	556
202	l3luatex implementation	557
202.1	Category code tables	558
	Index	561

Part I

Introduction to expl3 and this document

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

1 Naming functions and variables

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

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

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

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

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

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

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

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

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

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

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

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

skip “Rubber” lengths.

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

tl Token list variables: placeholder for a token list.

1.1 Terminological inexactitude

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

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

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

2 Documentation conventions

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

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

`\ExplSyntaxOn`
`\ExplSyntaxOff`

`\ExplSyntaxOn ... \ExplSyntaxOff`

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

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

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

<code>\seq_new:N</code>	<code>\seq_new:N</code> <i><sequence></i>
<code>\seq_new:c</code>	

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

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

<code>\cs_to_str:N</code> ★	<code>\cs_to_str:N</code> <i><cs></i>
-----------------------------	---

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 *<cs>*, shorthand for a *<control sequence>*.

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:

<code>\seq_map_function:NN</code> ☆	<code>\seq_map_function:NN</code> <i><seq></i> <i><function></i>
-------------------------------------	--

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:

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

The underlining and italic of **TF** indicates that `\xetex_if_engine:T`, `\xetex_if_engine:F` and `\xetex_if_engine:TF` are all available. Usually, the illustration will use the **TF** variant, and so both *<true code>* and *<false code>* will be shown. The two variant forms **T** and **F** take only *<true code>* and *<false code>*, 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:

<code>\l_tmpa_tl</code>	
-------------------------	--

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

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

The normal description text.

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

3 Formal language conventions which apply generally

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

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

Part II

The l3bootstrap package

Bootstrap code

4 Using the L^AT_EX3 modules

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

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

`\ExplSyntaxOn`
`\ExplSyntaxOff`

Updated: 2011-08-13

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

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

`\ExplSyntaxNamesOn`
`\ExplSyntaxNamesOff`

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

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

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

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

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

`\GetIdInfo`

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

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

To summarize: Every single package using this syntax should identify itself using one of the above methods. Special care is taken so that every package or class file loaded with `\RequirePackage` or alike are loaded with usual $\text{\LaTeX} 2_{\epsilon}$ category codes and the $\text{\LaTeX} 3$ category code scheme is reloaded when needed afterwards. See implementation for details. If you use the `\GetIdInfo` command you can use the information when loading a package with

```
\ProvidesExplPackage{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescript
```

Part III

The l3names package Namespace for primitives

5 Setting up the L^AT_EX3 programming language

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

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

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

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

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

`\pdftex_...` Introduced by pdfT_EX;

`\xetex_...` Introduced by X_YT_EX;

`\luatex_...` Introduced by LuaT_EX.

Part IV

The l3basics package

Basic definitions

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

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

6 No operation functions

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

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

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

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

7 Grouping material

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

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

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

`\group_insert_after:N`**`\group_insert_after:N` *(token)***

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

8 Control sequences and functions

As 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, $\langle code \rangle$ 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.

8.2 Defining new functions using primitive parameter text

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

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

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

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

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

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

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

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

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

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

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

<code>\cs_set:Npn</code>	<code>\cs_set:Npn <function> <parameters> {<code>}</code>
<code>\cs_set:(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. The assignment of a meaning to $\langle function \rangle$ is restricted to the current TeX group level.

<code>\cs_set_nopar:Npn</code>	<code>\cs_set_nopar:Npn <function> <parameters> {<code>}</code>
<code>\cs_set_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 $\langle function \rangle$ is restricted to the current TeX group level.

<code>\cs_set_protected:Npn</code>	<code>\cs_set_protected:Npn <function> <parameters> {<code>}</code>
<code>\cs_set_protected:(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. The assignment of a meaning to $\langle function \rangle$ is restricted to the current TeX group level. The $\langle function \rangle$ will not expand within an x-type argument.

<code>\cs_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 $\langle function \rangle$ 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 $\langle function \rangle$ is *not* restricted to the current TeX group level: the assignment is global.

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

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

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

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

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

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

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

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

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

8.3 Defining new functions using the signature

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Updated: 2011-09-05

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

8.4 Copying control sequences

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

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

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

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

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

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

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

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

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

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

8.5 Deleting control sequences

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

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

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

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

Updated: 2011-09-15

8.6 Showing control sequences

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

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

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

\TeX hackers note: This is \TeX 's $\backslash\text{meaning}$ primitive.

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

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

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

\TeX hackers note: This is the \TeX primitive $\backslash\text{show}$.

8.7 Converting to and from control sequences

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

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

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

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

and

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

would be equivalent to

```
\abc
```

after two expansions of `\use:c`.

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

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

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

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

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

and

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

would be equivalent to

```
\abc
```


after one expansion of `\cs:w`.

<code>\cs_to_str:N</code>	<code>\cs_to_str:N</code>	★	<code>\{control sequence\}</code>
---------------------------	---------------------------	---	-----------------------------------

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 a variable number of expansion steps (either 3 or 4), and so an **f**- or **x**-type expansion will be required to convert the *<control sequence>* to a sequence of characters in the input stream.

9 Using or removing tokens and arguments

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

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

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

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

will result in the input stream containing

`abc { def }`

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

<code>\use_i:nn</code>	★	<code>\use_i:nn</code>	<code>\{group₁\}</code> <code>\{group₂\}</code>
<code>\use_ii:nn</code>	★		

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

<code>\use_i:nnn</code>	★	<code>\use_i:nnn</code>	<code>\{group₁\}</code> <code>\{group₂\}</code> <code>\{group₃\}</code>
<code>\use_ii:nnn</code>	★		
<code>\use_iii:nnn</code>	★		

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

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

<code>\use_i_ii:nnn</code>	★	<code>\use_i_ii:nnn {⟨group₁⟩} {⟨group₂⟩} {⟨group₃⟩}</code>
----------------------------	---	--

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

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

will result in the input stream containing

`abc { def }`

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

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

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

<code>\use:x</code>		<code>\use:x {⟨expandable tokens⟩}</code>
---------------------	--	---

Fully expands the `⟨expandable tokens⟩` 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_recursion_stop:w</code>	★	

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

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

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

9.2 Decomposing control sequences

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

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

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

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

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

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

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

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

10 Predicates and conditionals

L^AT_EX3 has three concepts for conditional flow processing:

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

`\cs_if_free:cTF{abc} {\langle true code \rangle} {\langle false code \rangle}`

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

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

Important to note is that these branching conditionals with $\langle true\ code \rangle$ and/or $\langle false\ code \rangle$ are always defined in a way that the code of the chosen alternative can operate on following tokens in the input stream.

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

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

```
\cs_if_free_p:N
```

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

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

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

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

```
\c_true_bool
\c_false_bool
```

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

10.1 Tests on control sequences

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

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

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

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

10.2 Testing string equality

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

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

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

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

10.3 Engine-specific conditionals

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

Updated: 2011-09-06

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

Updated: 2011-09-06

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

Updated: 2011-09-06

10.4 Primitive conditionals

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

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

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

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

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

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

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

<code>\if:w</code>	★	<code>\if:w <token₁> <token₂> <true code> \else: <false code> \fi:</code>
<code>\if_charcode:w</code>	★	<code>\if_catcode:w <token₁> <token₂> <true code> \else: <false code> \fi:</code>
<code>\if_catcode:w</code>	★	These conditionals will expand any following tokens until two unexpandable tokens are left. If you wish to prevent this expansion, prefix the token in question with <code>\exp_not:N</code> . <code>\if_catcode:w</code> tests if the category codes of the two tokens are the same whereas <code>\if:w</code> tests if the character codes are identical. <code>\if_charcode:w</code> is an alternative name for <code>\if:w</code> .

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

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

<code>\if_mode_horizontal</code>	★	<code>\if_mode_horizontal: <true code> \else: <false code> \fi:</code>
<code>\if_mode_vertical</code>	★	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.

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

12 Experimental functions

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

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

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

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

Part V

The l3expan package

Argument expansion

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

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

13 Defining new variants

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

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

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

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

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

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

The steps above may be automated by using the function `\cs_generate_variant:Nn`, described next.

14 Methods for defining variants

`\cs_generate_variant:Nn`

Updated: 2011-09-15

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

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

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

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

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

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

15 Introducing the variants

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

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

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

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

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

The `f` type is so special that it deserves an example. Let's pretend we want to set `\aaa` equal to the control sequence stemming from turning `b \l_tmpa_tl b` into a control sequence. Furthermore we want to store the execution of it in a *tl var*. In this example we assume `\l_tmpa_tl` contains the text string `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>} {\<tokens_2>} ...
```

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

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

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

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

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

17 Manipulating two arguments

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

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

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

18 Manipulating three arguments

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

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

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

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

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

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

19 Unbraced expansion

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

`\exp_last_two_unbraced:Noo` ★ `\exp_last_two_unbraced:Noo` $\langle token \rangle$ $\langle tokens1 \rangle$ $\{\langle tokens2 \rangle\}$

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.

`\exp_after:wN` ★ `\exp_after:wN` $\langle token1 \rangle$ $\langle token2 \rangle$

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

\TeX hackers note: This is the \TeX 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.

`\exp_not:N` ★ `\exp_not:N` $\langle token \rangle$

Prevents expansion of the $\langle token \rangle$ in a context where it would otherwise be expanded, for example an `x`-type argument.

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

`\exp_not:c` ★ `\exp_not:c` $\{\langle tokens \rangle\}$

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.

`\exp_not:n` ★ `\exp_not:n` $\{\langle tokens \rangle\}$

Prevents expansion of the $\langle tokens \rangle$ in a context where they would otherwise be expanded, for example an `x`-type argument.

\TeX hackers note: This is the ε - \TeX `\unexpanded` primitive.

`\exp_not:V` ★ `\exp_not:V` $\langle variable \rangle$

Recovers the content of the $\langle variable \rangle$, then prevents expansion of the this material in a context where it would otherwise be expanded, for example an `x`-type argument.

<hr/> <hr/>	<code>\exp_not:v</code> ★	<code>\exp_not:v {\tokens}</code>	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 {\tokens}</code>	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 {\tokens}</code>	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 \tokens \exp_stop_f: \more tokens</code>	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).
Updated: 2011-06-03			

21 Internal functions and variables

<hr/> <hr/>	<code>\l_exp_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/> <hr/>	<code>\exp_eval_register:N</code> ★ <code>\exp_eval_register:c</code> ★	<code>\exp_eval_register:N \variable</code> These functions evaluates a $\langle variable \rangle$ as part of a V or v 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 \TeX register such as <code>\count</code> .
<hr/> <hr/>	<code>\:::n</code> <code>\:::N</code> <code>\:::c</code> <code>\:::o</code> <code>\:::f</code> <code>\:::x</code> <code>\:::v</code> <code>\:::V</code> <code>\:::</code>	<code>\cs_set_nopar:Npn \exp_args:Ncof { \:::c \:::o \:::f \::: }</code> Internal forms for the base expansion types. These names do <i>not</i> conform to the general \LaTeX 3 approach as this makes them more readily visible in the log and so forth.

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

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

Part VI

The l3prg package

Control structures

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

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

22 Defining a set of conditional functions

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

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

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

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

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

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

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

An example can easily clarify matters here:

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

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

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

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

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

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

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

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

```
\prg_new_eq_conditional:NN
\prg_set_eq_conditional:NN
```

```
\prg_new_eq_conditional:NN \langle name1 \rangle:\langle arg\ spec1 \rangle \langle name2 \rangle:\langle arg\ spec2 \rangle
```

These will set the definitions of the functions

- $\backslash\langle name1 \rangle_p:\langle arg\ spec1 \rangle$
- $\backslash\langle name1 \rangle:\langle arg\ spec1 \rangle T$
- $\backslash\langle name1 \rangle:\langle arg\ spec1 \rangle F$
- $\backslash\langle name1 \rangle:\langle arg\ spec1 \rangle TF$

equal to those for

- $\backslash\langle name2 \rangle_p:\langle arg\ spec2 \rangle$
- $\backslash\langle name2 \rangle:\langle arg\ spec2 \rangle T$
- $\backslash\langle name2 \rangle:\langle arg\ spec2 \rangle F$
- $\backslash\langle name2 \rangle:\langle arg\ spec2 \rangle TF$

In most cases, the two $\langle arg\ specs \rangle$ will be identical, although this is not enforced. In the case of the **new** function, a check is made for any existing definitions for $\langle name1 \rangle$. The functions are set globally.

```
\prg_return_true  ★ \prg_return_true:
\prg_return_false ★ \prg_return_false:
```

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 $\backslash prg_set_conditional:Npnn$, etc.

23 The boolean data type

This section describes a boolean data type which is closely connected to conditional processing as sometimes you want to execute some code depending on the value of a

switch (*e.g.*, draft/final) and other times you perhaps want to use it as a predicate function in an `\if_predicate:w` test. The problem of the primitive `\if_false:` and `\if_true:` tokens is that it is not always safe to pass them around as they may interfere with scanning for termination of primitive conditional processing. Therefore, we employ two canonical booleans: `\c_true_bool` or `\c_false_bool`. Besides preventing problems as described above, it also allows us to implement a simple boolean parser supporting the logical operations And, Or, Not, *etc.* which can then be used on both the boolean type and predicate functions.

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

<hr/> <code>\bool_new:N</code> <hr/> <code>\bool_new:c</code> <hr/>	<code>\bool_new:N <boolean></code> Creates a new <i><boolean></i> or raises an error if the name is already taken. The declaration is global. The <i><boolean></i> will initially be false .
<hr/> <code>\bool_set_false:N</code> <hr/> <code>\bool_set_false:c</code> <hr/>	<code>\bool_set_false:N <boolean></code> Sets <i><boolean></i> logically false within the current TeX group.
<hr/> <code>\bool_gset_false:N</code> <hr/> <code>\bool_gset_false:c</code> <hr/>	<code>\bool_gset_false:N <boolean></code> Sets <i><boolean></i> logically false globally.
<hr/> <code>\bool_set_true:N</code> <hr/> <code>\bool_set_true:c</code> <hr/>	<code>\bool_set_true:N <boolean></code> Sets <i><boolean></i> logically true within the current TeX group.
<hr/> <code>\bool_gset_true:N</code> <hr/> <code>\bool_gset_true:c</code> <hr/>	<code>\bool_gset_true:N <boolean></code> Sets <i><boolean></i> logically true globally.
<hr/> <code>\bool_set_eq:NN</code> <hr/> <code>\bool_set_eq:(cN Nc cc)</code> <hr/>	<code>\bool_set_eq:NN <boolean1> <boolean2></code> Sets the content of <i><boolean1></i> equal to that of <i><boolean2></i> . This assignment is restricted to the current TeX group level.
<hr/> <code>\bool_gset_eq:NN</code> <hr/> <code>\bool_gset_eq:(cN Nc cc)</code> <hr/>	<code>\bool_gset_eq:NN <boolean1> <boolean2></code> Sets the content of <i><boolean1></i> equal to that of <i><boolean2></i> . This assignment is global and so is not limited by the current TeX group level.
<hr/> <code>\bool_set:Nn</code> <hr/> <code>\bool_set:cn</code> <hr/>	<code>\bool_set:Nn <boolean> {<boolexpr>}</code> Evaluates the <i><boolean expression></i> as described for <code>\bool_if:n(TF)</code> , and sets the <i><boolean></i> variable to the logical truth of this evaluation. This assignment is local.

<hr/> <code>\bool_gset:Nn</code> <hr/>	<code>\bool_gset:Nn <boolean> {\<boolexpr>}</code>
<code>\bool_gset:cn</code> <hr/>	Evaluates the <i><boolean expression></i> as described for <code>\bool_if:n(TF)</code> , and sets the <i><boolean></i> variable to the logical truth of this evaluation. This assignment is global.
<hr/> <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> <hr/>	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> <hr/>	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/>	<pre> \prg_case_dim:nnn {<test dimension expression>} { {<dimexpr case₁>} {<code case₁>} {<dimexpr case₂>} {<code case₂>} ... {<dimexpr case_n>} {<code case_n>} } {<else case>} </pre>
--	---

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

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

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

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

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

27 Producing n copies

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

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

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

Updated: 2011-09-06

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

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

would print

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

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

Updated: 2011-09-06

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

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

Updated: 2011-09-06

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

28 Detecting T_EX's mode

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

Detects if T_EX is currently in horizontal mode.

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

Detects if T_EX is currently in inner mode.

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

Detects if T_EX is currently in maths mode.

Updated: 2011-09-05

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

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

29 Internal programming functions

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

Updated: 2011-08-11

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

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

Updated: 2011-09-06

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

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

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

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

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

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

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

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

<code>\if_bool:N</code>	★	<code>\if_bool:N \langle boolean \rangle \langle true code \rangle \else: \langle false code \rangle \fi:</code>
-------------------------	---	--

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

30 Experimental programmings functions

<code>\prg_quicksort:n</code>	<code>\prg_quicksort:n { {<item₁>} {<item₂>} ... {<item_n>} }</code>
-------------------------------	--

Performs a quicksort on the token list. The comparisons are performed by the function `\prg_quicksort_compare:nnTF` which is up to the programmer to define. When the sorting process is over, all items are given as argument to the function `\prg_quicksort_function:n` which the programmer also controls.

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

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

```
\cs_set_nopar:Npn\prg_quicksort_function:n #1{{#1}}
\cs_set_nopar:Npn\prg_quicksort_compare:nnTF #1#2 {\int_compare:nNnTF{#1}>{#2}}
```

Then the function call

```
\prg_quicksort:n {876234520}
```

would return `{0}{2}{2}{3}{4}{5}{6}{7}{8}`. An alternative example where one sorts a list of words, `\prg_quicksort_compare:nnTF` could be defined as

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

Part VII

The l3quark package

Quarks

A special type of constants in L^AT_EX3 are “quarks”. These are control sequences that expand to themselves and should therefore *never* be executed directly in the code. This would result in an endless loop!

They are meant to be used as delimiter is weird functions (for example as the stop token (*i.e.* `\q_stop`). They also permit the following ingenious trick: when you pick up a token in a temporary, and you want to know whether you have picked up a particular quark, all you have to do is compare the temporary to the quark using `\if_meaning: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.

By convention all constants of type quark start out with `\q_`.

31 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.

32 Quark tests

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

<code>\quark_if_nil_p:N</code>	★	<code>\quark_if_nil_p:N <token></code>
<code>\quark_if_nil: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).

33 Recursion

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

This quark is appended to the data structure in question and appears as a real element there. This means it gets any list separators around it.

<code>\q_recursion_stop</code>		This quark is added <i>after</i> the data structure. Its purpose is to make it possible to terminate the recursion at any point easily.
--------------------------------	--	---

<code>\quark_if_recursion_tail_stop:N</code>	<code>\quark_if_recursion_tail_stop:N {\token}</code>
--	---

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

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

Updated: 2011-09-06

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

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

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

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

Updated: 2011-09-06

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

34 Internal quark functions

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

<code>\char_set_uccode:nn</code>	<code>\char_set_uccode:nn {⟨integer₁⟩} {⟨integer₂⟩}</code>
----------------------------------	--

This function set up the behaviour of $\langle character \rangle$ when found inside `\tl_to_uppercase:n`, such that $\langle character_1 \rangle$ will be converted into $\langle character_2 \rangle$. The two $\langle characters \rangle$ may be specified using an $\langle integer\ expression \rangle$ for the character code concerned. This may include the T_EX ‘ $\langle character \rangle$ ’ method for converting a single character into its character code:

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

The setting applies within the current T_EX group.

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

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

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

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

<code>\char_set_mathcode:nn</code>	<code>\char_set_mathcode:nn {⟨integer₁⟩} {⟨integer₂⟩}</code>
------------------------------------	--

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

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

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

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

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

<code>\char_set_sfcode:nn</code>	<code>\char_set_sfcode:nn {⟨integer₁⟩} {⟨integer₂⟩}</code>
----------------------------------	--

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

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

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

<hr/> <code>\char_show_value_sfcode:n</code> <hr/>	<code>\char_show_value_sfcode:n {\langle integer expression \rangle}</code>
	Displays the current space factor for the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$ on the terminal.

37 Generic tokens

<hr/> <code>\token_new:Nn</code> <hr/>	<code>\token_new:Nn \langle token1 \rangle {\langle token2 \rangle}</code>
	Defines $\langle token1 \rangle$ to globally be a snapshot of $\langle token2 \rangle$. This will be an implicit representation of $\langle token2 \rangle$.

<hr/> <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> <hr/>	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.
--	---

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

<hr/> <code>\c_catcode_active_tl</code> <hr/>	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

<hr/> <code>\token_to_meaning:N</code> ★ <hr/>	<code>\token_to_meaning:N \langle token \rangle</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 <code>\cs_set_nopar:Npn</code> and token list variables defined using <code>\tl_new:N</code> will be described as macros .

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

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

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

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

39 Token conditionals

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Updated: 2001-05-23

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

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

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

<code>\token_if_expandable_p:N</code>	<code>*</code>	<code>\token_if_expandable_p:N</code>	<code><token></code>
<code>\token_if_expandable:NTF</code>	<code>*</code>	<code>\token_if_expandable:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is expandable. This test returns `<false>` for an undefined token.

<code>\token_if_long_macro_p:N</code>	<code>*</code>	<code>\token_if_long_macro_p:N</code>	<code><token></code>
<code>\token_if_long_macro:NTF</code>	<code>*</code>	<code>\token_if_long_macro:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is a long macro.

<code>\token_if_protected_macro_p:N</code>	<code>*</code>	<code>\token_if_protected_macro_p:N</code>	<code><token></code>
<code>\token_if_protected_macro:NTF</code>	<code>*</code>	<code>\token_if_protected_macro:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` 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>*</code>	<code>\token_if_protected_long_macro_p:N</code>	<code><token></code>
<code>\token_if_protected_long_macro:NTF</code>	<code>*</code>	<code>\token_if_protected_long_macro:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is a protected long macro.

<code>\token_if_chardef_p:N</code>	<code>*</code>	<code>\token_if_chardef_p:N</code>	<code><token></code>
<code>\token_if_chardef:NTF</code>	<code>*</code>	<code>\token_if_chardef:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be a chardef.

<code>\token_if_mathchardef_p:N</code>	<code>*</code>	<code>\token_if_mathchardef_p:N</code>	<code><token></code>
<code>\token_if_mathchardef:NTF</code>	<code>*</code>	<code>\token_if_mathchardef:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be a mathchardef.

<code>\token_if_dim_register_p:N</code>	<code>*</code>	<code>\token_if_dim_register_p:N</code>	<code><token></code>
<code>\token_if_dim_register:NTF</code>	<code>*</code>	<code>\token_if_dim_register:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be a dimension register.

<code>\token_if_int_register_p:N</code>	<code>*</code>	<code>\token_if_int_register_p:N</code>	<code><token></code>
<code>\token_if_int_register:NTF</code>	<code>*</code>	<code>\token_if_int_register:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be an integer register.

<code>\token_if_skip_register_p:N</code>	<code>*</code>	<code>\token_if_skip_register_p:N</code>	<code><token></code>
<code>\token_if_skip_register:NTF</code>	<code>*</code>	<code>\token_if_skip_register:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be a skip register.

<code>\token_if_toks_register_p:N</code>	<code>*</code>	<code>\token_if_toks_register_p:N</code>	<code><token></code>
<code>\token_if_toks_register:NTF</code>	<code>*</code>	<code>\token_if_toks_register:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if the `<token>` is defined to be a toks register (not used by L^AT_EX3).

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

Updated: 2001-05-23

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

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.

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

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

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

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

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

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

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

Updated: 2011-07-02

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

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

Updated: 2011-07-02

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

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

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

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

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

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

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

<u>\peek_meaning:NTF</u>	\peek_meaning: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 meaning as the $\langle test\ token \rangle$ (as defined by the test \token_if_eq_meaning: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).

<u>\peek_meaning_ignore_spaces:NTF</u>	\peek_meaning_ignore_spaces: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 meaning as the $\langle test\ token \rangle$ (as defined by the test \token_if_eq_meaning: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).

<u>\peek_meaning_remove:NTF</u>	\peek_meaning_remove: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 meaning as the $\langle test\ token \rangle$ (as defined by the test \token_if_eq_meaning: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).

<u>\peek_meaning_remove_ignore_spaces:NTF</u>	\peek_meaning_remove_ignore_spaces: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 meaning as the $\langle test\ token \rangle$ (as defined by the test \token_if_eq_meaning:NNTF). Spaces are ignored by the test and the $\langle token \rangle$ will be removed from the input stream if the test is true. The function will then place either the $\langle true\ code \rangle$ or $\langle false\ code \rangle$ in the input stream (as appropriate to the result of the test).

41 Decomposing a macro definition

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

`\token_get_arg_spec:N` ★ `\token_get_arg_spec:N` $\langle token \rangle$

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

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

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

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

`\token_get_replacement_text:N` ★ `\token_get_replacement_text:N` $\langle token \rangle$

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

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

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

`\token_get_prefix_spec:N` ★ `\token_get_prefix_spec:N` $\langle token \rangle$

If the $\langle token \rangle$ is a macro, this function will leave the T_EX 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

`\char_active_set:Npn` `\char_active_set:Npn` $\langle char \rangle$ $\langle parameters \rangle$ $\{ \langle code \rangle \}$

`\char_active_set:Npx`

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 This definition is local to the current T_EX group.

`\char_active_gset:Npn` `\char_active_gset:Npn` $\langle char \rangle$ $\langle parameters \rangle$ $\{ \langle code \rangle \}$

`\char_active_gset:Npx`

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 This definition is global.

<hr/> <hr/> <code>\char_active_set_eq:NN</code>	<code>\char_active_set_eq:NN</code> $\langle char \rangle$ $\langle function \rangle$
	Makes $\langle char \rangle$ an active character equivalent in meaning to the $\langle function \rangle$ (which may itself be an active character). This definition is local to the current \TeX group.
<hr/> <hr/> <code>\char_active_gset_eq:NN</code>	<code>\char_active_gset_eq:NN</code> $\langle char \rangle$ $\langle function \rangle$
	Makes $\langle char \rangle$ an active character equivalent in meaning to the $\langle function \rangle$ (which may itself be an active character). This definition is global.
<hr/> <hr/> <code>\peek_N_typeTF</code>	<code>\peek_N_type:TF</code> $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<hr/> <hr/> <small>New: 2011-08-14</small>	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, round any remainder. Note that this is identical to using `/` directly in an *⟨integer expression⟩*. The result is left in the input stream as a *⟨integer denotation⟩* after two expansions.

<hr/> <code>\int_div_truncate:nn</code> ★ <hr/>	<code>\int_div_truncate:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>	Evaluates the two <i>integer expressions</i> 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 <i>integer denotation</i> after two expansions.
<hr/> <code>\int_max:nn</code> ★ <code>\int_min:nn</code> ★ <hr/>	<code>\int_max:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code> <code>\int_min:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>	Evaluates the <i>integer expressions</i> as described for <code>\int_eval:n</code> and leaves either the larger or smaller value in the input stream as an <i>integer denotation</i> after two expansions.
<hr/> <code>\int_mod:nn</code> ★ <hr/>	<code>\int_mod:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>	Evaluates the two <i>integer expressions</i> 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 <i>integer denotation</i> after two expansions.

44 Creating and initialising integers

<hr/> <code>\int_new:N</code> <code>\int_new:c</code> <hr/>	<code>\int_new:N \langle integer \rangle</code>	Creates a new <i>integer</i> or raises an error if the name is already taken. The declaration is global. The <i>integer</i> will initially be equal to 0.
<hr/> <code>\int_const:Nn</code> <code>\int_const:cn</code> Updated: 2011-10-22 <hr/>	<code>\int_const:Nn \langle integer \rangle {\langle integer expression \rangle}</code>	Creates a new constant <i>integer</i> or raises an error if the name is already taken. The value of the <i>integer</i> will be set globally to the <i>integer expression</i> .
<hr/> <code>\int_zero:N</code> <code>\int_zero:c</code> <hr/>	<code>\int_zero:N \langle integer \rangle</code>	Sets <i>integer</i> to 0 within the scope of the current TeX group.
<hr/> <code>\int_gzero:N</code> <code>\int_gzero:c</code> <hr/>	<code>\int_gzero:N \langle integer \rangle</code>	Sets <i>integer</i> to 0 globally, <i>i.e.</i> not restricted by the current TeX group level.
<hr/> <code>\int_set_eq:NN</code> <code>\int_set_eq:(cN Nc cc)</code> <hr/>	<code>\int_set_eq:NN \langle integer1 \rangle \langle integer2 \rangle</code>	Sets the content of <i>integer1</i> equal to that of <i>integer2</i> . This assignment is restricted to the current TeX group level.
<hr/> <code>\int_gset_eq:NN</code> <code>\int_gset_eq:(cN Nc cc)</code> <hr/>	<code>\int_gset_eq:NN \langle integer1 \rangle \langle integer2 \rangle</code>	Sets the content of <i>integer1</i> equal to that of <i>integer2</i> . This assignment is global and so is not limited by the current TeX group level.

45 Setting and incrementing integers

<hr/> <code>\int_add:Nn</code> <hr/> <code>\int_add:cn</code> <hr/>	<code>\int_add:Nn <integer> {<integer expression>}</code>
Updated: 2011-10-22	Adds the result of the <i><integer expression></i> to the current content of the <i><integer></i> . This assignment is local.
<hr/> <code>\int_gadd:Nn</code> <hr/> <code>\int_gadd:cn</code> <hr/>	<code>\int_gadd:Nn <integer> {<integer expression>}</code>
	Adds the result of the <i><integer expression></i> to the current content of the <i><integer></i> . This assignment is global.
<hr/> <code>\int_decr:N</code> <hr/> <code>\int_decr:c</code> <hr/>	<code>\int_decr:N <integer></code>
	Decreases the value stored in <i><integer></i> by 1 within the scope of the current TeX group.
<hr/> <code>\int_gdecr:N</code> <hr/> <code>\int_gdecr:c</code> <hr/>	<code>\int_incr:N <integer></code>
	Decreases the value stored in <i><integer></i> by 1 globally (<i>i.e.</i> not limited by the current group level).
<hr/> <code>\int_incr:N</code> <hr/> <code>\int_incr:c</code> <hr/>	<code>\int_incr:N <integer></code>
	Increases the value stored in <i><integer></i> by 1 within the scope of the current TeX group.
<hr/> <code>\int_gincr:N</code> <hr/> <code>\int_gincr:c</code> <hr/>	<code>\int_incr:N <integer></code>
	Increases the value stored in <i><integer></i> by 1 globally (<i>i.e.</i> not limited by the current group level).
<hr/> <code>\int_set:Nn</code> <hr/> <code>\int_set:cn</code> <hr/>	<code>\int_set:Nn <integer> {<integer expression>}</code>
Updated: 2011-10-22	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>). This assignment is restricted to the current TeX group.
<hr/> <code>\int_gset:Nn</code> <hr/> <code>\int_gset:cn</code> <hr/>	<code>\int_gset:Nn <integer> {<integer expression>}</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>). This assignment is global and is not limited to the current TeX group level.
<hr/> <code>\int_sub:Nn</code> <hr/> <code>\int_sub:cn</code> <hr/>	<code>\int_sub:Nn <integer> {<integer expression>}</code>
Updated: 2011-10-22	Subtracts the result of the <i><integer expression></i> to the current content of the <i><integer></i> . This assignment is local.
<hr/> <code>\int_gsub:Nn</code> <hr/> <code>\int_gsub:cn</code> <hr/>	<code>\int_gsub:Nn <integer> {<integer expression>}</code>
	Subtracts the result of the <i><integer expression></i> to the current content of the <i><integer></i> . This assignment is global.

46 Using integers

<code>\int_use:N</code>	★	<code>\int_use:N</code> $\langle integer \rangle$
-------------------------	---	---

<code>\int_use:c</code>	★	
-------------------------	---	--

Updated: 2011-10-22		
---------------------	--	--

Recovers the content of a $\langle integer \rangle$ 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 $\langle integer \rangle$ is required (such as in the first and third arguments of `\int_compare:nNnTF`).

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</code> $\{\langle intexpr_1 \rangle\}$ $\langle relation \rangle$ $\{\langle intexpr_2 \rangle\}$
---------------------------------	---	--

<code>\int_compare:nNnTF</code>	★	<code>\int_compare:nNnTF</code> $\{\langle intexpr_1 \rangle\}$ $\langle relation \rangle$ $\{\langle intexpr_2 \rangle\}$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
---------------------------------	---	---

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

Equal	=
Greater than	>
Less than	<

<code>\int_compare_p:n</code>	★	<code>\int_compare_p:n</code> $\{\langle intexpr_1 \rangle \langle relation \rangle \langle intexpr_2 \rangle\}$
-------------------------------	---	--

<code>\int_compare:nTF</code>	★	<code>\int_compare:nTF</code> $\{\langle intexpr_1 \rangle \langle relation \rangle \langle intexpr_2 \rangle\}$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
-------------------------------	---	---

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

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</code> $\{\langle integer expression \rangle\}$
-------------------------------	---	---

<code>\int_if_even:nTF</code>	★	<code>\int_if_odd:nTF</code> $\{\langle integer expression \rangle\}$
-------------------------------	---	---

<code>\int_if_odd_p:n</code>	★	$\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
------------------------------	---	--

<code>\int_if_odd:nTF</code>	★	
------------------------------	---	--

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

48 Integer expression loops

<hr/> <code>\int_do_while:nNnn</code> ☆ <hr/>	<pre>\int_do_while:nNnn {\intexpr_1} <relation> {\intexpr_2} {\code}</pre> <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 true. After the <i><code></i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is false.</p>
<hr/> <code>\int_do_until:nNnn</code> ☆ <hr/>	<pre>\int_do_until:nNnn {\intexpr_1} <relation> {\intexpr_2} {\code}</pre> <p>Evaluates the relationship between the two <i><integer expressions></i> as described for <code>\int_compare:nNnTF</code>, and then places the <i><code></i> in the input stream if the <i><relation></i> is false. After the <i><code></i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is true.</p>
<hr/> <code>\int_until_do:nNnn</code> ☆ <hr/>	<pre>\int_until_do:nNnn {\intexpr_1} <relation> {\intexpr_2} {\code}</pre> <p>Places the <i><code></i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i><integer expressions></i> as described for <code>\int_compare:nNnTF</code>. If the test is false then the <i><code></i> will be inserted into the input stream again and a loop will occur until the <i><relation></i> is true.</p>
<hr/> <code>\int_while_do:nNnn</code> ☆ <hr/>	<pre>\int_while_do:nNnn {\intexpr_1} <relation> {\intexpr_2} {\code}</pre> <p>Places the <i><code></i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i><integer expressions></i> as described for <code>\int_compare:nNnTF</code>. If the test is true then the <i><code></i> will be inserted into the input stream again and a loop will occur until the <i><relation></i> is false.</p>
<hr/> <code>\int_do_while:nn</code> ☆ <hr/>	<pre>\int_do_while:nn { <intexpr1> <relation> <intexpr2> } {\code}</pre> <p>Evaluates the relationship between the two <i><integer expressions></i> as described for <code>\int_compare:nTF</code>, and then places the <i><code></i> in the input stream if the <i><relation></i> is true. After the <i><code></i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is false.</p>
<hr/> <code>\int_do_until:nn</code> ☆ <hr/>	<pre>\int_do_until:nn { <intexpr1> <relation> <intexpr2> } {\code}</pre> <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>

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

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

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

Places the $\langle code \rangle$ in the input stream for T_EX to process, and then evaluates the relationship between the two $\langle integer\ expressions \rangle$ as described for `\int_compare:nTF`. If the test is `true` then the $\langle code \rangle$ will be inserted into the input stream again and a loop will occur until the $\langle relation \rangle$ 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 $\langle integer\ expression \rangle$ in the input stream as digits, with category code 12 (other).

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

Updated: 2011-09-17

Evaluates the $\langle integer\ expression \rangle$ 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

`\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_binary: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.*

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

<code>\int_to_Roman:n</code> ☆

Updated: 2011-10-22

Places the value of the *<integer expression>* in the input stream as Roman numerals, either lower case (`\int_to_roman:n`) or upper case (`\int_to_Roman:n`). The Roman numerals are letters with category code 11 (letter).

50 Converting from other formats to integers

<code>\int_from_alph:n</code> ☆	<code>\int_from_alph:n</code> { <i><letters></i> }
---------------------------------	--

Converts the *<letters>* into the integer (base 10) representation and leaves this in the input stream. The *<letters>* 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 `\int_to_alph:n`.

<code>\int_from_binary:n</code> ☆	<code>\int_from_binary:n</code> { <i><binary number></i> }
-----------------------------------	--

Converts the *<binary number>* into the integer (base 10) representation and leaves this in the input stream.

<code>\int_from_hexadecimal:n</code> ☆	<code>\int_from_hexadecimal:n</code> { <i><hexadecimal number></i> }
--	--

Converts the *<hexadecimal number>* into the integer (base 10) representation and leaves this in the input stream. Digits greater than 9 may be represented in the *<hexadecimal number>* by upper or lower case letters.

<code>\int_from_octal:n</code> ☆	<code>\int_from_octal:n</code> { <i><octal number></i> }
----------------------------------	--

Converts the *<octal number>* into the integer (base 10) representation and leaves this in the input stream.

<code>\int_from_roman:n</code> ☆	<code>\int_from_roman:n</code> { <i><roman numeral></i> }
----------------------------------	---

Converts the *<roman numeral>* into the integer (base 10) representation and leaves this in the input stream. The *<roman numeral>* may be in upper or lower case; if the numeral is not valid then the resulting value will be -1 .

<code>\int_from_base:nn</code> ☆	<code>\int_from_base:nn</code> { <i><number></i> } { <i><base></i> }
----------------------------------	--

Converts the *<number>* in *<base>* into the appropriate value in base 10. The *<number>* should consist of digits and letters (either lower or upper case), plus optionally a leading sign. The maximum *<base>* value is 36.

51 Viewing integers

<code>\int_show:N</code>	<code>\int_show:N</code> <i><integer></i>
--------------------------	---

<code>\int_show:c</code>

Displays the value of the *<integer>* on the terminal.

52 Constant integers

`\c_minus_one`
`\c_zero`
`\c_one`
`\c_two`
`\c_three`
`\c_four`
`\c_five`
`\c_six`
`\c_seven`
`\c_eight`
`\c_nine`
`\c_ten`
`\c_eleven`
`\c_twelve`
`\c_thirteen`
`\c_fourteen`
`\c_fifteen`
`\c_sixteen`
`\c_thirty_two`
`\c_one_hundred`
`\c_two_hundred_fifty_five`
`\c_two_hundred_fifty_six`
`\c_one_thousand`
`\c_ten_thousand`

Integer values used with primitive tests and assignments: self-terminating nature makes these more convenient and faster than literal numbers.

`\c_max_int`

The maximum value that can be stored as an integer.

`\c_max_register_int`

Maximum number of registers.

53 Scratch integers

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

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

`\g_tmpa_int`
`\g_tmpb_int`

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

54 Internal functions

<hr/> <code>\int_get_digits:n</code> ★ <hr/>	<code>\int_get_digits:n <value></code> 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/> <code>\int_get_sign:n</code> ☆ <hr/>	<code>\int_get_sign:n <value></code> 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/> <code>\int_to_letter:n</code> ★ <hr/> <div>Updated: 2011-09-17</div>	<code>\int_to_letter:n <integer value></code> 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/> <code>\int_to_roman:w</code> ★ <hr/>	<code>\int_to_roman:w <integer> <space> or <non-expandable token></code> 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.
TeXhackers note: This is the TeX primitive <code>\romannumeral</code> renamed.	
<hr/> <code>\if_num:w</code> ★ <code>\if_int_compare:w</code> ★ <hr/>	<code>\if_num:w <integer1> <relation> <integer2></code> <code><true code></code> <code>\else:</code> <code><false code></code> <code>\fi:</code> Compare two integers using $\langle relation \rangle$, which must be one of =, < or > with category code 12. The <code>\else:</code> branch is optional.

TeXhackers note: These are both names for the TeX primitive `\ifnum`.

<code>\if_case:w</code>	★	<code>\if_case:w</code>	$\langle integer \rangle$	$\langle case0 \rangle$
<code>\or</code>	★	<code>\or:</code>	$\langle case1 \rangle$	
		<code>\or:</code>	\dots	
		<code>\else:</code>	$\langle default \rangle$	
		<code>\fi:</code>		

Selects a case to execute based on the value of the $\langle integer \rangle$. The first case ($\langle case0 \rangle$) is executed if $\langle integer \rangle$ is 0, the second ($\langle case1 \rangle$) if the $\langle integer \rangle$ is 1, *etc.* The $\langle integer \rangle$ 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</code>	$\langle integer \rangle$
		<code>\int_value:w</code>	$\langle tokens \rangle$ $\langle optional space \rangle$

Expands $\langle tokens \rangle$ until an $\langle integer \rangle$ 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</code>	$\langle intexpr \rangle$	<code>\int_eval_end:</code>
<code>\int_eval_end</code>	★			

Evaluates $\langle integer expression \rangle$ 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</code>	$\langle tokens \rangle$	$\langle optional space \rangle$
			$\langle true code \rangle$	
		<code>\else:</code>		
			$\langle true code \rangle$	
		<code>\fi:</code>		

Expands $\langle tokens \rangle$ until a non-numeric token or a space is found, and tests whether the resulting $\langle integer \rangle$ is odd. If so, $\langle true code \rangle$ 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` $\langle dimension \rangle$

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_zero:N` $\langle dimension \rangle$

Sets $\langle dimension \rangle$ to 0pt within the scope of the current T_EX group.

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

`\dim_gzero:N` $\langle dimension \rangle$

Sets $\langle dimension \rangle$ to 0pt globally, *i.e.* not restricted by the current T_EX group level.

56 Setting dim variables

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

`\dim_add:Nn` $\langle dimension \rangle$ $\{\langle dimension expression \rangle\}$

Adds the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$. This assignment is local.

Updated: 2011-10-22

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

`\dim_gadd:Nn` $\langle dimension \rangle$ $\{\langle dimension expression \rangle\}$

Adds the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$. This assignment is global.

`\dim_set:Nn`
`\dim_set:cn`

`\dim_set:Nn` $\langle dimension \rangle$ $\{\langle dimension expression \rangle\}$

Sets $\langle dimension \rangle$ to the value of $\langle dimension expression \rangle$, which must evaluate to a length with units. This assignment is restricted to the current T_EX group.

Updated: 2011-10-22

<hr/> <code>\dim_gset:Nn</code> <code>\dim_gset:cn</code> <hr/>	<code>\dim_gset:Nn <dimension> {<dimension expression>}</code> Sets $\langle dimension \rangle$ to the value of $\langle dimension expression \rangle$, which must evaluate to a length with units and may include a rubber component (for example 1 cm plus 0.5 cm. This assignment is global and is not limited to the current T _E X group level.
<hr/> <code>\dim_set_eq:NN</code> <code>\dim_set_eq:(cN Nc cc)</code> <hr/>	<code>\dim_set_eq:NN <dimension1> <dimension2></code> Sets the content of $\langle dimension1 \rangle$ equal to that of $\langle dimension2 \rangle$. This assignment is restricted to the current T _E X group level.
<hr/> <code>\dim_gset_eq:NN</code> <code>\dim_gset_eq:(cN Nc cc)</code> <hr/>	<code>\dim_gset_eq:NN <dimension1> <dimension2></code> Sets the content of $\langle dimension1 \rangle$ equal to that of $\langle dimension2 \rangle$. This assignment is global and so is not limited by the current T _E X group level.
<hr/> <code>\dim_set_max:Nn</code> <code>\dim_set_max:cn</code> Updated: 2011-10-22 <hr/>	<code>\dim_set_max:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the larger of these two value. This assignment is local to the current T _E X group.
<hr/> <code>\dim_gset_max:Nn</code> <code>\dim_gset_max:cn</code> Updated: 2011-10-22 <hr/>	<code>\dim_gset_max:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the larger of these two value. This assignment is global.
<hr/> <code>\dim_set_min:Nn</code> <code>\dim_set_min:cn</code> Updated: 2011-10-22 <hr/>	<code>\dim_set_min:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the smaller of these two value. This assignment is local to the current T _E X group.
<hr/> <code>\dim_gset_min:Nn</code> <code>\dim_gset_min:cn</code> Updated: 2011-10-22 <hr/>	<code>\dim_gset_min:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the smaller of these two value. This assignment is global.
<hr/> <code>\dim_sub:Nn</code> <code>\dim_sub:cn</code> Updated: 2011-10-22 <hr/>	<code>\dim_sub:Nn <dimension> {<dimension expression>}</code> Subtracts the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$. This assignment is local.
<hr/> <code>\dim_gsub:Nn</code> <code>\dim_gsub:cn</code> <hr/>	<code>\dim_gsub:Nn <dimension> {<dimension expression>}</code> Subtracts the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$. This assignment is global.

57 Utilities for dimension calculations

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

<hr/> <code>\dim_ratio:nn</code> ★	<code>\dim_ratio:nn {⟨dimexpr₁⟩} {⟨dimexpr₂⟩}</code>
<hr/> Updated: 2011-10-22	Parses the two <i>⟨dimension expressions⟩</i> and converts the ratio of the two to a form suitable for use inside a <i>⟨dimension expression⟩</i> . 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

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

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

Equal	=
Greater than	>
Less than	<

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

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

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

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**.

<hr/> <code>\dim_do_while:nn</code> ☆ <hr/>	<pre>\dim_do_while:nNnn { <dimexpr1> <relation> <dimexpr2> } {<code>}</pre> <p>Evaluates the relationship between the two <i><dimension expressions></i> as described for <code>\dim_compare:nTF</code>, and then places the <i><code></i> in the input stream if the <i><relation></i> is true. After the <i><code></i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is false.</p>
<hr/> <code>\dim_do_until:nn</code> ☆ <hr/>	<pre>\dim_do_until:nn { <dimexpr1> <relation> <dimexpr2> } {<code>}</pre> <p>Evaluates the relationship between the two <i><dimension expressions></i> as described for <code>\dim_compare:nTF</code>, and then places the <i><code></i> in the input stream if the <i><relation></i> is false. After the <i><code></i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is true.</p>
<hr/> <code>\dim_until_do:nn</code> ☆ <hr/>	<pre>\dim_until_do:nn { <dimexpr1> <relation> <dimexpr2> } {<code>}</pre> <p>Places the <i><code></i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i><dimension expressions></i> as described for <code>\dim_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>
<hr/> <code>\dim_while_do:nn</code> ☆ <hr/>	<pre>\dim_while_do:nn { <dimexpr1> <relation> <dimexpr2> } {<code>}</pre> <p>Places the <i><code></i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i><dimension expressions></i> as described for <code>\dim_compare:nTF</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>

60 Using dim expressions and variables

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

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

<u><code>\dim_show:N</code></u>	<code>\dim_show:N</code> $\langle dimension \rangle$
<u><code>\dim_show:c</code></u>	Displays the value of the $\langle dimension \rangle$ on the terminal.

62 Constant dimensions

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

63 Scratch dimensions

<u><code>\l_tmpa_dim</code></u>	Scratch dimension 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.
<u><code>\l_tmpb_dim</code></u>	
<u><code>\l_tmpc_dim</code></u>	
<u><code>\g_tmpa_dim</code></u>	Scratch dimension 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.
<u><code>\g_tmpb_dim</code></u>	

64 Creating and initialising skip variables

<u><code>\skip_new:N</code></u>	<code>\skip_new:N</code> $\langle skip \rangle$
<u><code>\skip_new:c</code></u>	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 0pt.
<u><code>\skip_zero:N</code></u>	<code>\skip_zero:N</code> $\langle skip \rangle$
<u><code>\skip_zero:c</code></u>	Sets $\langle skip \rangle$ to 0pt within the scope of the current T _E X group.
<u><code>\skip_gzero:N</code></u>	<code>\skip_gzero:N</code> $\langle skip \rangle$
<u><code>\skip_gzero:c</code></u>	Sets $\langle skip \rangle$ to 0pt globally, <i>i.e.</i> not restricted by the current T _E X group level.

65 Setting skip variables

<hr/> <code>\skip_add:Nn</code> <code>\skip_add:cn</code> <hr/> <small>Updated: 2011-10-22</small>	<code>\skip_add:Nn <skip> {<skip expression>}</code> Adds the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is local.
<hr/> <code>\skip_gadd:Nn</code> <code>\skip_gadd:cn</code> <hr/>	<code>\skip_gadd:Nn <skip> {<skip expression>}</code> Adds the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is global.
<hr/> <code>\skip_set:Nn</code> <code>\skip_set:cn</code> <hr/> <small>Updated: 2011-10-22</small>	<code>\skip_set:Nn <skip> {<skip expression>}</code> Sets <i><skip></i> to the value of <i><skip expression></i> , which must evaluate to a length with units and may include a rubber component (for example 1 cm plus 0.5 cm. This assignment is restricted to the current T _E X group.
<hr/> <code>\skip_gset:Nn</code> <code>\skip_gset:cn</code> <hr/>	<code>\skip_gset:Nn <skip> {<skip expression>}</code> Sets <i><skip></i> to the value of <i><skip expression></i> , which must evaluate to a length with units and may include a rubber component (for example 1 cm plus 0.5 cm. This assignment is global and is not limited to the current T _E X group level.
<hr/> <code>\skip_set_eq:NN</code> <code>\skip_set_eq:(cN Nc cc)</code> <hr/>	<code>\skip_set_eq:NN <skip1> <skip2></code> Sets the content of <i><skip1></i> equal to that of <i><skip2></i> . This assignment is restricted to the current T _E X group level.
<hr/> <code>\skip_gset_eq:NN</code> <code>\skip_gset_eq:(cN Nc cc)</code> <hr/>	<code>\skip_gset_eq:NN <skip1> <skip2></code> Sets the content of <i><skip1></i> equal to that of <i><skip2></i> . This assignment is global and so is not limited by the current T _E X group level.
<hr/> <code>\skip_sub:Nn</code> <code>\skip_sub:cn</code> <hr/> <small>Updated: 2011-10-22</small>	<code>\skip_sub:Nn <skip> {<skip expression>}</code> Subtracts the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is local.
<hr/> <code>\skip_gsub:Nn</code> <code>\skip_gsub:cn</code> <hr/>	<code>\skip_gsub:Nn <skip> {<skip expression>}</code> Subtracts the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is global.

66 Skip expression conditionals

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

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

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

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

67 Using skip expressions and variables

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

Updated: 2011-10-22

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

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

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

T_EXhackers note: `\skip_use:N` is the T_EX 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 <skip></code>
<code>\skip_show:c</code>	

Displays the value of the *<skip>* 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> <code>\l_tmpb_skip</code> <code>\l_tmpc_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>\g_tmpa_skip</code> <code>\g_tmpb_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.
--	---

71 Creating and initialising muskip variables

<code>\muskip_new:N</code> <code>\muskip_new:c</code>	<code>\muskip_new:N</code> $\langle muskip \rangle$ 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.
--	---

<code>\muskip_zero:N</code> <code>\muskip_zero:c</code>	<code>\skip_zero:N</code> $\langle muskip \rangle$ Sets $\langle muskip \rangle$ to 0 mu within the scope of the current T _E X group.
--	---

<code>\muskip_gzero:N</code> <code>\muskip_gzero:c</code>	<code>\muskip_gzero:N</code> $\langle muskip \rangle$ Sets $\langle muskip \rangle$ to 0 mu globally, <i>i.e.</i> not restricted by the current T _E X group level.
--	--

72 Setting muskip variables

<code>\muskip_add:Nn</code> <code>\muskip_add:cn</code>	<code>\muskip_add:Nn</code> $\langle muskip \rangle$ $\{ \langle muskip \text{ expression} \rangle \}$ Adds the result of the $\langle muskip \text{ expression} \rangle$ to the current content of the $\langle muskip \rangle$. This assignment is local.
--	---

Updated: 2011-10-22

<code>\muskip_gadd:Nn</code>	<code>\muskip_gadd:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_gadd:cn</code>	Adds the result of the $\langle muskip expression \rangle$ to the current content of the $\langle muskip \rangle$. This assignment is global.

<code>\muskip_set:Nn</code>	<code>\muskip_set:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_set:cn</code>	Sets $\langle muskip \rangle$ to the value of $\langle muskip expression \rangle$, which must evaluate to a math length with units and may include a rubber component (for example 1 mu plus 0.5 mu. This assignment is restricted to the current T _E X group.

Updated: 2011-10-22

<code>\muskip_gset:Nn</code>	<code>\muskip_gset:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_gset:cn</code>	Sets $\langle muskip \rangle$ to the value of $\langle muskip expression \rangle$, which must evaluate to a math length with units and may include a rubber component (for example 1 mu plus 0.5 mu. This assignment is global and is not limited to the current T _E X group level.

<code>\muskip_set_eq:NN</code>	<code>\muskip_set_eq:NN <muskip1> <muskip2></code>
<code>\muskip_set_eq:(cN Nc cc)</code>	Sets the content of $\langle muskip1 \rangle$ equal to that of $\langle muskip2 \rangle$. This assignment is restricted to the current T _E X group level.

<code>\muskip_gset_eq:NN</code>	<code>\muskip_gset_eq:NN <muskip1> <muskip2></code>
<code>\muskip_gset_eq:(cN Nc cc)</code>	Sets the content of $\langle muskip1 \rangle$ equal to that of $\langle muskip2 \rangle$. This assignment is global and so is not limited by the current T _E X group level.

<code>\muskip_sub:Nn</code>	<code>\muskip_sub:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_sub:cn</code>	Subtracts the result of the $\langle muskip expression \rangle$ to the current content of the $\langle skip \rangle$. This assignment is local.

Updated: 2011-10-22

<code>\muskip_gsub:Nn</code>	<code>\muskip_gsub:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_gsub:cn</code>	Subtracts the result of the $\langle muskip expression \rangle$ to the current content of the $\langle muskip \rangle$. This assignment is global.

73 Using muskip expressions and variables

<code>\muskip_eval:n</code> ★	<code>\muskip_eval:n {<muskip expression>}</code>
Updated: 2011-10-22	Evaluates the $\langle muskip expression \rangle$, expanding any skips and token list variables within the $\langle expression \rangle$ to their content (without requiring <code>\muskip_use:N/\tl_use:N</code>) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a $\langle muglue denotation \rangle$ after two expansions. This will be expressed in mu, and will require suitable termination if used in a T _E X-style assignment as it is <i>not</i> an $\langle internal muglue \rangle$.

<code>\muskip_use:N</code>	★	<code>\muskip_use:N</code>	<code>\muskip</code>
<code>\muskip_use:c</code>	★		

Recovers the content of a `\muskip` 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 `\muskip_eval:n`).

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

<code>\skip_horizontal:N</code>	<code>\skip_horizontal:N</code>	<code>\skip</code>
<code>\skip_horizontal:(c n)</code>	<code>\skip_horizontal:n</code>	<code>{\skipexpr}</code>

Updated: 2011-10-22

Inserts a horizontal `\skip` into the current list.

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

<code>\skip_vertical:N</code>	<code>\skip_vertical:N</code>	<code>\skip</code>
<code>\skip_vertical:(c n)</code>	<code>\skip_vertical:n</code>	<code>{\skipexpr}</code>

Updated: 2011-10-22

Inserts a vertical `\skip` into the current list.

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

75 Viewing muskip variables

<code>\muskip_show:N</code>	<code>\muskip_show:N</code>	<code>\muskip</code>
<code>\muskip_show:c</code>		

Displays the value of the `\muskip` on the terminal.

76 Internal functions

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

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

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

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

Evaluates $\langle dimension\ expression \rangle$ 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.

TeXhackers note: This is the ε -TeX primitive `\dimexpr`.

77 Experimental skip functions

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

Updated: 2011-10-22

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

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

If the $\{<dimension\ expression>\}$ 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_clear:N <tl var>
```

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

<code>\tl_gclear:N</code>	<code>\tl_gclear:N <tl var></code>
<code>\tl_gclear:c</code>	Clears all entries from the $\langle tl\ var\rangle$ globally.

<code>\tl_clear_new:N</code>	<code>\tl_clear_new:N <tl var></code>
<code>\tl_clear_new:c</code>	If the $\langle tl\ var\rangle$ already exists, clears it within the scope of the current T _E X group. If the $\langle tl\ var\rangle$ is not defined, it will be created (using <code>\tl_new:N</code>). Thus the sequence is guaranteed to be available and clear within the current T _E X group. The $\langle tl\ var\rangle$ will exist globally, but the content outside of the current T _E X group is not specified.

<code>\tl_gclear_new:N</code>	<code>\tl_gclear_new:N <tl var></code>
<code>\tl_gclear_new:c</code>	If the $\langle tl\ var\rangle$ already exists, clears it globally. If the $\langle tl\ var\rangle$ is not defined, it will be created (using <code>\tl_new:N</code>). Thus the sequence is guaranteed to be available and globally clear.

<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 $\langle tl\ var1\rangle$ equal to that of $\langle tl\ var2\rangle$. This assignment is restricted to the current T _E X group level.

<code>\tl_gset_eq:NN</code>	<code>\tl_gset_eq:NN <tl var1> <tl var2></code>
<code>\tl_gset_eq:(cN Nc cc)</code>	Sets the content of $\langle tl\ var1\rangle$ equal to that of $\langle tl\ var2\rangle$. This assignment is global and so is not limited by the current T _E X group level.

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>	Sets $\langle tl\ var\rangle$ to contain $\langle tokens\rangle$, removing any previous content from the variable. This assignment is restricted to the current T _E X group.

<code>\tl_gset:Nn</code>	<code>\tl_gset:Nn <tl var> {\tokens}</code>
<code>\tl_gset:(NV Nv No Nf Nx cn cV cv co cf cx)</code>	Sets $\langle tl\ var\rangle$ to contain $\langle tokens\rangle$, removing any previous content from the variable. This assignment is global and is not limited to the current T _E X group level.

<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>	Appends $\langle tokens\rangle$ to the left side of the current content of $\langle tl\ var\rangle$. This modification is restricted to the current T _E X group level.

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

Globally appends *<tokens>* to the left side of the current content of *<tl var>*. This modification is not limited by T_EX grouping.

<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>	

Appends *<tokens>* to the right side of the current content of *<tl var>*. This modification is restricted to the current T_EX group level.

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

Globally appends *<tokens>* to the right side of the current content of *<tl var>*. This modification is not limited by T_EX grouping.

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>	

Updated: 2011-08-11

Replaces the first (leftmost) occurrence of *<old tokens>* in the *<tl var>* with *<new tokens>*. *<Old tokens>* cannot contain `{`, `}` or `#` (assuming normal T_EX category codes). The assignment is restricted to the current T_EX group.

<code>\tl_greplace_once:Nnn</code>	<code>\tl_greplace_once:Nnn <tl var> {<old tokens>} {<new tokens>}</code>
<code>\tl_greplace_once:cnn</code>	

Updated: 2011-08-11

Replaces the first (leftmost) occurrence of *<old tokens>* in the *<tl var>* with *<new tokens>*. *<Old tokens>* cannot contain `{`, `}` or `#` (assuming normal T_EX category codes). The assignment is applied globally.

<code>\tl_replace_all:Nnn</code>	<code>\tl_replace_all:Nnn <tl var> {<old tokens>} {<new tokens>}</code>
<code>\tl_replace_all:cnn</code>	

Updated: 2011-08-11

Replaces all occurrences of *<old tokens>* in the *<tl var>* with *<new tokens>*. *<Old tokens>* cannot contain `{`, `}` or `#` (assuming normal T_EX category codes). As this function operates from left to right, the pattern *<old tokens>* may remain after the replacement (see `\tl_remove_all:Nn` for an example). The assignment is restricted to the current T_EX group.

<code>\tl_greplace_all:Nnn</code>	<code>\tl_greplace_all:Nnn <tl var> {<old tokens>} {<new tokens>}</code>
<code>\tl_greplace_all:cnn</code>	

Updated: 2011-08-11

Replaces all occurrences of *<old tokens>* in the *<tl var>* with *<new tokens>*. *<Old tokens>* cannot contain `{`, `}` or `#` (assuming normal T_EX category codes). As this function operates from left to right, the pattern *<old tokens>* may remain after the replacement (see `\tl_remove_all:Nn` for an example). The assignment is applied globally.

<hr/> <code>\tl_remove_once:Nn</code> <code>\tl_remove_once:cn</code> <hr/> Updated: 2011-08-11 <hr/>	<code>\tl_remove_once:Nn <tl var> {<tokens>}</code> Removes the first (leftmost) occurrence of <code><tokens></code> from the <code><tl var></code> . <code><Tokens></code> cannot contain <code>{, }</code> or <code>#</code> (assuming normal T _E X category codes). The assignment is restricted to the current T _E X group.
<hr/> <code>\tl_gremove_once:Nn</code> <code>\tl_gremove_once:cn</code> <hr/> Updated: 2011-08-11 <hr/>	<code>\tl_gremove_once:Nn <tl var> {<tokens>}</code> Removes the first (leftmost) occurrence of <code><tokens></code> from the <code><tl var></code> . <code><Tokens></code> cannot contain <code>{, }</code> or <code>#</code> (assuming normal T _E X category codes). The assignment is applied globally.
<hr/> <code>\tl_remove_all:Nn</code> <code>\tl_remove_all:cn</code> <hr/> Updated: 2011-08-11 <hr/>	<code>\tl_remove_all:Nn <tl var> {<tokens>}</code> Removes all occurrences of <code><tokens></code> from the <code><tl var></code> . <code><Tokens></code> cannot contain <code>{, }</code> or <code>#</code> (assuming normal T _E X category codes). As this function operates from left to right, the pattern <code><tokens></code> may remain after the removal, for instance, <code>\tl_set:Nn \l_tmpa_tl {abbccd} \tl_remove_all:Nn \l_tmpa_tl {bc}</code> will result in <code>\l_tmpa_tl</code> containing <code>abcd</code> . The assignment is restricted to the current T _E X group.
<hr/> <code>\tl_gremove_all:Nn</code> <code>\tl_gremove_all:cn</code> <hr/> Updated: 2011-08-11 <hr/>	<code>\tl_gremove_all:Nn <tl var> {<tokens>}</code> Removes all occurrences of <code><tokens></code> from the <code><tl var></code> . <code><Tokens></code> cannot contain <code>{, }</code> or <code>#</code> (assuming normal T _E X category codes). As this function operates from left to right, the pattern <code><tokens></code> may remain after the removal (see <code>\tl_remove_all:Nn</code> for an example). The assignment is applied globally.

82 Reassigning token list category codes

<hr/> <code>\tl_set_rescan:Nnn</code> <code>\tl_set_rescan:(Nno Nnx cnn cno cnx)</code> <hr/> Updated: 2011-08-11 <hr/>	<code>\tl_set_rescan:Nnn <tl var> {<setup>} {<tokens>}</code> Sets <code><tl var></code> to contain <code><tokens></code> , applying the category code régime specified in the <code><setup></code> before carrying out the assignment. This allows the <code><tl var></code> to contain material with category codes other than those that apply when <code><tokens></code> are absorbed. The assignment is local to the current T _E X group. See also <code>\tl_rescan:nn</code> .
---	--

```
\tl_gset_rescan:Nnn
\tl_gset_rescan:(Nno|Nnx|cnn|cno|cnx)
```

Updated: 2011-08-11

```
\tl_gset_rescan:Nnn <tl var> {\<setup>} {\<tokens>}
```

Sets $\langle tl\ var\rangle$ to contain $\langle tokens\rangle$, applying the category code régime specified in the $\langle setup\rangle$ before carrying out the assignment. This allows the $\langle tl\ var\rangle$ to contain material with category codes other than those that apply when $\langle tokens\rangle$ are absorbed. The assignment is global. See also `\tl_rescan:nn`.

```
\tl_rescan:nn
```

Updated: 2011-08-11

```
\tl_rescan:nn {\<setup>} {\<tokens>}
```

Rescans $\langle tokens\rangle$ applying the category code régime specified in the $\langle setup\rangle$, and leaves the resulting tokens in the input stream. See also `\tl_set_rescan:Nnn`.

83 Reassigning token list character codes

```
\tl_to_lowercase:n
```

```
\tl_to_lowercase:n {\<tokens>}
```

Works through all of the $\langle tokens\rangle$, 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 $\langle tokens\rangle$.

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 $\langle tokens\rangle$, 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 $\langle tokens\rangle$.

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:(V|o) ★
\tl_if_blank:nTF ★
\tl_if_blank:(V|o)TF ★
```

```
\tl_if_blank_p:n {\<token list>}
```

```
\tl_if_blank:nTF {\<token list>} {\<true code>} {\<false code>}
```

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

<code>\tl_if_empty_p:N</code>	★	<code>\tl_if_empty_p:N <tl var></code>
<code>\tl_if_empty_p:c</code>	★	<code>\tl_if_empty:NTF <tl var> {\true code} {\false code}</code>
<code>\tl_if_empty:NTF</code>	★	Tests if the <i><token list variable></i> is entirely empty (<i>i.e.</i> contains no tokens at all).
<code>\tl_if_empty:cTF</code>	★	

<code>\tl_if_empty_p:n</code>	★	<code>\tl_if_empty_p:n {\token list}</code>
<code>\tl_if_empty_p:(V o)</code>	★	<code>\tl_if_empty:nTF {\token list} {\true code} {\false code}</code>
<code>\tl_if_empty:nTF</code>	★	Tests if the <i><token list></i> is entirely empty (<i>i.e.</i> contains no tokens at all).
<code>\tl_if_empty:(V o)TF</code>	★	

<code>\tl_if_eq_p:NN</code>	★	<code>\tl_if_eq_p:NN {\tl var₁} {\tl var₂}</code>
<code>\tl_if_eq_p:(Nc cN cc)</code>	★	<code>\tl_if_eq:NNTF {\tl var₁} {\tl var₂} {\true code} {\false code}</code>
<code>\tl_if_eq:NNTF</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
<code>\tl_if_eq:(Nc cN cc)TF</code>	★	

```

\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:NTF {\tl var} {\true code} {\false code}</code>
<code>\tl_if_single:NTF</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> .
<code>\tl_if_single:cTF</code>	★	

Updated: 2011-08-13

`\tl_if_single_p:n` ☆
`\tl_if_single:nTF` ☆

Updated: 2011-08-13

`\tl_if_single_p:n` $\{ \langle token\ list \rangle \}$
`\tl_if_single:nTF` $\{ \langle token\ list \rangle \} \{ \langle true\ code \rangle \} \{ \langle false\ code \rangle \}$

Tests if the token list has exactly one item, *i.e.* 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 `\tl_length:n`.

`\tl_if_single_token_p:n` ☆
`\tl_if_single_token:nTF` ☆

New: 2011-08-13

`\tl_if_single_token_p:n` $\{ \langle token\ list \rangle \}$
`\tl_if_single_token:nTF` $\{ \langle token\ list \rangle \} \{ \langle true\ code \rangle \} \{ \langle false\ code \rangle \}$

Tests if the token list consists of exactly one token, *i.e.* is either a single space character or a single “normal” token. Token groups $\{ \dots \}$ are not single tokens.

85 Mapping to token lists

`\tl_map_function:NN` ☆
`\tl_map_function:cN` ☆

`\tl_map_function:NN` $\langle tl\ var \rangle \langle function \rangle$

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 `\tl_map_function:nN`.

`\tl_map_function:nN` ☆

`\tl_map_function:nN` $\langle token\ list \rangle \langle function \rangle$

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 `\tl_map_function:NN`.

`\tl_map_inline:Nn`
`\tl_map_inline:cn`

`\tl_map_inline:Nn` $\langle tl\ var \rangle \{ \langle inline\ function \rangle \}$

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 `\tl_map_function:Nn`.

`\tl_map_inline:nn`

`\tl_map_inline:nn` $\langle token\ list \rangle \{ \langle inline\ function \rangle \}$

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 `\tl_map_function:nn`.

`\tl_map_variable:NNn`
`\tl_map_variable:cNn`

`\tl_map_variable:NNn` $\langle tl\ var \rangle \langle variable \rangle \{ \langle function \rangle \}$

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 `\tl_map_inline:Nn`.

<code>\tl_map_variable:nNn</code>	<code>\tl_map_variable:nNn <token list> <variable> {<function>}</code> Applies the <i><function></i> to every <i><item></i> stored within the <i><token list></i> . The <i><function></i> should consist of code which will receive the <i><item></i> stored in the <i><variable></i> . One variable mapping can be nested inside another. See also <code>\tl_map_inline:nn</code> .
-----------------------------------	---

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

```

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

```

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

86 Using token lists

<code>\tl_to_str:N</code> ☆	<code>\tl_to_str:N <tl var></code>
<code>\tl_to_str:c</code> ☆	

Converts the content of the *<tl var>* into a series of characters with category code 12 (other) with the exception of spaces, which retain category code 10 (space). This *<string>* is then left in the input stream.

<code>\tl_to_str:n</code> ☆	<code>\tl_to_str:n {<tokens>}</code>
-----------------------------	--

Converts the given *<tokens>* into a series of characters with category code 12 (other) with the exception of spaces, which retain category code 10 (space). This *<string>* is then left in the input stream. Note that this function requires only a single expansion.

TeXhackers note: This is the ε -TeX primitive `\detokenize`.

<code>\tl_use:N</code> ☆	<code>\tl_use:N <tl var></code>
<code>\tl_use:c</code> ☆	

Recovers the content of a *<tl var>* 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 *<tl var>* directly without an accessor function.

87 Working with the content of token lists

<hr/> <code>\tl_length:n</code> ★	<code>\tl_length:n {\tokens}</code>
<code>\tl_length:(V o)</code> ★	
<hr/> Updated: 2011-08-13 <hr/>	Counts the number of $\langle items \rangle$ in $\langle tokens \rangle$ 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 <code>\tl_length:N</code> . This function requires three expansions, giving an $\langle integer denotation \rangle$.
<hr/> <code>\tl_length:N</code> ★	<code>\tl_length:N {\tl var}</code>
<code>\tl_length:c</code> ★	
<hr/> Updated: 2011-08-13 <hr/>	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 <code>\tl_length:n</code> . This function requires three expansions, giving an $\langle integer denotation \rangle$.
<hr/> <code>\tl_reverse:n</code> ★	<code>\tl_reverse:n {\token list}</code>
<code>\tl_reverse:(V o)</code> ★	
<hr/> Updated: 2011-08-13 <hr/>	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 <code>\tl_reverse_items:n</code> . See also <code>\tl_reverse:N</code> .
<hr/> <code>\tl_reverse:N</code>	<code>\tl_reverse:N {\tl var}</code>
<code>\tl_reverse:c</code>	
<hr/> Updated: 2011-08-13 <hr/>	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. The reversal is local to the current TeX group. See also <code>\tl_reverse:n</code> .
<hr/> <code>\tl_reverse_items:n</code> ★	<code>\tl_reverse_items:n {\token list}</code>
<hr/> New: 2011-08-13 <hr/>	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 <code>\tl_reverse:n</code> or <code>\tl_reverse_tokens:n</code> .
<hr/> <code>\tl_trim_spaces:n</code> ★	<code>\tl_trim_spaces:n \langle token list \rangle</code>
<hr/> New: 2011-07-09 Updated: 2011-08-13 <hr/>	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.

TeXhackers note: The result is return 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_trim_spaces:N</code> <hr/>	<code>\tl_trim_spaces:N</code> $\langle tl\ var \rangle$
<code>\tl_trim_spaces:c</code> <hr/>	Removes any leading and trailing explicit space characters from the content of the $\langle tl\ var \rangle$ within the current \TeX group.
<hr/> <small>New: 2011-07-09</small> <hr/>	
<hr/> <code>\tl_gtrim_spaces:N</code> <hr/>	<code>\tl_gtrim_spaces:N</code> $\langle tl\ var \rangle$
<code>\tl_gtrim_spaces:c</code> <hr/>	Removes any leading and trailing explicit space characters from the content of the $\langle tl\ var \rangle$ globally.
<hr/> <small>New: 2011-07-09</small> <hr/>	

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.

<hr/> <code>\tl_head:n</code> ★	<code>\tl_head:n</code> $\{ \langle tokens \rangle \}$
<code>\tl_head:(V v f)</code> ★	Leaves in the input stream the first non-space token from the $\langle tokens \rangle$. Any leading space tokens will be discarded, and thus for example
<hr/> <small>Updated: 2011-08-09</small> <hr/>	

`\tl_head:n` { abc }

and

`\tl_head:n` { ~ abc }

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

<hr/> <code>\tl_head:w</code> ★	<code>\tl_head:w</code> $\langle tokens \rangle$ <code>\q_stop</code>
<hr/>	Leaves in the input stream the first non-space token from the $\langle tokens \rangle$. An empty list of $\langle tokens \rangle$ or one which consists only of space (category code 10) tokens will result in an error, and thus $\langle tokens \rangle$ must <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 <code>o</code> -type expansion. In general, <code>\tl_head:n</code> should be preferred if the number of expansions is not critical.

<code>\tl_tail:n</code>	★	<code>\tl_tail:n {⟨tokens⟩}</code>
-------------------------	---	------------------------------------

<code>\tl_tail:(V v f)</code>	★
-------------------------------	---

Updated: 2011-08-09

Discards the all leading space tokens and the first non-space token in the *⟨tokens⟩*, 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.

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

<code>\str_head:n</code>	★	<code>\str_head:n {⟨tokens⟩}</code>
--------------------------	---	-------------------------------------

<code>\str_tail:n</code>	★	<code>\str_tail:n {⟨tokens⟩}</code>
--------------------------	---	-------------------------------------

New: 2011-08-10

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

<code>\tl_if_head_eq_catcode_p:nN</code>	★	<code>\tl_if_head_eq_catcode_p:nN {⟨token list⟩} ⟨test token⟩</code>
<code>\tl_if_head_eq_catcode:nNTF</code>	★	<code>\tl_if_head_eq_catcode:nNTF {⟨token list⟩} ⟨test token⟩</code>
		<code>{⟨true code⟩} {⟨false code⟩}</code>

Updated: 2011-08-10

Tests if the first *⟨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:n</code>	★	<code>\tl_if_head_eq_meaning_p:nN</code>	{ <i>⟨token list⟩</i> }	<i>⟨test token⟩</i>
<code>\tl_if_head_eq_meaning:nTF</code>	★	<code>\tl_if_head_eq_meaning:nTF</code>	{ <i>⟨token list⟩</i> }	<i>⟨test token⟩</i>
			{ <i>⟨true code⟩</i> }	{ <i>⟨false code⟩</i> }

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</code>	{ <i>⟨token list⟩</i> }	
<code>\tl_if_head_group:nTF</code>	★	<code>\tl_if_head_group:nTF</code>	{ <i>⟨token list⟩</i> }	{ <i>⟨true code⟩</i> } { <i>⟨false code⟩</i> }

Updated: 2011-08-11

Tests if the first *⟨token⟩* in the *⟨token list⟩* is an explicit begin-group

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</code>	{ <i>⟨token list⟩</i> }		
<code>\tl_if_head_N_type:nTF</code>	★	<code>\tl_if_head_N_type:nTF</code>	{ <i>⟨token list⟩</i> }	{ <i>⟨true code⟩</i> }	{ <i>⟨false code⟩</i> }

New: 2011-08-11

Tests if the first *⟨token⟩* in the *⟨token list⟩* is a normal N-type argument

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</code>	{ <i>⟨token list⟩</i> }		
<code>\tl_if_head_space:nTF</code>	★	<code>\tl_if_head_space:nTF</code>	{ <i>⟨token list⟩</i> }	{ <i>⟨true code⟩</i> }	{ <i>⟨false code⟩</i> }

Updated: 2011-08-11

Tests if the first *⟨token⟩* in the *⟨token list⟩* is an explicit space charac

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</code>	<i>⟨tl var⟩</i>
<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 *<token list>* 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: 2011-08-11

This function, which works directly on T_EX tokens, reverses the order of the *<tokens>*: the first will be the last and the last will become first. Spaces are preserved. The reversal also operates within brace groups, but the braces themselves are not exchanged, as this would lead to an unbalanced token list. For instance, `\tl_reverse_tokens:n {a~{b()}}` leaves `{() (b)~a` in the input stream. This function requires two steps of expansion.

<hr/> <code>\tl_length_tokens:n</code> ★ <hr/>	<code>\tl_length_tokens:n {⟨tokens⟩}</code>
<hr/> New: 2011-08-11 <hr/>	Counts the number of T _E X tokens in the <i>⟨tokens⟩</i> 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 <code>a~{bc}</code> is 6. This function requires three expansions, giving an <i>⟨integer denotation⟩</i> .

<hr/> <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>
<hr/> New: 2011-08-13 <hr/>	

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.

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/>	

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_clear:N` *⟨sequence⟩*

Clears all items from the *⟨sequence⟩* within the scope of the current T_EX group.

`\seq_gclear:N`
`\seq_gclear:c`

`\seq_gclear:N` *⟨sequence⟩*

Clears all entries from the *⟨sequence⟩* globally.

`\seq_clear_new:N`
`\seq_clear_new:c`

`\seq_clear_new:N` *⟨sequence⟩*

If the *⟨sequence⟩* already exists, clears it within the scope of the current T_EX group. If the *⟨sequence⟩* is not defined, it will be created (using `\seq_new:N`). Thus the sequence is guaranteed to be available and clear within the current T_EX group. The *⟨sequence⟩* will exist globally, but the content outside of the current T_EX group is not specified.

`\seq_gclear_new:N`
`\seq_gclear_new:c`

`\seq_gclear_new:N` *⟨sequence⟩*

If the *⟨sequence⟩* already exists, clears it globally. If the *⟨sequence⟩* is not defined, it will be created (using `\seq_new:N`). Thus the sequence is guaranteed to be available and globally clear.

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

`\seq_set_eq:NN` *⟨sequence1⟩* *⟨sequence2⟩*

Sets the content of *⟨sequence1⟩* equal to that of *⟨sequence2⟩*. This assignment is restricted to the current T_EX group level.

<hr/>	
<code>\seq_gset_eq:Nn</code> <code>\seq_gset_eq:(cN Nc cc)</code>	<code>\seq_gset_eq:Nn <sequence1> <sequence2></code> Sets the content of $\langle sequence1 \rangle$ equal to that of $\langle sequence2 \rangle$. This assignment is global and so is not limited by the current T _E X group level.
<hr/>	
<code>\seq_concat:NNN</code> <code>\seq_concat:ccc</code>	<code>\seq_concat:NNN <sequence1> <sequence2> <sequence3></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. This operation is local to the current T _E X group and will remove any existing content in $\langle sequence1 \rangle$.
<hr/>	
<code>\seq_gconcat:NNN</code> <code>\seq_gconcat:ccc</code>	<code>\seq_gconcat:NNN <sequence1> <sequence2> <sequence3></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. This operation is global and will remove any existing content in $\langle sequence1 \rangle$.

95 Appending data to sequences

<hr/>	
<code>\seq_put_left:Nn</code> <code>\seq_put_left:(NV Nv No Nx cn cV cv co cx)</code>	<code>\seq_put_left:Nn <sequence> {\item}</code> Appends the $\langle item \rangle$ to the left of the $\langle sequence \rangle$. The assignment is restricted to the current T _E X group.
<hr/>	
<code>\seq_gput_left:Nn</code> <code>\seq_gput_left:(NV Nv No Nx cn cV cv co cx)</code>	<code>\seq_gput_left:Nn <sequence> {\item}</code> Appends the $\langle item \rangle$ to the left of the $\langle sequence \rangle$. The assignment is global.
<hr/>	
<code>\seq_put_right:Nn</code> <code>\seq_put_right:(NV Nv No Nx cn cV cv co cx)</code>	<code>\seq_put_right:Nn <sequence> {\item}</code> Appends the $\langle item \rangle$ to the right of the $\langle sequence \rangle$. The assignment is restricted to the current T _E X group.
<hr/>	
<code>\seq_gput_right:Nn</code> <code>\seq_gput_right:(NV Nv No Nx cn cV cv co cx)</code>	<code>\seq_gput_right:Nn <sequence> {\item}</code> Appends the $\langle item \rangle$ to the right of the $\langle sequence \rangle$. The assignment is global.

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> <hr/> <code>\seq_get_left:cN</code> <hr/>	<code>\seq_get_left:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ 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> <hr/> <code>\seq_get_right:cN</code> <hr/>	<code>\seq_get_right:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ 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> <hr/> <code>\seq_pop_left:cN</code> <hr/>	<code>\seq_pop_left:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ Pops the left-most item from a $\langle sequence \rangle$ into the $\langle token\ list\ variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token\ list\ variable \rangle$. Both of the variables are assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.
<hr/> <code>\seq_gpop_left:NN</code> <hr/> <code>\seq_gpop_left:cN</code> <hr/>	<code>\seq_gpop_left:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ Pops the left-most item from a $\langle sequence \rangle$ into the $\langle token\ list\ variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token\ list\ variable \rangle$. The $\langle sequence \rangle$ is modified globally, while the assignment of the $\langle token\ list\ variable \rangle$ is local. If $\langle sequence \rangle$ is empty an error will be raised.
<hr/> <code>\seq_pop_right:NN</code> <hr/> <code>\seq_pop_right:cN</code> <hr/>	<code>\seq_pop_right:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ Pops the right-most item from a $\langle sequence \rangle$ into the $\langle token\ list\ variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in in the $\langle token\ list\ variable \rangle$. Both of the variables are assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.
<hr/> <code>\seq_gpop_right:NN</code> <hr/> <code>\seq_gpop_right:cN</code> <hr/>	<code>\seq_gpop_right:NN</code> $\langle sequence \rangle$ $\langle token\ list\ variable \rangle$ Pops the right-most item from a $\langle sequence \rangle$ into the $\langle token\ list\ variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token\ list\ variable \rangle$. The $\langle sequence \rangle$ is modified globally, while the assignment of the $\langle token\ list\ variable \rangle$ is local. If $\langle sequence \rangle$ is empty an error will be raised.

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_remove_duplicates:N` $\langle sequence \rangle$

Removes duplicate items from the $\langle sequence \rangle$, leaving the left most copy of each item in the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is local to the current \TeX group.

\TeX hackers note: This function iterates through every item in the $\langle sequence \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large sequences.

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

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

Removes duplicate items from the $\langle sequence \rangle$, leaving the left most copy of each item in the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is applied globally.

\TeX hackers note: This function iterates through every item in the $\langle sequence \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large sequences.

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

`\seq_remove_all:Nn` $\langle sequence \rangle$ $\{\langle item \rangle\}$

Removes every occurrence of $\langle item \rangle$ from the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is local to the current \TeX group.

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

`\seq_gremove_all:Nn` $\langle sequence \rangle$ $\{\langle item \rangle\}$

Removes each occurrence of $\langle item \rangle$ from the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is applied globally.

98 Sequence conditionals

`\seq_if_empty_p:N` ★
`\seq_if_empty_p:c` ★
`\seq_if_empty:N \underline{TF}` ★
`\seq_if_empty:c \underline{TF}` ★

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

`\seq_if_empty:N \underline{TF}` $\langle sequence \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle sequence \rangle$ is empty (containing no items).

`\seq_if_in:Nn \underline{TF}`
`\seq_if_in:(\underline{NV} | \underline{Nv} | \underline{No} | \underline{Nx} | \underline{cn} | \underline{cV} | \underline{cv} | \underline{co} | \underline{cx}) \underline{TF}`

`\seq_if_in:Nn \underline{TF}` $\langle sequence \rangle$ $\{\langle item \rangle\}$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle item \rangle$ is present in the $\langle sequence \rangle$.

99 Mapping to sequences

<hr/> <code>\seq_map_function:NN</code> ☆ <code>\seq_map_function:cN</code> ☆ <hr/>	<code>\seq_map_function:NN</code> $\langle sequence \rangle$ $\langle function \rangle$ <p>Applies $\langle function \rangle$ to every $\langle item \rangle$ stored in the $\langle sequence \rangle$. The $\langle function \rangle$ will receive one argument for each iteration. The $\langle items \rangle$ are returned from left to right. The function <code>\seq_map_inline:Nn</code> is in general more efficient than <code>\seq_map_function:NN</code>. One mapping may be nested inside another.</p>
<hr/> <code>\seq_map_inline:Nn</code> <code>\seq_map_inline:cn</code> <hr/>	<code>\seq_map_inline:Nn</code> $\langle sequence \rangle$ $\{ \langle inline function \rangle \}$ <p>Applies $\langle inline function \rangle$ to every $\langle item \rangle$ stored within the $\langle sequence \rangle$. 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.</p>
<hr/> <code>\seq_map_variable:NNn</code> <code>\seq_map_variable:(Ncn cNn ccn)</code> <hr/>	<code>\seq_map_variable:NNn</code> $\langle sequence \rangle$ $\langle tl var. \rangle$ $\{ \langle function using tl var. \rangle \}$ <p>Stores each entry in the $\langle sequence \rangle$ in turn in the $\langle tl var. \rangle$ and applies the $\langle function using tl var. \rangle$. The $\langle function \rangle$ will usually consist of code making use of the $\langle tl var. \rangle$, but this is not enforced. One variable mapping can be nested inside another. The $\langle items \rangle$ are returned from left to right.</p>
<hr/> <code>\seq_map_break</code> ☆ <hr/>	<code>\seq_map_break:</code> <p>Used to terminate a <code>\seq_map...</code> function before all entries in the $\langle sequence \rangle$ have been processed. This will normally take place within a conditional statement, for example</p> <pre> \seq_map_inline:Nn \l_my_seq { \str_if_eq:nnTF { #1 } { bingo } { \seq_map_break: } { % Do something useful } } </pre> <p>Use outside of a <code>\seq_map...</code> scenario will lead to low level TeX errors.</p> <p>TeXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro <code>\seq_break_point:n</code> before further items are taken from the input stream. This will depend on the design of the mapping function.</p>

`\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 `\seq_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:NN ⟨sequence⟩ ⟨token list variable⟩`

`\seq_get:cN`

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:NN ⟨sequence⟩ ⟨token list variable⟩`

`\seq_pop:cN`

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:NN ⟨sequence⟩ ⟨token list variable⟩`

`\seq_gpop:cN`

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.

```
\seq_push:Nn
\seq_push:(NV|Nv|No|Nx|cn|cV|cv|co|cx)
```

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

Adds the $\{\langle item \rangle\}$ to the top of the $\langle sequence \rangle$. The assignment is restricted to the current $\text{T}_{\text{E}}\text{X}$ group.

```
\seq_gpush:Nn
\seq_gpush:(NV|Nv|No|Nx|cn|cV|cv|co|cx)
```

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

Pushes the $\langle item \rangle$ onto the end of the top of the $\langle sequence \rangle$. The assignment is global.

101 Viewing sequences

```
\seq_show:N
\seq_show:c
```

```
\seq_show:N <sequence>
```

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.

```
\seq_get_left:NNTF
\seq_get_left:cNTF
```

```
\seq_get_left:NNTF <sequence> <token list variable> {\true code} {\false 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.

```
\seq_get_right:NNTF
\seq_get_right:cNTF
```

```
\seq_get_right:NNTF <sequence> <token list variable> {\true code} {\false 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.

```
\seq_pop_left:NNTF
\seq_pop_left:cNTF
```

```
\seq_pop_left:NNTF <sequence> <token list variable> {\true code} {\false 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.e.* removes the item from a $\langle sequence \rangle$. Both the $\langle sequence \rangle$ and the $\langle token list variable \rangle$ are assigned locally.

```
\seq_gpop_left:NNTF
\seq_gpop_left:cNTF
```

```
\seq_gpop_left:NNTF <sequence> <token list variable> {\true code} {\false 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.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.

<hr/> <u><code>\seq_pop_right:NNTF</code></u> <u><code>\seq_pop_right:cNTF</code></u> <hr/>	<code>\seq_pop_right:NNTF</code> $\langle sequence \rangle$ $\langle token list variable \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$ If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. Both the $\langle sequence \rangle$ and the $\langle token list variable \rangle$ are assigned locally.
<hr/> <u><code>\seq_gpop_right:NNTF</code></u> <u><code>\seq_gpop_right:cNTF</code></u> <hr/>	<code>\seq_gpop_right:NNTF</code> $\langle sequence \rangle$ $\langle token list variable \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$ If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. The $\langle sequence \rangle$ is modified globally, while the $\langle token list variable \rangle$ is assigned locally.
<hr/> <u><code>\seq_length:N</code> ☆</u> <u><code>\seq_length:c</code> ☆</u> <hr/>	<code>\seq_length:N</code> $\langle sequence \rangle$ Leaves the number of items in the $\langle sequence \rangle$ in the input stream as an $\langle integer denotation \rangle$. The total number of items in a $\langle sequence \rangle$ will include those which are empty and duplicates, <i>i.e.</i> every item in a $\langle sequence \rangle$ is unique.
<hr/> <u><code>\seq_item:Nn</code> ☆</u> <u><code>\seq_item:cn</code> ☆</u> <hr/>	<code>\seq_item:Nn</code> $\langle sequence \rangle$ $\{\langle integer expression \rangle\}$ Indexing items in the $\langle sequence \rangle$ from 0 at the top (left), this function will evaluate the $\langle integer expression \rangle$ and leave the appropriate item from the sequence in the input stream. If the $\langle integer expression \rangle$ is negative, indexing occurs from the bottom (right) of the sequence. When the $\langle integer expression \rangle$ is larger than the number of items in the $\langle sequence \rangle$ (as calculated by <code>\seq_length:N</code>) then the function will expand to nothing.
<hr/> <u><code>\seq_use:N</code> ☆</u> <u><code>\seq_use:c</code> ☆</u> <hr/>	<code>\seq_use:N</code> $\langle sequence \rangle$ Places each $\langle item \rangle$ in the $\langle sequence \rangle$ in turn in the input stream. This occurs in an expandable fashion, and is implemented as a mapping. This means that the process may be prematurely terminated using <code>\seq_map_break:</code> or <code>\seq_map_break:n</code> . The $\langle items \rangle$ in the $\langle sequence \rangle$ will be used from left (top) to right (bottom).
<hr/> <u><code>\seq_mapthread_function:NNN</code> ☆</u> <u><code>\seq_mapthread_function:(NcN cNN ccN)</code> ☆</u> <hr/>	<code>\seq_mapthread_function:NNN</code> $\langle seq1 \rangle$ $\langle seq2 \rangle$ $\langle function \rangle$ Applies $\langle function \rangle$ to every pair of items $\langle seq1-item \rangle$ – $\langle seq2-item \rangle$ from the two sequences, returning items from both sequences from left to right. The $\langle function \rangle$ will receive two n -type arguments for each iteration. The mapping will terminate when the end of either sequence is reached (<i>i.e.</i> whichever sequence has fewer items determines how many iterations occur).
<hr/> <u><code>\seq_set_from_clist:NN</code></u> <u><code>\seq_set_from_clist:(cN Nc cc Nn cn)</code></u> <hr/>	<code>\seq_set_from_clist:NN</code> $\langle sequence \rangle$ $\langle comma-list \rangle$ Sets the $\langle sequence \rangle$ within the current \TeX group to be equal to the content of the $\langle comma-list \rangle$.

<code>\seq_gset_from_clist:NN</code>	<code>\seq_gset_from_clist:NN</code> $\langle sequence \rangle$ $\langle comma-list \rangle$
<code>\seq_gset_from_clist:(cN Nc cc Nn cn)</code>	

Sets the $\langle sequence \rangle$ globally to equal to the content of the $\langle comma-list \rangle$.

<code>\seq_set_reverse:N</code>	<code>\seq_set_reverse:N</code> $\langle sequence \rangle$
<code>\seq_gset_reverse:N</code>	

New: 2011-08-12

Reverses the order of items in the $\langle sequence \rangle$, and assigns the result to $\langle sequence \rangle$, locally or globally according to the variant chosen.

<code>\seq_set_split:Nnn</code>	<code>\seq_set_split:Nnn</code> $\langle sequence \rangle$ $\{\langle delimiter \rangle\}$ $\{\langle token list \rangle\}$
<code>\seq_gset_split:Nnn</code>	

New: 2011-08-15

This function splits the $\langle token list \rangle$ into $\langle items \rangle$ separated by $\langle delimiter \rangle$, ignoring all explicit space characters from both sides of each $\langle item \rangle$, then removing one set of outer braces if any. The result is assigned to $\langle sequence \rangle$, locally or globally according to the function chosen. The $\langle delimiter \rangle$ may not contain $\{$, $\}$ or $\#$ (assuming TeX's normal category code régime).

103 Internal sequence functions

<code>\seq_if_empty_err_break:N</code>	<code>\seq_if_empty_err_break:N</code> $\langle sequence \rangle$
--	---

Tests if the $\langle sequence \rangle$ is empty, and if so issues an error message before skipping over any tokens up to `\seq_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 $\langle sequence \rangle$.

<code>\seq_item:n</code> ★	<code>\seq_item:n</code> $\langle item \rangle$
----------------------------	---

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.

<code>\seq_push_item_def:n</code>	<code>\seq_push_item_def:n</code> $\{\langle code \rangle\}$
<code>\seq_push_item_def:x</code>	

Saves the definition of `\seq_item:n` and redefines it to accept one parameter and expand to $\langle code \rangle$. This function should always be balanced by use of `\seq_pop_item_def:.`

<code>\seq_pop_item_def:</code>	<code>\seq_pop_item_def:</code>
---------------------------------	---------------------------------

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`.

<code>\seq_break</code> ★	<code>\seq_break:</code>
---------------------------	--------------------------

Used to terminate sequence functions by gobbling all tokens up to `\seq_break_point:n`. This function is a copy of `\seq_map_break:`, but is used in situations which are not mappings.

`\seq_break:n` ★ `\seq_break:n {(tokens)}`

Used to terminate sequence functions by gobbling all tokens up to `\seq_break_point:n`, then inserting the $\langle tokens \rangle$ 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.

`\seq_break_point:n` ★ `\seq_break_point:n \langle tokens \rangle`

Used to mark the end of a recursion or mapping: the functions `\seq_map_break:` and `\seq_map_break:n` 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 *not* applied: `\seq_break_point:n` is functionally-equivalent in these cases to `\use:n`.

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 *<comma list>* or raises an error if the name is already taken. The declaration is global. The *<comma list>* 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 *<comma list>* within the scope of the current TeX group.

<code>\clist_gclear:N</code>	<code>\clist_gclear:N <comma list></code>
------------------------------	---

<code>\clist_gclear:c</code>	
------------------------------	--

Clears all entries from the *<comma list>* globally.

<code>\clist_clear_new:N</code>	<code>\clist_clear_new:N <comma list></code>
---------------------------------	--

<code>\clist_clear_new:c</code>	
---------------------------------	--

If the *<comma list>* already exists, clears it within the scope of the current TeX group. If the *<comma list>* is not defined, it will be created (using `\clist_new:N`). Thus the comma list is guaranteed to be available and clear within the current TeX group. The *<comma list>* will exist globally, but the content outside of the current TeX group is not specified.

<hr/> <code>\clist_gclear_new:N</code> <code>\clist_gclear_new:c</code> <hr/>	<code>\clist_gclear_new:N <comma list></code> If the $\langle comma list \rangle$ already exists, clears it globally. If the $\langle comma list \rangle$ is not defined, it will be created (using <code>\clist_new:N</code>). Thus the comma list is guaranteed to be available and globally clear.
<hr/> <code>\clist_set_eq:NN</code> <code>\clist_set_eq:(cN Nc cc)</code> <hr/>	<code>\clist_set_eq:NN <comma list1> <comma list2></code> Sets the content of $\langle comma list1 \rangle$ equal to that of $\langle comma list2 \rangle$. This assignment is restricted to the current T _E X group level.
<hr/> <code>\clist_gset_eq:NN</code> <code>\clist_gset_eq:(cN Nc cc)</code> <hr/>	<code>\clist_gset_eq:NN <comma list1> <comma list2></code> Sets the content of $\langle comma list1 \rangle$ equal to that of $\langle comma list2 \rangle$. This assignment is global and so is not limited by the current T _E X group level.
<hr/> <code>\clist_concat:NNN</code> <code>\clist_concat:ccc</code> <hr/>	<code>\clist_concat:NNN <comma list1> <comma list2> <comma list3></code> Concatenates the content of $\langle comma list2 \rangle$ and $\langle comma list3 \rangle$ together and saves the result in $\langle comma list1 \rangle$. The items in $\langle comma list2 \rangle$ will be placed at the left side of the new comma list. This operation is local to the current T _E X group and will remove any existing content in $\langle comma list1 \rangle$.
<hr/> <code>\clist_gconcat:NNN</code> <code>\clist_gconcat:ccc</code> <hr/>	<code>\clist_gconcat:NNN <comma list1> <comma list2> <comma list3></code> Concatenates the content of $\langle comma list2 \rangle$ and $\langle comma list3 \rangle$ together and saves the result in $\langle comma list1 \rangle$. The items in $\langle comma list2 \rangle$ will be placed at the left side of the new comma list. This operation is global and will remove any existing content in $\langle comma list1 \rangle$.

105 Adding data to comma lists

<hr/> <code>\clist_set:Nn</code> <code>\clist_set:(NV No Nx cn cV co cx)</code> <hr/>	<code>\clist_set:Nn <comma list> {\langle item1 \rangle, \dots, \langle item_n \rangle}</code> New: 2011-09-06
	Sets $\langle comma list \rangle$ to contain the $\langle items \rangle$, removing any previous content from the variable. Spaces are removed from both sides of each item. The assignment is restricted to the current T _E X group.
<hr/> <code>\clist_gset:Nn</code> <code>\clist_gset:(NV No Nx cn cV co cx)</code> <hr/>	<code>\clist_gset:Nn <comma list> {\langle item1 \rangle, \dots, \langle item_n \rangle}</code> New: 2011-09-06
	Sets $\langle comma list \rangle$ to contain the $\langle items \rangle$, removing any previous content from the variable. Spaces are removed from both sides of each item. The assignment is global.

```
\clist_put_left:Nn
\clist_put_left:(NV|No|Nx|cn|cV|co|cx)
```

Updated: 2011-09-05

Appends the $\langle items \rangle$ to the left of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is restricted to the current $\text{T}_{\text{E}}\text{X}$ group.

```
\clist_gput_left:Nn
\clist_gput_left:(NV|No|Nx|cn|cV|co|cx)
```

Updated: 2011-09-05

Appends the $\langle items \rangle$ to the left of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is global.

```
\clist_put_right:Nn
\clist_put_right:(NV|No|Nx|cn|cV|co|cx)
```

Updated: 2011-09-05

Appends the $\langle items \rangle$ to the right of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is restricted to the current $\text{T}_{\text{E}}\text{X}$ group.

```
\clist_gput_right:Nn
\clist_gput_right:(NV|No|Nx|cn|cV|co|cx)
```

Updated: 2011-09-05

Appends the $\langle item \rangle$ to the right of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is global.

106 Using comma lists

```
\clist_use:N ★ \clist_use:N \langle comma list \rangle
```

```
\clist_use:c ★
```

Places the $\langle comma list \rangle$ directly into the input stream, thus treating it as a $\langle token list \rangle$.

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.

```
\clist_remove_duplicates:N
\clist_remove_duplicates:c
```

```
\clist_remove_duplicates:N <comma list>
```

Removes duplicate items from the $\langle comma list \rangle$, leaving the left most copy of each item in the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is local to the current \TeX group.

\TeX hackers note: This function iterates through every item in the $\langle comma list \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large comma lists. Furthermore, it will not work if any of the items in the $\langle comma list \rangle$ contains `{`, `}`, or `#` (assuming the usual \TeX category codes apply).

```
\clist_gremove_duplicates:N \clist_gremove_duplicates:N <comma list>
\clist_gremove_duplicates:c
```

Removes duplicate items from the $\langle comma list \rangle$, leaving the left most copy of each item in the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is applied globally.

\TeX hackers note: This function iterates through every item in the $\langle comma list \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large comma lists. Furthermore, it will not work if any of the items in the $\langle comma list \rangle$ contains `{`, `}`, or `#` (assuming the usual \TeX category codes apply).

```
\clist_remove_all:Nn
\clist_remove_all:cn
```

Updated: 2011-09-06

```
\clist_remove_all:Nn <comma list> {\item}
```

Removes every occurrence of $\langle item \rangle$ from the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is local to the current \TeX group.

\TeX hackers note: The $\langle item \rangle$ may not contain `{`, `}`, or `#` (assuming the usual \TeX category codes apply).

```
\clist_gremove_all:Nn
\clist_gremove_all:cn
```

Updated: 2011-09-06

```
\clist_gremove_all:Nn <comma list> {\item}
```

Removes each occurrence of $\langle item \rangle$ from the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`. The removal is applied globally.

\TeX hackers note: The $\langle item \rangle$ may not contain `{`, `}`, or `#` (assuming the usual \TeX category codes apply).

108 Comma list conditionals

```
\clist_if_empty_p:N *
\clist_if_empty_p:c *
\clist_if_empty:NTF *
\clist_if_empty:cTF *
```

```
\clist_if_empty_p:N <comma list>
```

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

Tests if the $\langle comma list \rangle$ is empty (containing no items).

<code>\clist_if_eq_p:NN</code>	★	<code>\clist_if_eq_p:NN {⟨clist₁⟩} {⟨clist₂⟩}</code>
<code>\clist_if_eq_p:(Nc cN cc)</code>	★	<code>\clist_if_eq:NNTF {⟨clist₁⟩} {⟨clist₂⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\clist_if_eq:NNTF</code>	★	
<code>\clist_if_eq:(Nc cN cc)TF</code>	★	Compares the content of two <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:NnTF ⟨comma list⟩ {⟨item⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\clist_if_in:(NV No cn cV co nn nV no)TF</code>	

Updated: 2011-09-06

Tests if the *⟨item⟩* is present in the *⟨comma list⟩*. In the case of an **n**-type *⟨comma list⟩*, 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 *⟨item⟩* 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:NN ⟨comma list⟩ ⟨function⟩</code>
<code>\clist_map_function:(cN nN)</code>	☆	

Applies *⟨function⟩* to every *⟨item⟩* stored in the *⟨comma list⟩*. The *⟨function⟩* will receive one argument for each iteration. The *⟨items⟩* 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:Nn ⟨comma list⟩ {⟨inline function⟩}</code>
<code>\clist_map_inline:(cn nn)</code>	

Applies *⟨inline function⟩* to every *⟨item⟩* stored within the *⟨comma list⟩*. 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>\clist_map_variable:Nn</code>	<code>\clist_map_variable:Nn <comma list> <tl var.> {<function using tl var.>}</code>
<code>\clist_map_variable:(cNn nNn)</code>	

Stores each entry in the *<comma list>* in turn in the *<tl var.>* and applies the *<function using tl var.>* The *<function>* will usually consist of code making use of the *<tl var.>*, but this is not enforced. One variable mapping can be nested inside another. The *<items>* are returned from left to right.

<code>\clist_map_break</code> ☆	<code>\clist_map_break:</code>
---------------------------------	--------------------------------

Used to terminate a `\clist_map...` function before all entries in the *<comma list>* have been processed. This will normally take place within a conditional statement, for example

```
\clist_map_inline:Nn \l_my_clist
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \clist_map_break: }
  {
    % Do something useful
  }
}
```

Use outside of a `\clist_map...` scenario will lead to low level TeX errors.

TeXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro `\clist_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 `\clist_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

`\clist_gpop:Nn`
`\clist_gpop:cN`

`\clist_gpop:Nn` $\langle comma list \rangle$ $\langle token list variable \rangle$

Pops the left-most item from a $\langle comma list \rangle$ into the $\langle token list variable \rangle$, *i.e.* 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.

`\clist_push:Nn`
`\clist_push:(NV|No|Nx|cn|cV|co|cx)`

`\clist_push:Nn` $\langle comma list \rangle$ $\{ \langle items \rangle \}$

Adds the $\{ \langle items \rangle \}$ to the top of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is restricted to the current \TeX group.

`\clist_gpush:Nn`
`\clist_gpush:(NV|No|Nx|cn|cV|co|cx)`

`\clist_gpush:Nn` $\langle comma list \rangle$ $\{ \langle items \rangle \}$

Pushes the $\langle items \rangle$ onto the end of the top of the $\langle comma list \rangle$. Spaces are removed from both sides of each item. The assignment is global.

111 Viewing comma lists

`\clist_show:N`
`\clist_show:c`

`\clist_show:N` $\langle comma list \rangle$

Displays the entries in the $\langle comma list \rangle$ in the terminal.

112 Scratch comma lists

`\l_tmpa_clist`
`\l_tmpb_clist`
 New: 2011-09-06

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

`\g_tmpa_clist`
`\g_tmpb_clist`
 New: 2011-09-06

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

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.

`\clist_length:N` ★
`\clist_length:(c|n)` ★

`\clist_length:N` $\langle comma list \rangle$

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 empty and duplicates, *i.e.* every item in a $\langle comma list \rangle$ is unique.

New: 2011-06-25
 Updated: 2011-09-06

<hr/>	
<code>\clist_item:Nn</code> ★	<code>\clist_item:Nn <comma list> {<integer expression>}</code>
<code>\clist_item:(cn nn)</code> ★	
Updated: 2011-09-06	
	Indexing items in the <i><comma list></i> from 0 at the top (left), this function will evaluate the <i><integer expression></i> and leave the appropriate item from the comma list in the input stream. If the <i><integer expression></i> is negative, indexing occurs from the bottom (right) of the comma list. When the <i><integer expression></i> is larger than the number of items in the <i><comma list></i> (as calculated by <code>\clist_length:N</code>) then the function will expand to nothing.

<hr/>	
<code>\clist_set_from_seq:NN</code>	<code>\clist_set_from_seq:NN <comma list> <sequence></code>
<code>\clist_set_from_seq:(cN Nc cc)</code>	
Updated: 2011-08-31	
	Sets the <i><comma list></i> within the current \TeX group to be equal to the content of the <i><sequence></i> . Items which contain either spaces or commas are surrounded by braces.

<hr/>	
<code>\clist_gset_from_seq:NN</code>	<code>\clist_gset_from_seq:NN <comma list> <sequence></code>
<code>\clist_gset_from_seq:(cN Nc cc)</code>	
Updated: 2011-08-31	
	Sets the <i><comma list></i> globally to equal to the content of the <i><sequence></i> . Items which contain either spaces or commas are surrounded by braces.

114 Internal comma-list functions

<hr/>	
<code>\clist_trim_spaces:n</code> ☆	<code>\clist_trim_spaces:n {<comma list>}</code>
New: 2011-07-09	
	Removes leading and trailing spaces from each <i><item></i> in the <i><comma list></i> , leaving the resulting modified list in the input stream. This is used by the functions which add data into a comma list.

Part XIV

The l3prop package

Property lists

L^AT_EX3 implements a “property list” data type, which contain an unordered list of entries each of which consists of a $\langle key \rangle$ and an associated $\langle value \rangle$. The $\langle key \rangle$ and $\langle value \rangle$ may both be any $\langle balanced\ text \rangle$. It is possible to map functions to property lists such that the function is applied to every key–value pair within the list.

Each entry in a property list must have a unique $\langle key \rangle$: if an entry is added to a property list which already contains the $\langle key \rangle$ then the new entry will overwrite the existing one. The $\langle keys \rangle$ are compared on a string basis, using the same method as `\str_if_eq:nn`.

Property lists are intended for storing key-based information for use within code. This is in contrast to key–value lists, which are a form of *input* parsed by the `keys` module.

115 Creating and initialising property lists

 $\backslash\text{prop_new:N}$
 $\backslash\text{prop_new:c}$

 $\backslash\text{prop_new:N}$ $\langle property\ list \rangle$

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.

 $\backslash\text{prop_clear:N}$
 $\backslash\text{prop_clear:c}$

 $\backslash\text{prop_clear:N}$ $\langle property\ list \rangle$

Clears all entries from the $\langle property\ list \rangle$ within the scope of the current T_EX group.

 $\backslash\text{prop_gclear:N}$
 $\backslash\text{prop_gclear:c}$

 $\backslash\text{prop_gclear:N}$ $\langle property\ list \rangle$

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

 $\backslash\text{prop_clear_new:N}$
 $\backslash\text{prop_clear_new:c}$

 $\backslash\text{prop_clear_new:N}$ $\langle property\ list \rangle$

If the $\langle property\ list \rangle$ already exists, clears it within the scope of the current T_EX group. If the $\langle property\ list \rangle$ is not defined, it will be created (using `\prop_new:N`). Thus the property list is guaranteed to be available and clear within the current T_EX group. The $\langle property\ list \rangle$ will exist globally, but the content outside of the current T_EX group is not specified.

 $\backslash\text{prop_gclear_new:N}$
 $\backslash\text{prop_gclear_new:c}$

 $\backslash\text{prop_gclear_new:N}$ $\langle property\ list \rangle$

If the $\langle property\ list \rangle$ already exists, clears it globally. If the $\langle property\ list \rangle$ is not defined, it will be created (using `\prop_new:N`). Thus the property list is guaranteed to be available and globally clear.

<code>\prop_set_eq:NN</code>	<code>\prop_set_eq:NN <property list1> <property list2></code>
<code>\prop_set_eq:(cN Nc cc)</code>	Sets the content of <i><property list1></i> equal to that of <i><property list2></i> . This assignment is restricted to the current T _E X group level.

<code>\prop_gset_eq:NN</code>	<code>\prop_gset_eq:NN <property list1> <property list2></code>
<code>\prop_gset_eq:(cN Nc cc)</code>	Sets the content of <i><property list1></i> equal to that of <i><property list2></i> . This assignment is global and so is not limited by the current T _E X group level.

116 Adding entries to property lists

<code>\prop_put:Nnn</code>	<code>\prop_put:Nnn <property list> {<key>}</code>
<code>\prop_put:(NnV Nno Nnx NVn NVV Non Noo cnn cnV cno cnx cVn cVV con coo)</code>	<code>{<value>}</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>*. The assignment is restricted to the current T_EX group.

<code>\prop_gput:Nnn</code>	<code>\prop_gput:Nnn <property list></code>
<code>\prop_gput:(NnV Nno Nnx NVn NVV Non Noo cnn cnV cno cnx cVn cVV con coo)</code>	<code>{<key>} {<value>}</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>*. The assignment is global.

<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>	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. The assignment is restricted to the current T _E X group.

<code>\prop_gput_if_new:Nnn</code>	<code>\prop_gput_if_new:Nnn <property list> {<key>} {<value>}</code>
<code>\prop_gput_if_new:cnn</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. The assignment is global.

117 Recovering values from property lists

`\prop_get:NnN`
`\prop_get:(NVN|NoN|cnN|cVN|coN)`

Updated: 2011-08-28

`\prop_get:NnN <property list> {<key>} <tl var>`

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`.

`\prop_pop:NnN`
`\prop_pop:(NoN|cnN|coN)`

Updated: 2011-08-18

`\prop_pop:NnN <property list> {<key>} <tl var>`

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 *<key>* and *<value>* are then deleted from the property list. Both assignments are local.

`\prop_gpop:NnN`
`\prop_gpop:(NoN|cnN|coN)`

Updated: 2011-08-18

`\prop_gpop:NnN <property list> {<key>} <tl var>`

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 *<key>* and *<value>* are then deleted from the property list. The *<property list>* is modified globally, while the assignment of the *<token list variable>* is local.

118 Modifying property lists

`\prop_del:Nn`
`\prop_del:(NV|cn|cV)`

`\prop_del:Nn <property list> {<key>}`

Deletes the entry listed under *<key>* from the *<property list>* which may be accessed. If the *<key>* is not found in the *<property list>* no change occurs, *i.e* there is no need to test for the existence of a key before deleting it. The deletion is restricted to the current \TeX group.

`\prop_gdel:Nn`
`\prop_gdel:(NV|cn|cV)`

`\prop_gdel:Nn <property list> {<key>}`

Deletes the entry listed under *<key>* from the *<property list>* which may be accessed. If the *<key>* is not found in the *<property list>* no change occurs, *i.e* there is no need to test for the existence of a key before deleting it. The deletion is not restricted to the current \TeX group: it is global.

119 Property list conditionals

<code>\prop_if_empty_p:N</code>	★	<code>\prop_if_empty_p:N</code>	$\langle \textit{property list} \rangle$
<code>\prop_if_empty_p:c</code>	★	<code>\prop_if_empty:N</code>	$\langle \textit{property list} \rangle$ $\{\langle \textit{true code} \rangle\}$ $\{\langle \textit{false code} \rangle\}$
<code>\prop_if_empty:N</code>	★	Tests if the $\langle \textit{property list} \rangle$ is empty (containing no entries).	
<code>\prop_if_empty:c</code>	★	<code><u>TF</u></code>	

<code>\prop_if_in_p:Nn</code>	★	<code>\prop_if_in:Nn</code>	$\langle \textit{property list} \rangle$ $\{\langle \textit{key} \rangle\}$ $\{\langle \textit{true code} \rangle\}$ $\{\langle \textit{false code} \rangle\}$
<code>\prop_if_in_p:(NV No cn cV co)</code>	★		
<code>\prop_if_in:Nn</code>	★	<code><u>TF</u></code>	
<code>\prop_if_in:(NV No cn cV co)</code>	★	<code><u>TF</u></code>	

Updated: 2011-09-15

Tests if the $\langle \textit{key} \rangle$ is present in the $\langle \textit{property list} \rangle$, making the comparison using the method described by `\str_if_eq:n`.

T_EXhackers note: This function iterates through every key–value pair in the $\langle \textit{property list} \rangle$ and is therefore slower than using the non-expandable `\prop_get:Nn`.

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:Nn</code>	<code><u>TF</u></code>	<code>\prop_get:Nn</code>	$\langle \textit{property list} \rangle$ $\{\langle \textit{key} \rangle\}$ $\langle \textit{token list variable} \rangle$
<code>\prop_get:(NVN NoN cnN cVN coN)</code>	<code><u>TF</u></code>		$\{\langle \textit{true code} \rangle\}$ $\{\langle \textit{false code} \rangle\}$

Updated: 2011-08-28

If the $\langle \textit{key} \rangle$ is not present in the $\langle \textit{property list} \rangle$, leaves the $\langle \textit{false code} \rangle$ in the input stream and leaves the $\langle \textit{token list variable} \rangle$ unchanged. If the $\langle \textit{key} \rangle$ is present in the $\langle \textit{property list} \rangle$, stores the corresponding $\langle \textit{value} \rangle$ in the $\langle \textit{token list variable} \rangle$ without removing it from the $\langle \textit{property list} \rangle$. The $\langle \textit{token list variable} \rangle$ is assigned locally.

<code>\prop_pop:Nn</code>	<code><u>TF</u></code>	<code>\prop_pop:Nn</code>	$\langle \textit{property list} \rangle$ $\{\langle \textit{key} \rangle\}$ $\langle \textit{token list variable} \rangle$
<code>\prop_pop:cn</code>	<code><u>TF</u></code>		$\{\langle \textit{true code} \rangle\}$ $\{\langle \textit{false code} \rangle\}$

New: 2011-08-18

If the $\langle \textit{key} \rangle$ is not present in the $\langle \textit{property list} \rangle$, leaves the $\langle \textit{false code} \rangle$ in the input stream and leaves the $\langle \textit{token list variable} \rangle$ unchanged. If the $\langle \textit{key} \rangle$ is present in the $\langle \textit{property list} \rangle$, pops the corresponding $\langle \textit{value} \rangle$ in the $\langle \textit{token list variable} \rangle$, *i.e.* removes the item from the $\langle \textit{property list} \rangle$. Both the $\langle \textit{property list} \rangle$ and the $\langle \textit{token list variable} \rangle$ 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 T_EX errors.

`\prop_map_break:n` ☆

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

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

<code>\prop_show:N</code>	<code>\prop_show:N</code> \langle <i>property list</i> \rangle
<code>\prop_show:c</code>	Displays the entries in the \langle <i>property list</i> \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.

<code>\prop_gpop:NnNTF</code>	<code>\prop_gpop:NnNTF</code> \langle <i>property list</i> \rangle $\{ \langle$ <i>key</i> $\rangle \}$ \langle <i>token list variable</i> \rangle
<code>\prop_gpop:cnNTF</code>	$\{ \langle$ <i>true code</i> $\rangle \}$ $\{ \langle$ <i>false code</i> $\rangle \}$
New: 2011-08-18	If the \langle <i>key</i> \rangle is not present in the \langle <i>property list</i> \rangle , leaves the \langle <i>false code</i> \rangle in the input stream and leaves the \langle <i>token list variable</i> \rangle unchanged. If the \langle <i>key</i> \rangle is present in the \langle <i>property list</i> \rangle , pops the corresponding \langle <i>value</i> \rangle in the \langle <i>token list variable</i> \rangle , <i>i.e.</i> removes the item from the \langle <i>property list</i> \rangle . The \langle <i>property list</i> \rangle is modified globally, while the \langle <i>token list variable</i> \rangle is assigned locally.

<code>\prop_map_tokens:Nn</code> ☆	<code>\prop_map_tokens:Nn</code> \langle <i>property list</i> \rangle $\{ \langle$ <i>code</i> $\rangle \}$
<code>\prop_map_tokens:cn</code> ☆	Analogue of <code>\prop_map_function:NN</code> 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.
New: 2011-08-18	

<code>\prop_get:Nn</code> ★	<code>\prop_get:Nn</code> \langle <i>property list</i> \rangle $\{ \langle$ <i>key</i> $\rangle \}$
<code>\prop_get:cn</code> ★	Expands to the \langle <i>value</i> \rangle corresponding to the \langle <i>key</i> \rangle in the \langle <i>property list</i> \rangle . If the \langle <i>key</i> \rangle is missing, this has an empty expansion.
Updated: 2011-09-15	

T_EXhackers note: This function is slower than the non-expandable analogue `\prop_get:NnN`.

124 Internal property list functions

<code>\q_prop</code>	The internal token used to separate out property list entries, separating both the \langle <i>key</i> \rangle from the \langle <i>value</i> \rangle and also one entry from another.
----------------------	--

<code>\c_empty_prop</code>	A permanently-empty property list used for internal comparisons.
----------------------------	--

<hr/> <hr/>	<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> .
<hr/> <hr/>	<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

<hr/> <code>\box_new:N</code> <code>\box_new:c</code> <hr/>	<code>\box_new:N</code> $\langle box \rangle$ Creates a new $\langle box \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle box \rangle$ will initially be void.
<hr/> <code>\box_clear:N</code> <code>\box_clear:c</code> <hr/>	<code>\box_clear:N</code> $\langle box \rangle$ Clears the content of the $\langle box \rangle$ by setting the box equal to <code>\c_void_box</code> within the current \TeX group level.
<hr/> <code>\box_gclear:N</code> <code>\box_gclear:c</code> <hr/>	<code>\box_gclear:N</code> $\langle box \rangle$ Clears the content of the $\langle box \rangle$ by setting the box equal to <code>\c_void_box</code> globally.
<hr/> <code>\box_clear_new:N</code> <code>\box_clear_new:c</code> <hr/>	<code>\box_clear_new:N</code> $\langle box \rangle$ If the $\langle box \rangle$ is not defined, globally creates it. If the $\langle box \rangle$ is defined, clears the content of the $\langle box \rangle$ by setting the box equal to <code>\c_void_box</code> within the current \TeX group level.
<hr/> <code>\box_gclear_new:N</code> <code>\box_gclear_new:c</code> <hr/>	<code>\box_gclear_new:N</code> $\langle box \rangle$ If the $\langle box \rangle$ is not defined, globally creates it. If the $\langle box \rangle$ is defined, clears the content of the $\langle box \rangle$ by setting the box equal to <code>\c_void_box</code> globally.
<hr/> <code>\box_set_eq:NN</code> <code>\box_set_eq:(cN Nc cc)</code> <hr/>	<code>\box_set_eq:NN</code> $\langle box1 \rangle$ $\langle box2 \rangle$ Sets the content of $\langle box1 \rangle$ equal to that of $\langle box2 \rangle$. This assignment is restricted to the current \TeX group level.
<hr/> <code>\box_gset_eq:NN</code> <code>\box_gset_eq:(cN Nc cc)</code> <hr/>	<code>\box_gset_eq:NN</code> $\langle box1 \rangle$ $\langle box2 \rangle$ Sets the content of $\langle box1 \rangle$ equal to that of $\langle box2 \rangle$ globally.

<code>\box_set_eq_clear:NN</code>	<code>\box_set_eq_clear:NN <box1> <box2></code>
<code>\box_set_eq_clear:(cN Nc cc)</code>	Sets the content of $\langle box1 \rangle$ within the current TeX group equal to that of $\langle box2 \rangle$, then clears $\langle box2 \rangle$ globally.

<code>\box_gset_eq_clear:NN</code>	<code>\box_gset_eq_clear:NN <box1> <box2></code>
<code>\box_gset_eq_clear:(cN Nc cc)</code>	Sets the content of $\langle box1 \rangle$ equal to that of $\langle box2 \rangle$, then clears $\langle box2 \rangle$. These assignments are global.

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.

TeXhackers note: This is the TeX 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 $\langle box \rangle$.

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

<code>\box_move_right:nn</code>	<code>\box_move_right:nn {<dimexpr>} {<box function>}</code>
<code>\box_move_left:nn</code>	This function operates in vertical mode, and inserts the material specified by the $\langle box \text{ function} \rangle$ such that its reference point is displaced horizontally by the given $\langle dimexpr \rangle$ from the reference point for typesetting, to the right or left as appropriate. The $\langle box \text{ function} \rangle$ should be a box operation such as <code>\box_use:N \<box></code> or a “raw” box specification such as <code>\vbox:n { xyz }</code> .

<code>\box_move_up:nn</code>	<code>\box_move_up:nn {<dimexpr>} {<box function>}</code>
<code>\box_move_down:nn</code>	This function operates in horizontal mode, and inserts the material specified by the $\langle box \text{ function} \rangle$ such that its reference point is displaced vertical by the given $\langle dimexpr \rangle$ from the reference point for typesetting, up or down as appropriate. The $\langle box \text{ function} \rangle$ should be a box operation such as <code>\box_use:N \<box></code> or a “raw” box specification such as <code>\vbox:n { xyz }</code> .

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 $\langle dimension expression \rangle$.

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 $\langle dimension expression \rangle$.

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

<code>\box_wd:N</code>	<code>\box_wd:N <box></code>
<code>\box_wd:c</code>	Calculates the width of the $\langle box \rangle$ in a form suitable for use in a $\langle dimension expression \rangle$.

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

<code>\box_set_dp:Nn</code>	<code>\box_set_dp:Nn <box> {\langle dimension expression \rangle}</code>
<code>\box_set_dp:cn</code>	Set the depth (below the baseline) of the $\langle box \rangle$ to the value of the $\{\langle dimension expression \rangle\}$. This is a global assignment.

Updated: 2011-10-22

<code>\box_set_ht:Nn</code>	<code>\box_set_ht:Nn <box> {\langle dimension expression \rangle}</code>
<code>\box_set_ht:cn</code>	Set the height (above the baseline) of the $\langle box \rangle$ to the value of the $\{\langle dimension expression \rangle\}$. This is a global assignment.

Updated: 2011-10-22

<code>\box_set_wd:Nn</code>	<code>\box_set_wd:Nn <box> {\langle dimension expression \rangle}</code>
<code>\box_set_wd:cn</code>	Set the width of the $\langle box \rangle$ to the value of the $\{\langle dimension expression \rangle\}$. 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 T_EX 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.

`\box_resize:Nnn`**`\box_resize:cnn`**

New: 2011-09-02

`\box_resize:Nnn` $\langle box \rangle$ $\{\langle x-size \rangle\}$ $\{\langle y-size \rangle\}$

Resize the $\langle box \rangle$ to $\langle x-size \rangle$ horizontally and $\langle y-size \rangle$ vertically (both of the sizes are dimension expressions). 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. Negative sizes 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_ht_plus_dp:Nn`**`\box_resize_to_ht_plus_dp:cn`**

New: 2011-09-02

Updated: 2011-10-22

`\box_resize_to_ht_plus_dp:Nn` $\langle box \rangle$ $\{\langle y-size \rangle\}$

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:Nn`**`\box_resize_to_wd:cn`**

New: 2011-09-02

Updated: 2011-10-22

`\box_resize_to_wd:Nnn` $\langle box \rangle$ $\{\langle x-size \rangle\}$

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:cn`**

New: 2011-09-02

Updated: 2011-10-22

`\box_rotate:Nn` $\langle box \rangle$ $\{\langle angle \rangle\}$

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:cnn`**

New: 2011-09-02

Updated: 2011-10-22

`\box_scale:Nnn` $\langle box \rangle$ $\{\langle x-scale \rangle\}$ $\{\langle y-scale \rangle\}$

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 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/> <hr/> <code>\hbox_to_wd:nn</code>	<code>\hbox_to_wd:nn {<dimexpr>} {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of width $\langle dimexpr \rangle$ and then includes this box in the current list for typesetting.
<hr/> <hr/> <code>\hbox_to_zero:n</code>	<code>\hbox_to_zero:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width and then includes this box in the current list for typesetting.
<hr/> <hr/> <code>\hbox_set:Nn</code> <code>\hbox_set:cn</code>	<code>\hbox_set:Nn <box> {<contents>}</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is local.
<hr/> <hr/> <code>\hbox_gset:Nn</code> <code>\hbox_gset:cn</code>	<code>\hbox_gset:Nn <box> {<contents>}</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is global.
<hr/> <hr/> <code>\hbox_set_to_wd:Nnn</code> <code>\hbox_set_to_wd:cnn</code>	<code>\hbox_set_to_wd:Nnn <box> {<dimexpr>} {<contents>}</code> Typesets the $\langle contents \rangle$ to the width given by the $\langle dimexpr \rangle$ and then stores the result inside the $\langle box \rangle$. The assignment is local.
<hr/> <hr/> <code>\hbox_gset_to_wd:Nnn</code> <code>\hbox_gset_to_wd:cnn</code>	<code>\hbox_gset_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 $\langle box \rangle$. The assignment is global.
<hr/> <hr/> <code>\hbox_overlap_right:n</code>	<code>\hbox_overlap_right:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the right of the insertion point.
<hr/> <hr/> <code>\hbox_overlap_left:n</code>	<code>\hbox_overlap_left:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the left of the insertion point.
<hr/> <hr/> <code>\hbox_set:Nw</code> <code>\hbox_set:cw</code> <code>\hbox_set_end</code>	<code>\hbox_set:Nw <box> <contents> \hbox_set_end:</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is local. 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.

<code>\hbox_gset:Nw</code>	<code>\hbox_gset:Nw <box> <contents> \hbox_gset_end:</code>
<code>\hbox_gset:cw</code>	
<code>\hbox_gset_end</code>	Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is global. 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.

<code>\hbox_unpack:N</code>	<code>\hbox_unpack:N <box></code>
<code>\hbox_unpack:c</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`.

<code>\hbox_unpack_clear:N</code>	<code>\hbox_unpack_clear:N <box></code>
<code>\hbox_unpack_clear:c</code>	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.

<code>\vbox:n</code>	<code>\vbox:n {<contents>}</code>
	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`.

<code>\vbox_top:n</code>	<code>\vbox_top:n {<contents>}</code>
	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 <i>first</i> item added to the box.

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

<hr/> <hr/> <code>\vbox_to_ht:nn</code>	<code>\vbox_to_ht:nn {<dimexpr>} {<contents>}</code> 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.
<hr/> <hr/> <code>\vbox_to_zero:n</code>	<code>\vbox_to_zero:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a vertical box of zero height and then includes this box in the current list for typesetting.
<hr/> <hr/> <code>\vbox_set:Nn</code> <code>\vbox_set:cn</code>	<code>\vbox_set:Nn <box> {<contents>}</code> Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. The assignment is local.
<hr/> <hr/> <code>\vbox_gset:Nn</code> <code>\vbox_gset:cn</code>	<code>\vbox_gset:Nn <box> {<contents>}</code> Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. The assignment is global.
<hr/> <hr/> <code>\vbox_set_top:Nn</code> <code>\vbox_set_top:cn</code>	<code>\vbox_set_top:Nn <box> {<contents>}</code> 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 <i>first</i> item added to the box. The assignment is local.
<hr/> <hr/> <code>\vbox_gset_top:Nn</code> <code>\vbox_gset_top:cn</code>	<code>\vbox_gset_top:Nn <box> {<contents>}</code> 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 <i>first</i> item added to the box. The assignment is global.
<hr/> <hr/> <code>\vbox_set_to_ht:Nnn</code> <code>\vbox_set_to_ht:cn</code>	<code>\vbox_set_to_ht:Nnn <box> {<dimexpr>} {<contents>}</code> Typesets the $\langle contents \rangle$ to the height given by the $\langle dimexpr \rangle$ and then stores the result inside the $\langle box \rangle$. The assignment is local.
<hr/> <hr/> <code>\vbox_gset_to_ht:Nnn</code> <code>\vbox_gset_to_ht:cn</code>	<code>\vbox_gset_to_ht:Nnn <box> {<dimexpr>} {<contents>}</code> Typesets the $\langle contents \rangle$ to the height given by the $\langle dimexpr \rangle$ and then stores the result inside the $\langle box \rangle$. The assignment is global.
<hr/> <hr/> <code>\vbox_set:Nw</code> <code>\vbox_set:cw</code> <code>\vbox_set_end</code>	<code>\vbox_begin:Nw <box> <contents> \vbox_set_end:</code> Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. The assignment is local. In contrast to <code>\vbox_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.

```
\vbox_gset:Nw
\vbox_gset:cw
\vbox_gset_end
```

```
\vbox_gset:Nw <box> <contents> \vbox_gset_end:
```

Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. The assignment is global. 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 <box1> <box2> {<dimexpr>}
```

Sets $\langle box1 \rangle$ to contain material to the height given by the $\langle dimexpr \rangle$ by removing content from the top of $\langle box2 \rangle$ (which must be a vertical box).

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

```
\vbox_unpack:N
\vbox_unpack:c
```

```
\vbox_unpack:N <box>
```

Unpacks the content of the vertical $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set.

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

```
\vbox_unpack_clear:N
\vbox_unpack_clear:c
```

```
\vbox_unpack:N <box>
```

Unpacks the content of the vertical $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set. The $\langle box \rangle$ is then cleared globally.

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

137 Primitive box conditionals

```
\if_hbox:N ★
```

```
\if_hbox:N <box>
  <true code>
\else:
  <false code>
\fi:
```

Tests if $\langle box \rangle$ is a horizontal box.

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

<code>\if_vbox:N</code>	<code>★</code>	<code>\if_vbox:N</code>	<code>⟨box⟩</code>
		<code>⟨true code⟩</code>	
		<code>\else:</code>	
		<code>⟨false code⟩</code>	
		<code>\fi:</code>	

Tests is `⟨box⟩` is a vertical box.

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

<code>\if_box_empty:N</code>	<code>★</code>	<code>\if_box_empty:N</code>	<code>⟨box⟩</code>
		<code>⟨true code⟩</code>	
		<code>\else:</code>	
		<code>⟨false code⟩</code>	
		<code>\fi:</code>	

Tests is `⟨box⟩` is an empty (void) box.

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

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).

138 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.

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

140 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 `\TotalHeight`, `\Height`, `\Depth` and `\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` $\langle coffin \rangle$ $\{ \langle x-scale \rangle \}$ $\{ \langle y-scale \rangle \}$

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.

141 Joining and using coffins

`\coffin_attach:NnnNnnnn`

`\coffin_attach:(cnnNnnnn|NnnNnnnn|cnnNnnnn)`

`\coffin_attach:NnnNnnnn`

$\langle coffin1 \rangle$ $\{ \langle coffin1-pole1 \rangle \}$ $\{ \langle coffin1-pole2 \rangle \}$

$\langle coffin2 \rangle$ $\{ \langle coffin2-pole1 \rangle \}$ $\{ \langle coffin2-pole2 \rangle \}$

$\{ \langle x-offset \rangle \}$ $\{ \langle y-offset \rangle \}$

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|NnnNnnnn|cnnNnnnn)`

`\coffin_join:NnnNnnnn`

$\langle coffin1 \rangle$ $\{ \langle coffin1-pole1 \rangle \}$ $\{ \langle coffin1-pole2 \rangle \}$

$\langle coffin2 \rangle$ $\{ \langle coffin2-pole1 \rangle \}$ $\{ \langle coffin2-pole2 \rangle \}$

$\{ \langle x-offset \rangle \}$ $\{ \langle y-offset \rangle \}$

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` $\langle coffin \rangle$ $\{ \langle pole1 \rangle \}$ $\{ \langle pole2 \rangle \}$

$\{ \langle x-offset \rangle \}$ $\{ \langle y-offset \rangle \}$

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.

142 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 $\langle coffin \rangle$ in a form suitable for use in a $\langle dimension expression \rangle$.
<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 $\langle coffin \rangle$ in a form suitable for use in a $\langle dimension expression \rangle$.
<hr/> <code>\coffin_wd:N</code> <hr/>	<code>\coffin_wd:N <coffin></code>
<code>\coffin_wd:c</code> <hr/>	Calculates the width of the $\langle coffin \rangle$ in a form suitable for use in a $\langle dimension expression \rangle$.

143 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 $\langle poles \rangle$ of the $\langle coffin \rangle$ to give a set of $\langle handles \rangle$. It then prints the $\langle coffin \rangle$ at the current location in the source, with the position of the $\langle handles \rangle$ marked on the coffin. The $\langle handles \rangle$ will be labelled as part of this process: the locations of the $\langle handles \rangle$ and the labels are both printed in the $\langle colour \rangle$ 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 $\langle handle \rangle$ for the $\langle coffin \rangle$ as defined by the intersection of $\langle pole_1 \rangle$ and $\langle pole_2 \rangle$. It then marks the position of the $\langle handle \rangle$ on the $\langle coffin \rangle$. The $\langle handle \rangle$ will be labelled as part of this process: the location of the $\langle handle \rangle$ and the label are both printed in the $\langle colour \rangle$ 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 $\langle coffin \rangle$ in the terminal. The width, height and depth of the typeset material are given, along with the location of all of the poles of the coffin.
	Notice that the poles of a coffin are defined by four values: the x and y co-ordinates of a point that the pole passes through and the x - and y -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.

144 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).

145 Managing streams

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

New: 2011-09-26

`\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: 2011-09-26

`\ior_open:Nn` $\langle stream \rangle$ $\{\langle file name \rangle\}$

Opens $\langle file name \rangle$ for reading using $\langle stream \rangle$ as the control sequence for file access. If the $\langle stream \rangle$ was already open it is closed before the new operation begins. The $\langle stream \rangle$ is available for access immediately and will remain allocated to $\langle file name \rangle$ until a `\ior_close:N` instruction is given or the file ends.

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

Updated: 2011-09-26

`\iow_open:Nn` $\langle stream \rangle$ $\{\langle file name \rangle\}$

Opens $\langle file name \rangle$ for writing using $\langle stream \rangle$ as the control sequence for file access. If the $\langle stream \rangle$ was already open it is closed before the new operation begins. The $\langle stream \rangle$ is available for access immediately and will remain allocated to $\langle file name \rangle$ until a `\iow_close:N` instruction is given or the file ends. Opening a file for writing will clear any existing content in the file (*i.e.* writing is *not* additive).

<code>\ior_close:N</code>	<code>\ior_close:N <stream></code>
<code>\ior_close:c</code>	
Updated: 2011-09-26	

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.

<code>\iow_close:N</code>	<code>\iow_close:N <stream></code>
<code>\iow_close:c</code>	
Updated: 2011-09-26	

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.

<code>\ior_list_streams</code>	<code>\ior_list_streams:</code>
<code>\iow_list_streams</code>	<code>\iow_list_streams:</code>

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

146 Writing to files

<code>\iow_now:Nn</code>	<code>\iow_now:Nn <stream> {\tokens}</code>
<code>\iow_now:Nx</code>	

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`.

<code>\iow_log:n</code>	<code>\iow_log:n {\tokens}</code>
<code>\iow_log:x</code>	

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

<code>\iow_term:n</code>	<code>\iow_term:n {\tokens}</code>
<code>\iow_term:x</code>	

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

<code>\iow_now_when_avail:Nn</code>	<code>\iow_now_when_avail:Nn <stream> {\tokens}</code>
<code>\iow_now_when_avail:Nx</code>	

If $\langle stream \rangle$ is open, writes the $\langle tokens \rangle$ to the $\langle stream \rangle$ in the same manner as `\iow_now:Nn`. If the $\langle stream \rangle$ is not open, the $\langle tokens \rangle$ are simply thrown away.

<code>\iow_shipout:Nn</code>	<code>\iow_shipout:Nn <stream> {\tokens}</code>
<code>\iow_shipout:Nx</code>	

This functions writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ when the current page is finalised (*i.e.* at shipout). The x-type variants expand the $\langle tokens \rangle$ at the point where the function is used but *not* when the resulting tokens are written to the $\langle stream \rangle$ (*cf.* `\iow_shipout_x:Nn`).

<code>\iow_shipout_x:Nn</code> <code>\iow_shipout_x:Nx</code>	<code>\iow_shipout_x:Nn <stream> {<tokens>}</code> <p>This functions writes <i><tokens></i> to the specified <i><stream></i> when the current page is finalised (<i>i.e.</i> at shipout). The <i><tokens></i> are expanded at the time of writing in addition to any expansion when the function is used. This makes these functions suitable for including material finalised during the page building process (such as the page number integer).</p>
--	---

TeXhackers note: `\iow_shipout_x:Nn` is the TeX primitive `\write` renamed.

<code>\iow_char:N</code> ★	<code>\iow_char:N <token></code> <p>Inserts <i><token></i> into the output stream. Useful when trying to write difficult characters such as %, {, }, <i>etc.</i> in messages, for example:</p>
----------------------------	---

`\iow_now:Nx \g_my_stream { \iow_char:N \{ text \iow_char:N \} }`

The function has no effect if writing is taking place without expansion (*e.g.* in the second argument of `\iow_now:Nn`).

<code>\iow_newline</code> ★	<code>\iow_newline:</code> <p>Function to add a new line within the <i><tokens></i> written to a file. The function has no effect if writing is taking place without expansion (<i>e.g.</i> in the second argument of <code>\iow_now:Nn</code>).</p>
-----------------------------	---

147 Wrapping lines in output

<hr/> <code>\iow_wrap:xnnnN</code> <hr/>	<code>\iow_wrap:xnnnN {<text>} {<run-on text>} {<run-on length>} {<set up>} <function></code>
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>\#, \%, \{, \}, \sim</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 {<text>}</code>
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	

148 Reading from files

`\ior_to:NN` `\ior_to:NN` $\langle stream \rangle$ $\langle token\ list\ variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result locally in the $\langle token\ list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ will be tokenized by T_EX according to the category codes in force when the function is used.

T_EXhackers note: The is protected macro which expands to the T_EX primitive `\read` along with the `to` keyword.

`\ior_gto:NN` `\ior_gto:NN` $\langle stream \rangle$ $\langle token\ list\ variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result globally in the $\langle token\ list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ will be tokenized by T_EX according to the category codes in force when the function is used.

T_EXhackers note: The is protected macro which expands to the T_EX primitives `\global\read` along with the `to` keyword.

`\ior_str_to:NN` `\ior_str_to:NN` $\langle stream \rangle$ $\langle token\ list\ variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result locally in the $\langle token\ list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space).

T_EXhackers note: The is protected macro which expands to the ε -T_EX primitive `\readline` along with the `to` keyword.

`\ior_str_gto:NN` `\ior_str_gto:NN` $\langle stream \rangle$ $\langle token\ list\ variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result globally in the $\langle token\ list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space).

T_EXhackers note: The is protected macro which expands to the primitives `\global\readline` along with the `to` keyword.

<code>\ior_if_eof_p:N</code> ★	<code>\ior_if_eof_p:N</code> $\langle stream \rangle$
<code>\ior_if_eof:NTF</code> ★	<code>\ior_if_eof:NTF</code> $\langle stream \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Updated: 2011-09-26

Tests if the end of a $\langle stream \rangle$ has been reached during a reading operation. The test will also return a `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.

149 Internal input–output functions

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

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

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

<code>\ior_raw_new:N</code> <code>\ior_raw_new:c</code>	<code>\ior_raw_new:N</code> $\langle stream \rangle$
--	--

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.

<code>\iow_raw_new:N</code> <code>\iow_raw_new:c</code>	<code>\iow_raw_new:N</code> $\langle stream \rangle$
--	--

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_EX level, when a new stream is requested by the stack itself.

Part XIX

The l3msg package

Messages

Messages need to be passed to the user by modules, either when errors occur or to indicate how the code is proceeding. The `l3msg` module provides a consistent method for doing this (as opposed to writing directly to the terminal or log).

The system used by `l3msg` to create messages divides the process into two distinct parts. Named messages are created in the first part of the process; at this stage, no decision is made about the type of output that the message will produce. The second part of the process is actually producing a message. At this stage a choice of message *class* has to be made, for example `error`, `warning` or `info`.

By separating out the creation and use of messages, several benefits are available. First, the messages can be altered later without needing details of where they are used in the code. This makes it possible to alter the language used, the detail level and so on. Secondly, the output which results from a given message can be altered. This can be done on a message class, module or message name basis. In this way, message behaviour can be altered and messages can be entirely suppressed.

150 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_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.

151 Contextual information for messages

<hr/> <code>\msg_line_context</code> ☆ <hr/>	<code>\msg_line_context:</code> Prints the current line number when a message is given, and thus suitable for giving context to messages. The number itself is proceeded by the text <code>on line</code> .
<hr/> <code>\msg_line_number</code> ☆ <hr/>	<code>\msg_line_number:</code> Prints the current line number when a message is given.
<hr/> <code>\c_msg_return_text_tl</code> <hr/>	Standard text to indicate that the user should try pressing <code><return></code> to continue. The standard definition reads: Try typing <code><return></code> to proceed. If that doesn't work, type <code>X <return></code> to quit.
<hr/> <code>\c_msg_trouble_text_tl</code> <hr/>	Standard text to indicate that the more errors are likely and that aborting the run is advised. The standard definition reads: More errors will almost certainly follow: the LaTeX run should be aborted.
<hr/> <code>\msg_fatal_text:n</code> ☆ <hr/>	<code>\msg_fatal_text:n {<module>}</code> Produces the standard text: Fatal <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.
<hr/> <code>\msg_critical_text:n</code> ☆ <hr/>	<code>\msg_critical_text:n {<module>}</code> Produces the standard text: Critical <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.
<hr/> <code>\msg_error_text:n</code> ☆ <hr/>	<code>\msg_error_text:n {<module>}</code> Produces the standard text: <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.

<code>\msg_warning_text:n</code>	★	<code>\msg_warning_text:n {<module>}</code>
----------------------------------	---	---

Produces the standard text:

`<module> warning`

This function can be redefined to alter the language in which the message is give, using #1 as the name of the `<module>` to be included.

<code>\msg_info_text:n</code>	★	<code>\msg_info_text:n {<module>}</code>
-------------------------------	---	--

Produces the standard text:

`<module> info`

This function can be redefined to alter the language in which the message is give, using #1 as the name of the `<module>` to be included.

152 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).

153 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 }
```

154 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 \TeX run, issuing a formatted message comprising $\langle first\ line \rangle$ and $\langle text \rangle$ laid out in the format

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! <first line>
!
! <text>
!.....

```

where the $\langle text \rangle$ will be wrapped to fit within the current line length. The user may then request more information, at which stage the $\langle extra\ text \rangle$ will be shown in the terminal in the format

```

|,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
|  <extra text>
|.....

```

where the $\langle extra\ text \rangle$ will be wrapped to fit within the current line length.

`\msg_log:x` `\msg_log:x {<text>}`
 Writes to the log file with the $\langle text \rangle$ laid out in the format

```

.....
. <text>
.....

```

where the $\langle text \rangle$ 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 $\langle text \rangle$ laid out in the format

```

*****
* <text>
*****

```

where the $\langle text \rangle$ will be wrapped to fit within the current line length.

155 Kernel-specific functions

Messages from $\text{\LaTeX}3$ itself are handled by the general message system, but have their own functions. This allows some text to be pre-defined, and also ensures that serious errors can be handled properly.

```
\msg_kernel_new:nnnn
\msg_kernel_new:nnn
```

Updated: 2011-08-16

```
\msg_kernel_new:nnnn {<module>} {<message>} {<text>} {<more text>}
```

Creates a kernel *<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_kernel_set:nnnn
\msg_kernel_set:nnn
```

```
\msg_kernel_set:nnnn {<module>} {<message>} {<text>} {<more text>}
```

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.

```
\msg_kernel_fatal:nnxxxx
\msg_kernel_fatal:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_fatal:nnxxxx {<module>} {<message>} {<arg one>}
{<arg two>} {<arg three>} {<arg four>}
```

Issues kernel *<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.

```
\msg_kernel_error:nnxxxx
\msg_kernel_error:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_error:nnxxxx {<module>} {<message>} {<arg one>}
{<arg two>} {<arg three>} {<arg four>}
```

Issues kernel *<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.

```
\msg_kernel_warning:nnxxxx
\msg_kernel_warning:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_warning:nnxxxx {<module>} {<message>} {<arg one>}
{<arg two>} {<arg three>} {<arg four>}
```

Issues kernel *<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.

```
\msg_kernel_info:nnxxxx
\msg_kernel_info:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_info:nnxxxx {<module>} {<message>} {<arg one>}
{<arg two>} {<arg three>} {<arg four>}
```

Issues kernel *<module>* information *<message>*, passing *<arg one>* to *<arg four>* to the text-creating functions. The information text will be added to the log file.

156 Expandable errors

In a few places, the L^AT_EX3 kernel needs to produce errors in an expansion only context. This must be handled very differently from normal error messages, as none of the tools to print to the terminal or the log file are expandable.

<hr/> <code>\msg_expandable_error:n</code> <hr/>	<code>\msg_expandable_error:n {\langle error message \rangle}</code>
New: 2011-08-11	Issues an “Undefined error” message from T _E X itself, and prints the $\langle error\ message \rangle$.
Updated: 2011-08-13	The $\langle error\ message \rangle$ must be short: it is cropped at the end of one line.

T_EXhackers note: This function expands to an empty token list after two steps. Tokens inserted in response to T_EX’s prompt are read with the current category code setting, and inserted just after the place where the error message was issued.

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 158, 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`.

157 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 159.</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 159.</p> <p>This property is experimental.</p>
<hr/> .choice_code:n <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 159.</p>
<hr/> .clist_set:N <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/> .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/> .generate_choices:n <hr/>	$\langle key \rangle$.generate_choices:n = { $\langle list \rangle$ }	This property will mark $\langle key \rangle$ as a multiple choice key, and will use the $\langle list \rangle$ to define the choices. The $\langle list \rangle$ should consist of a comma-separated list of choice names. Each choice will be set up to execute $\langle code \rangle$ as set using .choice_code:n (or .choice_code:x). Choices are discussed in detail in section 159.
<hr/> .int_set:N <hr/> .int_set:c <hr/>	$\langle key \rangle$.int_set:N = $\langle integer \rangle$	Defines $\langle key \rangle$ to set $\langle integer \rangle$ to $\langle value \rangle$ (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned locally.
<hr/> .int_gset:N <hr/> .int_gset:c <hr/>	$\langle key \rangle$.int_gset:N = $\langle integer \rangle$	Defines $\langle key \rangle$ to set $\langle integer \rangle$ to $\langle value \rangle$ (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned globally.
<hr/> .meta:n <hr/> .meta:x <hr/>	$\langle key \rangle$.meta:n = { $\langle keyval list \rangle$ }	Makes $\langle key \rangle$ a meta-key, which will set $\langle keyval list \rangle$ in one go. If $\langle key \rangle$ is given with a value at the time the key is used, then the value will be passed through to the subsidiary $\langle keys \rangle$ for processing (as #1).
<hr/> .multichoice: <hr/> <small>New: 2011-08-21</small> <hr/>	$\langle key \rangle$.multichoice:	Sets $\langle key \rangle$ to act as a multiple choice key. Each valid choice for $\langle key \rangle$ must then be created, as discussed in section 159. This property is experimental.
<hr/> .multichoice:nn <hr/> <small>New: 2011-08-21</small> <hr/>	$\langle key \rangle$.multichoice:nn $\langle choices \rangle$ $\langle code \rangle$	Sets $\langle key \rangle$ to act as a multiple choice key, and defines a series $\langle choices \rangle$ which are implemented using the $\langle code \rangle$. Inside $\langle code \rangle$, \l_keys_choice_tl will be the name of the choice made, and \l_keys_choice_int will be the position of the choice in the list of $\langle choices \rangle$ (indexed from 0). Choices are discussed in detail in section 159. This property is experimental.
<hr/> .skip_set:N <hr/> .skip_set:c <hr/>	$\langle key \rangle$.skip_set:N = $\langle skip \rangle$	Defines $\langle key \rangle$ to set $\langle skip \rangle$ to $\langle value \rangle$ (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle skip \rangle$ will be assigned locally.
<hr/> .skip_gset:N <hr/> .skip_gset:c <hr/>	$\langle key \rangle$.skip_gset:N = $\langle skip \rangle$	Defines $\langle key \rangle$ to set $\langle skip \rangle$ to $\langle value \rangle$ (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle skip \rangle$ will be assigned globally.

<hr/> <code>.tl_set:N</code> <hr/>	<code><key> .tl_set:N = <token list variable></code>
<code>.tl_set:c</code>	Defines <code><key></code> to set <code><token list variable></code> to <code><value></code> . If the variable does not exist, it will be created at the point that the key is set up. The <code><token list variable></code> will be assigned locally.
<hr/> <code>.tl_gset:N</code> <hr/>	<code><key> .tl_gset:N = <token list variable></code>
<code>.tl_gset:c</code>	Defines <code><key></code> to set <code><token list variable></code> to <code><value></code> . If the variable does not exist, it will be created at the point that the key is set up. The <code><token list variable></code> will be assigned globally.
<hr/> <code>.tl_set_x:N</code> <hr/>	<code><key> .tl_set_x:N = <token list variable></code>
<code>.tl_set_x:c</code>	Defines <code><key></code> to set <code><token list variable></code> to <code><value></code> , 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 <code><token list variable></code> will be assigned locally.
<hr/> <code>.tl_gset_x:N</code> <hr/>	<code><key> .tl_gset_x:N = <token list variable></code>
<code>.tl_gset_x:c</code>	Defines <code><key></code> to set <code><token list variable></code> to <code><value></code> , 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 <code><token list variable></code> will be assigned globally.
<hr/> <code>.value_forbidden:</code> <hr/>	<code><key> .value_forbidden:</code>
	Specifies that <code><key></code> cannot receive a <code><value></code> when used. If a <code><value></code> is given then an error will be issued.
<hr/> <code>.value_required:</code> <hr/>	<code><key> .value_required:</code>
	Specifies that <code><key></code> must receive a <code><value></code> when used. If a <code><value></code> is not given then an error will be issued.

158 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.

159 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.

160 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.

161 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>*.

162 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>*.

163 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 <function1> <function2> {<key-value list>}
```

Parses the *<key-value list>* into a series of *<keys>* and associated *<values>*, or keys alone (if no *<value>* was given). *<function1>* should take one argument, while *<function2>* should absorb two arguments. After `\keyval_parse:NNn` has parsed the *<key-value list>*, *<function1>* will be used to process keys given with no value and *<function2>* will be used to process keys given with a value. The order of the *<keys>* in the *<key-value list>* 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 *<key>* and *<value>*, and any *outer* set of braces are removed from the *<value>* 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.

164 File operation functions

`\g_file_current_name_tl`

Contains the name of the current L^AT_EX file. This variable should not be modified: it is intended for information only. It will be equal to `\c_job_name_tl` at the start of a L^AT_EX run and will be modified each time a file is read using `\file_input:n`.

`\file_if_exist:nTF`

`\file_if_exist:nTF {<file name>} {<true code>} {<false code>}`

Searches for `<file name>` using the current T_EX search path and the additional paths controlled by `\file_path_include:n`.

T_EXhackers note: The `<file name>` may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

`\file_add_path:nN`

`\file_add_path:nN {<file name>} <tl var>`

Searches for `<file name>` in the path as detailed for `\file_if_exist:nTF`, and if found sets the `<tl var>` the fully-qualified name of the file, *i.e.* the path and file name. If the file is not found then the `<tl var>` will be empty.

T_EXhackers note: The `<file name>` may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

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

Searches for *<file name>* in the path as detailed for `\file_if_exist:nTF`, and if found reads in the file as additional L^AT_EX source. All files read are recorded for information and the file name stack is updated by this function.

T_EXhackers note: The *<file name>* may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

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

Adds *<path>* to the list of those used to search for files by the `\file_input:n` and `\file_if_exist:n` function. The assignment is local.

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

Removes *<path>* from the list of those used to search for files by the `\file_input:n` and `\file_if_exist:n` function. The assignment is local.

<code>\file_list</code>	<code>\file_list:</code>
-------------------------	--------------------------

This function will list all files loaded using `\file_input:n` in the log file.

165 Internal file functions

<code>\g_file_stack_seq</code>	
--------------------------------	--

Stores the stack of nested files loaded using `\file_input:n`. This is needed to restore the appropriate file name to `\g_file_current_name_tl` at the end of each file.

<code>\g_file_record_seq</code>	
---------------------------------	--

Stores the name of every file loaded using `\file_input:n`. In contrast to `\g_file_stack_seq`, no items are ever removed from this sequence.

<code>\l_file_name_tl</code>	
------------------------------	--

Used to return the full name of a file for internal use.

<code>\l_file_search_path_seq</code>	
--------------------------------------	--

The sequence of file paths to search when loading a file.

<code>\l_file_search_path_saved_seq</code>	
--	--

When loaded on top of L^AT_EX 2_ε, there is a need to save the search path so that `\input@path` can be used as appropriate.

<code>\l_file_tmpa_seq</code>	
-------------------------------	--

When loaded on top of L^AT_EX 2_ε, there is a need to convert the comma lists `\input@path` and `\@filelist` to sequences.

New: 2011-09-06

Part XXII

The l3fp package

Floating-point operations

A floating point number is one which is stored as a mantissa and a separate exponent. This module implements arithmetic using radix 10 floating point numbers. This means that the mantissa should be a real number in the range $1 \leq |x| < 10$, with the exponent given as an integer between -99 and 99 . In the input, the exponent part is represented starting with an `e`. As this is a low-level module, error-checking is minimal. Numbers which are too large for the floating point unit to handle will result in errors, either from `TeX` or from `LATeX`. The `LATeX` code does not check that the input will not overflow, hence the possibility of a `TeX` error. On the other hand, numbers which are too small will be dropped, which will mean that extra decimal digits will simply be lost.

When parsing numbers, any missing parts will be interpreted as zero. So for example

```
\fp_set:Nn \l_my_fp { }  
\fp_set:Nn \l_my_fp { . }  
\fp_set:Nn \l_my_fp { - }
```

will all be interpreted as zero values without raising an error.

Operations which give an undefined result (such as division by 0) will not lead to errors. Instead special marker values are returned, which can be tested for using for example `\fp_if_undefined:N(TF)`. In this way it is possible to work with asymptotic functions without first checking the input. If these special values are carried forward in calculations they will be treated as 0.

Floating point numbers are stored in the `fp` floating point variable type. This has a standard range of functions for variable management.

166 Floating-point variables

<code>\fp_new:N</code>	<code>\fp_new:N</code> <i><floating point variable></i>
------------------------	---

<code>\fp_new:c</code>	Creates a new <i><floating point variable></i> or raises an error if the name is already taken. The declaration global. The <i><floating point></i> will initially be set to <code>+0.000000000e0</code> (the zero floating point).
------------------------	---

<code>\fp_const:Nn</code>	<code>\fp_const:Nn</code> <i><floating point variable></i> <i>{<value>}</i>
---------------------------	---

<code>\fp_const:cn</code>	Creates a new constant <i><floating point variable></i> or raises an error if the name is already taken. The value of the <i><floating point variable></i> will be set globally to the <i><value></i> .
---------------------------	---

<code>\fp_set_eq:NN</code>	<code>\fp_set_eq:NN</code> <i><fp var1></i> <i><fp var2></i>
----------------------------	--

<code>\fp_set_eq:(cN Nc cc)</code>	Sets the value of <i><floating point variable1></i> equal to that of <i><floating point variable2></i> . This assignment is restricted to the current <code>TeX</code> group level.
------------------------------------	---

<hr/> <code>\fp_gset_eq:NN</code> <code>\fp_gset_eq:(cN Nc cc)</code> <hr/>	<code>\fp_gset_eq:NN <fp var1> <fp var2></code> Sets the value of <i><floating point variable1></i> equal to that of <i><floating point variable2></i> . This assignment is global and so is not limited by the current \TeX group level.
<hr/> <code>\fp_zero:N</code> <code>\fp_zero:c</code> <hr/>	<code>\fp_zero:N <floating point variable></code> Sets the <i><floating point variable></i> to +0.000000000e0 within the current scope.
<hr/> <code>\fp_gzero:N</code> <code>\fp_gzero:c</code> <hr/>	<code>\fp_gzero:N <floating point variable></code> Sets the <i><floating point variable></i> to +0.000000000e0 globally.
<hr/> <code>\fp_set:Nn</code> <code>\fp_set:cn</code> <hr/>	<code>\fp_set:Nn <floating point variable> {<value>}</code> Sets the <i><floating point variable></i> variable to <i><value></i> within the scope of the current \TeX group.
<hr/> <code>\fp_gset:Nn</code> <code>\fp_gset:cn</code> <hr/>	<code>\fp_gset:Nn <floating point variable> {<value>}</code> Sets the <i><floating point variable></i> variable to <i><value></i> globally.
<hr/> <code>\fp_set_from_dim:Nn</code> <code>\fp_set_from_dim:cn</code> <hr/>	<code>\fp_set_from_dim:Nn <floating point variable> {<dimexpr>}</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> . The assignment is local.
<hr/> <code>\fp_gset_from_dim:Nn</code> <code>\fp_gset_from_dim:cn</code> <hr/>	<code>\fp_gset_from_dim:Nn <floating point variable> {<dimexpr>}</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> . The assignment is global.
<hr/> <code>\fp_use:N</code> ☆ <code>\fp_use:c</code> ☆ <hr/>	<code>\fp_use:N <floating point variable></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, <pre> \fp_new:Nn \test \fp_set:Nn \test { 1.234 e 5 } \fp_use:N \test </pre> will insert 12345.00000 into the input stream. As illustrated, a floating point will always be inserted with ten significant digits given. Very large and very small values will include additional zeros for place value.
<hr/> <code>\fp_show:N</code> <code>\fp_show:c</code> <hr/>	<code>\fp_show:N <floating point variable></code> Displays the content of the <i><floating point variable></i> on the terminal.

167 Conversion of floating point values to other formats

It is useful to be able to convert floating point variables to other forms. These functions are expandable, so that the material can be used in a variety of contexts. The `\fp_use:N` function should also be consulted in this context, as it will insert the value of the floating point variable as a real number.

<hr/>	
<code>\fp_to_dim:N</code> ☆	<code>\fp_to_dim:N</code> <i><floating point variable></i>
<code>\fp_to_dim:c</code> ☆	Inserts the value of the <i><floating point variable></i> into the input stream converted into a dimension in points.
<hr/>	
<code>\fp_to_int:N</code> ☆	<code>\fp_to_int:N</code> <i><floating point variable></i>
<code>\fp_to_int:c</code> ☆	Inserts the integer value of the <i><floating point variable></i> into the input stream. The decimal part of the number will not be included, but will be used to round the integer.
<hr/>	
<code>\fp_to_tl:N</code> ☆	<code>\fp_to_tl:N</code> <i><floating point variable></i>
<code>\fp_to_tl:c</code> ☆	Inserts a representation of the <i><floating point variable></i> into the input stream as a token list. The representation follows the conventions of a pocket calculator:

Floating point value	Representation
1.234000000000e0	1.234
-1.234000000000e0	-1.234
1.234000000000e3	1234
1.234000000000e13	1234e13
1.234000000000e-1	0.1234
1.234000000000e-2	0.01234
1.234000000000e-3	1.234e-3

Notice that trailing zeros are removed in this process, and that numbers which do not require a decimal part do *not* include a decimal marker.

168 Rounding floating point values

The module can round floating point values to either decimal places or significant figures using the usual method in which exact halves are rounded up.

<hr/>	
<code>\fp_round_figures:Nn</code>	<code>\fp_round_figures:Nn</code> <i><floating point variable></i> <i>{<target>}</i>
<code>\fp_round_figures:cn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of significant figures (an integer expression). The rounding is carried out locally.
<hr/>	
<code>\fp_ground_figures:Nn</code>	<code>\fp_ground_figures:Nn</code> <i><floating point variable></i> <i>{<target>}</i>
<code>\fp_ground_figures:cn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of significant figures (an integer expression). The rounding is carried out globally.

<code>\fp_round_places:Nn</code>	<code>\fp_round_places:Nn <floating point variable> {<target>}</code>
<code>\fp_round_places:cn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of decimal places (an integer expression). The rounding is carried out locally.

<code>\fp_ground_places:Nn</code>	<code>\fp_ground_places:Nn <floating point variable> {<target>}</code>
<code>\fp_ground_places:cn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of decimal places (an integer expression). The rounding is carried out globally.

169 Floating-point conditionals

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

Tests if *<floating point>* 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 <fixed-point></code>
<code>\fp_if_zero:NTF</code> ★	<code>\fp_if_zero:NTF <fixed-point> {<true code>} {<false code>}</code>

Tests if *<floating point>* is equal to zero (*i.e.* equal to the special `\c_zero_fp` variable).

<code>\fp_compare:nNnTF</code>	<code>\fp_compare:nNnTF</code> <code>{<floating point₁>} <relation> {<floating point₂>}</code> <code>{<true code>} {<false code>}</code>
--------------------------------	--

This function compared the two *<floating point>* values, which may be stored as `fp` variables, using the *<relation>*:

Equal	=
Greater than	>
Less than	<

The tests treat undefined floating points as zero as the comparison is intended for real numbers only.

<u><code>\fp_compare:nTF</code></u>	<code>\fp_compare:nTF</code> <code>{ \langle floating point1 \rangle \langle relation \rangle \langle floating point2 \rangle }</code> <code>{\langle true code \rangle} {\langle false code \rangle}</code>
-------------------------------------	--

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.

170 Unary floating-point operations

The unary operations alter the value stored within an `fp` variable.

<u><code>\fp_abs:N</code></u> <u><code>\fp_abs:c</code></u>	<code>\fp_abs:N \langle floating point variable \rangle</code> Converts the $\langle floating point variable \rangle$ to its absolute value, assigning the result within the current TeX group.
--	--

<u><code>\fp_gabs:N</code></u> <u><code>\fp_gabs:c</code></u>	<code>\fp_gabs:N \langle floating point variable \rangle</code> Converts the $\langle floating point variable \rangle$ to its absolute value, assigning the result globally.
--	---

<u><code>\fp_neg:N</code></u> <u><code>\fp_neg:c</code></u>	<code>\fp_neg:N \langle floating point variable \rangle</code> Reverse the sign of the $\langle floating point variable \rangle$, assigning the result within the current TeX group.
--	--

<u><code>\fp_gneg:N</code></u> <u><code>\fp_gneg:c</code></u>	<code>\fp_gneg:N \langle floating point variable \rangle</code> Reverse the sign of the $\langle floating point variable \rangle$, assigning the result globally.
--	---

171 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).

<hr/> <u><code>\fp_add:Nn</code></u> <u><code>\fp_add:cn</code></u>	<code>\fp_add:Nn <floating point> {<value>}</code> Adds the <i><value></i> to the <i><floating point></i> , making the assignment within the current T _E X group level.
<hr/> <u><code>\fp_gadd:Nn</code></u> <u><code>\fp_gadd:cn</code></u>	<code>\fp_gadd:Nn <floating point> {<value>}</code> Adds the <i><value></i> to the <i><floating point></i> , making the assignment globally.
<hr/> <u><code>\fp_sub:Nn</code></u> <u><code>\fp_sub:cn</code></u>	<code>\fp_sub:Nn <floating point> {<value>}</code> Subtracts the <i><value></i> from the <i><floating point></i> , making the assignment within the current T _E X group level.
<hr/> <u><code>\fp_gsub:Nn</code></u> <u><code>\fp_gsub:cn</code></u>	<code>\fp_gsub:Nn <floating point> {<value>}</code> Subtracts the <i><value></i> from the <i><floating point></i> , making the assignment globally.
<hr/> <u><code>\fp_mul:Nn</code></u> <u><code>\fp_mul:cn</code></u>	<code>\fp_mul:Nn <floating point> {<value>}</code> Multiplies the <i><floating point></i> by the <i><value></i> , making the assignment within the current T _E X group level.
<hr/> <u><code>\fp_gmul:Nn</code></u> <u><code>\fp_gmul:cn</code></u>	<code>\fp_gmul:Nn <floating point> {<value>}</code> Multiplies the <i><floating point></i> by the <i><value></i> , making the assignment globally.
<hr/> <u><code>\fp_div:Nn</code></u> <u><code>\fp_div:cn</code></u>	<code>\fp_div:Nn <floating point> {<value>}</code> Divides the <i><floating point></i> by the <i><value></i> , making the assignment within the current T _E X group level. If the <i><value></i> is zero, the <i><floating point></i> will be set to <code>\c_undefined_fp</code> . The assignment is local.
<hr/> <u><code>\fp_gdiv:Nn</code></u> <u><code>\fp_gdiv:cn</code></u>	<code>\fp_gdiv:Nn <floating point> {<value>}</code> Divides the <i><floating point></i> by the <i><value></i> , making the assignment globally. If the <i><value></i> is zero, the <i><floating point></i> will be set to <code>\c_undefined_fp</code> . The assignment is global.

172 Floating-point power operations

<hr/> <u><code>\fp_pow:Nn</code></u> <u><code>\fp_pow:cn</code></u>	<code>\fp_pow:Nn <floating point> {<value>}</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 <code>\c_undefined_fp</code> . The assignment is local.
--	---

<code>\fp_gpow:Nn</code>	<code>\fp_gpow:Nn <floating point> {<value>}</code>
<code>\fp_gpow:cn</code>	

Raises the *<floating point>* to the given *<value>*. If the *<floating point>* is negative, then the *<value>* should be either a positive real number or a negative integer. If the *<floating point>* is positive, then the *<value>* may be any real value. Mathematically invalid operations such as 0^0 will give set the *<floating point>* to to `\c_undefined_fp`. The assignment is global.

173 Exponential and logarithm functions

<code>\fp_exp:Nn</code>	<code>\fp_exp:Nn <floating point> {<value>}</code>
<code>\fp_exp:cn</code>	

Calculates the exponential of the *<value>* and assigns this to the *<floating point>*. The assignment is local.

<code>\fp_gexp:Nn</code>	<code>\fp_gexp:Nn <floating point> {<value>}</code>
<code>\fp_gexp:cn</code>	

Calculates the exponential of the *<value>* and assigns this to the *<floating point>*. The assignment is global.

<code>\fp_ln:Nn</code>	<code>\fp_ln:Nn <floating point> {<value>}</code>
<code>\fp_ln:cn</code>	

Calculates the natural logarithm of the *<value>* and assigns this to the *<floating point>*. The assignment is local.

<code>\fp_gln:Nn</code>	<code>\fp_gln:Nn <floating point> {<value>}</code>
<code>\fp_gln:cn</code>	

Calculates the natural logarithm of the *<value>* and assigns this to the *<floating point>*. The assignment is global.

174 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.

<code>\fp_sin:Nn</code>	<code>\fp_sin:Nn <floating point> {<value>}</code>
<code>\fp_sin:cn</code>	

Assigns the sine of the *<value>* to the *<floating point>*. The *<value>* should be given in radians. The assignment is local.

<code>\fp_gsin:Nn</code>	<code>\fp_gsin:Nn <floating point> {<value>}</code>
<code>\fp_gsin:cn</code>	

Assigns the sine of the *<value>* to the *<floating point>*. The *<value>* should be given in radians. The assignment is global.

<code>\fp_cos:Nn</code>	<code>\fp_cos:Nn <floating point> {<value>}</code>
<code>\fp_cos:cn</code>	

Assigns the cosine of the *<value>* to the *<floating point>*. The *<value>* should be given in radians. The assignment is local.

<u><code>\fp_gcos:Nn</code></u>	<code>\fp_gcos:Nn <floating point> {<value>}</code>
<u><code>\fp_gcos:cn</code></u>	Assigns the cosine of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians. The assignment is global.
<u><code>\fp_tan:Nn</code></u>	<code>\fp_tan:Nn <floating point> {<value>}</code>
<u><code>\fp_tan:cn</code></u>	Assigns the tangent of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians. The assignment is local.
<u><code>\fp_gtan:Nn</code></u>	<code>\fp_gtan:Nn <floating point> {<value>}</code>
<u><code>\fp_gtan:cn</code></u>	Assigns the tangent of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians. The assignment is global.

175 Constant floating point values

<u><code>\c_e_fp</code></u>	The value of the base of natural numbers, e .
<u><code>\c_one_fp</code></u>	A floating point variable with permanent value 1: used for speeding up some comparisons.
<u><code>\c_pi_fp</code></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.

176 Notes on the floating point unit

As calculation of the elemental transcendental functions is computationally expensive compared to storage of results, after calculating a trigonometric function, exponent, *etc.* the module stored the result for reuse. Thus the performance of the module for repeated operations, most probably trigonometric functions, should be much higher than if the values were re-calculated every time they were needed.

Anyone with experience of programming floating point calculations will know that this is a complex area. The aim of the unit is to be accurate enough for the likely applications in a typesetting context. The arithmetic operations are therefore intended to provide ten digit accuracy with the last digit accurate to ± 1 . The elemental transcendental functions may not provide such high accuracy in every case, although the design aim has been to provide 10 digit accuracy for cases likely to be relevant in typesetting situations. A good overview of the challenges in this area can be found in J.-M. Muller,

Elementary functions: algorithms and implementation, 2nd edition, Birkhäuser Boston, New York, USA, 2006.

The internal representation of numbers is tuned to the needs of the underlying \TeX system. This means that the format is somewhat different from that used in, for example, computer floating point units. Programming in \TeX makes it most convenient to use a radix 10 system, using \TeX `count` registers for storage and taking advantage where possible of delimited arguments.

Part XXIII

The l3luatex package

LuaTeX-specific functions

177 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 Lua \TeX primitive `\latelua` named using the \LaTeX 3 scheme.

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

178 Category code tables

As well as providing methods to break out into Lua, there are places where additional \LaTeX 3 functions are provided by the Lua \TeX engine. In particular, Lua \TeX provides category code tables. These can be used to ensure that a set of category codes are in force in a more robust way than is possible with other engines. These are therefore used by `\ExplSyntaxOn` and `\ExplSyntaxOff` when using the Lua \TeX engine.

<hr/> <code>\cctab_new:N</code> <hr/>	<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>
<hr/> <code>\cctab_gset:Nn</code> <hr/>	<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>
<hr/> <code>\cctab_begin:N</code> <hr/>	<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>
<hr/> <code>\cctab_end</code> <hr/>	<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>
<hr/> <code>\c_code_cctab</code> <hr/>	<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> <hr/> <hr/>	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> <hr/> <hr/>	Category code table as set up by IniT _E X.
<hr/> <hr/> <code>\c_other_cctab</code> <hr/> <hr/>	Category code table where all characters have category code 12 (other).
<hr/> <hr/> <code>\c_string_cctab</code> <hr/> <hr/>	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

179 Bootstrap code

```
1 <*initex | package>
```

179.1 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>

```

179.2 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 `pdfetexmc` package is used to do this if necessary. At present, there is also a need to deal with some low-level allocation stuff that could usefully be added to `lualatex.ini`. As it is currently not, load Heiko Oberdiek's `luatex` package instead.

```

53 <*package>
54 \def\@tempa%
55 {%

```

```

56 \def\@tempa{}%
57 \RequirePackage{luatex}%
58 \RequirePackage{pdfsync}%
59 \let\pdfsync\pdfsync
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 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 ??.)

```

179.3 Dealing with package-mode meta-data

`\GetIdInfo` Functions for collecting up meta-data from the SVN information used by the L^AT_EX3 Project.

`\GetIdInfoFull`

`\GetIdInfoAuxI`

`\GetIdInfoAuxII`

`\GetIdInfoAuxIII`

`\GetIdInfoAuxCVS`

`\GetIdInfoAuxSVN`

```

97  <*package>
98  \protected\def\GetIdInfo
99  {
100    \begingroup
101    \catcode 32 = 10 \relax
102    \GetIdInfoAuxI
103  }
104  \protected\def\GetIdInfoAuxI$#1$#2%
105  {
106    \def\tempa{#1}%
107    \def\tempb{Id}%
108    \ifx\tempa\tempb
109      \def\tempa
110      {%
111        \endgroup
112        \def\ExplFileName{9999/99/99}%
113        \def\ExplFileDescription{#2}%
114        \def\ExplFileName{[unknown name]}%
115        \def\ExplFileVersion{999}%
116      }%
117    \else
118      \def\tempa
119      {%
120        \endgroup
121        \GetIdInfoAuxII$#1$#2}%
122      }%
123    \fi
124    \tempa
125  }
126  \protected\def\GetIdInfoAuxII$#1 #2.#3 #4 #5 #6 #7 #8$#9%
127  {%
128    \def\ExplFileName{#2}%
129    \def\ExplFileVersion{#4}%
130    \def\ExplFileDescription{#9}%
131    \GetIdInfoAuxIII#5\relax#3\relax#5\relax#6\relax
132  }
133  \protected\def\GetIdInfoAuxIII#1#2#3#4#5#6\relax
134  {%
135    \ifx#5/%
136      \expandafter\GetIdInfoAuxCVS
137    \else
138      \expandafter\GetIdInfoAuxSVN
139    \fi
140  }
141  \protected\def\GetIdInfoAuxCVS#1,v\relax#2\relax#3\relax
142  {\def\ExplFileName{#2}}

```

```

143 \protected\def\GetIdInfoAuxSVN#1\relax#2-#3-#4\relax#5Z\relax
144   {\def\ExplFileDate{#2/#3/#4}}
145 \</package>
      (End definition for \GetIdInfo. This function is documented on page ??.)

```

\ProvidesExplPackage For other packages and classes building on this one it is convenient not to need
 \ProvidesExplClass \ExplSyntaxOn each time.
 \ProvidesExplFile

```

146 <*package>
147 \protected\def\ProvidesExplPackage#1#2#3#4%
148   {%
149     \ProvidesPackage{#1}[#2 v#3 #4]%
150     \ExplSyntaxOn
151   }
152 \protected\def\ProvidesExplClass#1#2#3#4%
153   {%
154     \ProvidesClass{#1}[#2 v#3 #4]%
155     \ExplSyntaxOn
156   }
157 \protected\def\ProvidesExplFile#1#2#3#4%
158   {%
159     \ProvidesFile{#1}[#2 v#3 #4]%
160     \ExplSyntaxOn
161   }
162 \</package>

```

(End definition for \ProvidesExplPackage, \ProvidesExplClass, and \ProvidesExplFile. These functions are documented on page 6.)

\@pushfilename The idea here is to use L^AT_EX 2_ε's \@pushfilename and \@popfilename to track the
 \@popfilename current syntax status. This can be achieved by saving the current status flag at each
 push to a stack, then recovering it at the pop stage and checking if the code environment
 should still be active.

```

163 <*package>
164 \edef\@pushfilename
165   {%
166     \edef\expandafter\noexpand
167       \csname\detokenize{l_expl_status_stack_tl}\endcsname
168     {%
169       \noexpand\ifodd\expandafter\noexpand
170       \csname\detokenize{l_expl_status_bool}\endcsname
171       1%
172     \noexpand\else
173       0%
174     \noexpand\fi
175     \expandafter\noexpand
176     \csname\detokenize{l_expl_status_stack_tl}\endcsname
177   }%
178   \ExplSyntaxOff
179   \unexpanded\expandafter{\@pushfilename}%
180 }

```

```

181 \edef\@popfilename
182 {%
183   \unexpanded\expandafter{\@popfilename}%
184   \noexpand\if a\expandafter\noexpand\csname
185     \detokenize{l_expl_status_stack_tl}\endcsname a%
186     \ExplSyntaxOff
187   \noexpand\else
188     \noexpand\expandafter
189     \expandafter\noexpand\csname
190       \detokenize{expl_status_pop:w}\endcsname
191       \expandafter\noexpand\csname
192         \detokenize{l_expl_status_stack_tl}\endcsname
193       \noexpand\@nil
194   \noexpand\fi
195 }
196 \</package>

```

(End definition for \@pushfilename and \@popfilename. These functions are documented on page ??.)

`\l_expl_status_stack_tl` As expl3 itself cannot be loaded with the code environment already active, at the end of the package `\ExplSyntaxOff` can safely be called.

```

197 \<package>
198 \@namedef{\detokenize{l_expl_status_stack_tl}}{0}
199 \</package>

```

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

`\expl_status_pop:w` The pop auxiliary function removes the first item from the stack, saves the rest of the stack and then does the test. As `\ExplSyntaxOff` is already defined as a protected macro, there is no need for `\noexpand` here.

```

200 \<package>
201 \expandafter\edef\csname\detokenize{expl_status_pop:w}\endcsname#1#2\@nil
202 {%
203   \def\expandafter\noexpand
204     \csname\detokenize{l_expl_status_stack_tl}\endcsname{#2}%
205   \noexpand\ifodd#1\space
206     \noexpand\expandafter\noexpand\ExplSyntaxOn
207   \noexpand\else
208     \noexpand\expandafter\ExplSyntaxOff
209   \noexpand\fi
210 }
211 \</package>

```

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

We want the expl3 bundle to be loaded “as one”; this command is used to ensure that one of the 13 packages isn’t loaded on its own.

```

212 \<package>
213 \expandafter\protected\expandafter\def
214   \csname\detokenize{package_check_loaded_expl:}\endcsname
215   {%

```

```

216 \@ifpackageloaded{expl3}
217 {}
218 {%
219 \PackageError{expl3}
220 {Cannot load the expl3 modules separately}
221 {%
222 The expl3 modules cannot be loaded separately;\MessageBreak
223 please \string\usepackage\string{expl3\string} instead.
224 }%
225 }%
226 }
227 \</package>

```

179.4 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

```

179.5 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

```

179.6 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>

```

180 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	<code>\name_primitive:NN \delimiterfactor</code>	<code>\tex_delimiterfactor:D</code>
510	<code>\name_primitive:NN \nulldelimiterspace</code>	<code>\tex_nulldelimiterspace:D</code>
511	<code>\name_primitive:NN \everymath</code>	<code>\tex_everymath:D</code>
512	<code>\name_primitive:NN \mathsurround</code>	<code>\tex_mathsurround:D</code>
513	<code>\name_primitive:NN \medmuskip</code>	<code>\tex_medmuskip:D</code>
514	<code>\name_primitive:NN \thinmuskip</code>	<code>\tex_thinmuskip:D</code>
515	<code>\name_primitive:NN \thickmuskip</code>	<code>\tex_thickmuskip:D</code>
516	<code>\name_primitive:NN \scriptspace</code>	<code>\tex_scriptspace:D</code>
517	<code>\name_primitive:NN \noboundary</code>	<code>\tex_noboundary:D</code>
518	<code>\name_primitive:NN \accent</code>	<code>\tex_accent:D</code>
519	<code>\name_primitive:NN \char</code>	<code>\tex_char:D</code>
520	<code>\name_primitive:NN \discretionary</code>	<code>\tex_discretionary:D</code>
521	<code>\name_primitive:NN \hfil</code>	<code>\tex_hfil:D</code>
522	<code>\name_primitive:NN \hfilneg</code>	<code>\tex_hfilneg:D</code>
523	<code>\name_primitive:NN \hfill</code>	<code>\tex_hfill:D</code>
524	<code>\name_primitive:NN \hskip</code>	<code>\tex_hskip:D</code>
525	<code>\name_primitive:NN \hss</code>	<code>\tex_hss:D</code>
526	<code>\name_primitive:NN \vfil</code>	<code>\tex_vfil:D</code>
527	<code>\name_primitive:NN \vfilneg</code>	<code>\tex_vfilneg:D</code>
528	<code>\name_primitive:NN \vfill</code>	<code>\tex_vfill:D</code>
529	<code>\name_primitive:NN \vskip</code>	<code>\tex_vskip:D</code>
530	<code>\name_primitive:NN \vss</code>	<code>\tex_vss:D</code>
531	<code>\name_primitive:NN \unskip</code>	<code>\tex_unskip:D</code>
532	<code>\name_primitive:NN \kern</code>	<code>\tex_kern:D</code>
533	<code>\name_primitive:NN \unkern</code>	<code>\tex_unkern:D</code>
534	<code>\name_primitive:NN \hrule</code>	<code>\tex_hrule:D</code>
535	<code>\name_primitive:NN \vrule</code>	<code>\tex_vrule:D</code>
536	<code>\name_primitive:NN \leaders</code>	<code>\tex_leaders:D</code>
537	<code>\name_primitive:NN \cleaders</code>	<code>\tex_cleaders:D</code>
538	<code>\name_primitive:NN \xleaders</code>	<code>\tex_xleaders:D</code>
539	<code>\name_primitive:NN \lastkern</code>	<code>\tex_lastkern:D</code>
540	<code>\name_primitive:NN \lastskip</code>	<code>\tex_lastskip:D</code>
541	<code>\name_primitive:NN \indent</code>	<code>\tex_indent:D</code>
542	<code>\name_primitive:NN \par</code>	<code>\tex_par:D</code>
543	<code>\name_primitive:NN \noindent</code>	<code>\tex_noindent:D</code>
544	<code>\name_primitive:NN \vadjust</code>	<code>\tex_vadjust:D</code>
545	<code>\name_primitive:NN \baselineskip</code>	<code>\tex_baselineskip:D</code>
546	<code>\name_primitive:NN \lineskip</code>	<code>\tex_lineskip:D</code>
547	<code>\name_primitive:NN \lineskiplimit</code>	<code>\tex_lineskiplimit:D</code>
548	<code>\name_primitive:NN \clubpenalty</code>	<code>\tex_clubpenalty:D</code>
549	<code>\name_primitive:NN \widowpenalty</code>	<code>\tex_widowpenalty:D</code>
550	<code>\name_primitive:NN \exhyphenpenalty</code>	<code>\tex_exhyphenpenalty:D</code>
551	<code>\name_primitive:NN \hyphenpenalty</code>	<code>\tex_hyphenpenalty:D</code>
552	<code>\name_primitive:NN \linepenalty</code>	<code>\tex_linepenalty:D</code>
553	<code>\name_primitive:NN \doublehyphendemerits</code>	<code>\tex_doublehyphendemerits:D</code>
554	<code>\name_primitive:NN \finalhyphendemerits</code>	<code>\tex_finalhyphendemerits:D</code>
555	<code>\name_primitive:NN \adjdemerits</code>	<code>\tex_adjdemerits:D</code>
556	<code>\name_primitive:NN \hangafter</code>	<code>\tex_hangafter:D</code>
557	<code>\name_primitive:NN \hangindent</code>	<code>\tex_hangindent:D</code>
558	<code>\name_primitive:NN \parshape</code>	<code>\tex_parshape:D</code>

559	\name_primitive:NN \hsize	\tex_hsize:D
560	\name_primitive:NN \lefthyphenmin	\tex_lefthyphenmin:D
561	\name_primitive:NN \righthyphenmin	\tex_righthyphenmin:D
562	\name_primitive:NN \leftskip	\tex_leftskip:D
563	\name_primitive:NN \rightskip	\tex_rightskip:D
564	\name_primitive:NN \looseness	\tex_looseness:D
565	\name_primitive:NN \parskip	\tex_parskip:D
566	\name_primitive:NN \parindent	\tex_parindent:D
567	\name_primitive:NN \uchyph	\tex_uchyph:D
568	\name_primitive:NN \emergencystretch	\tex_emergencystretch:D
569	\name_primitive:NN \pretolerance	\tex_pretolerance:D
570	\name_primitive:NN \tolerance	\tex_tolerance:D
571	\name_primitive:NN \spaceskip	\tex_spaceskip:D
572	\name_primitive:NN \xspaceskip	\tex_xspaceskip:D
573	\name_primitive:NN \parfillskip	\tex_parfillskip:D
574	\name_primitive:NN \everypar	\tex_everypar:D
575	\name_primitive:NN \prevgraf	\tex_prevgraf:D
576	\name_primitive:NN \spacefactor	\tex_spacefactor:D
577	\name_primitive:NN \shipout	\tex_shipout:D
578	\name_primitive:NN \vsize	\tex_vsize:D
579	\name_primitive:NN \interlinepenalty	\tex_interlinepenalty:D
580	\name_primitive:NN \brokenpenalty	\tex_brokenpenalty:D
581	\name_primitive:NN \topskip	\tex_topskip:D
582	\name_primitive:NN \maxdeadcycles	\tex_maxdeadcycles:D
583	\name_primitive:NN \maxdepth	\tex_maxdepth:D
584	\name_primitive:NN \output	\tex_output:D
585	\name_primitive:NN \deadcycles	\tex_deadcycles:D
586	\name_primitive:NN \pagedepth	\tex_pagedepth:D
587	\name_primitive:NN \pagestretch	\tex_pagestretch:D
588	\name_primitive:NN \pagefilstretch	\tex_pagefilstretch:D
589	\name_primitive:NN \pagefillstretch	\tex_pagefillstretch:D
590	\name_primitive:NN \pagefillllstretch	\tex_pagefillllstretch:D
591	\name_primitive:NN \pageshrink	\tex_pageshrink:D
592	\name_primitive:NN \pagegoal	\tex_pagegoal:D
593	\name_primitive:NN \pagetotal	\tex_pagetotal:D
594	\name_primitive:NN \outputpenalty	\tex_outputpenalty:D
595	\name_primitive:NN \hoffset	\tex_hoffset:D
596	\name_primitive:NN \voffset	\tex_voffset:D
597	\name_primitive:NN \insert	\tex_insert:D
598	\name_primitive:NN \holdinginserts	\tex_holdinginserts:D
599	\name_primitive:NN \floatingpenalty	\tex_floatingpenalty:D
600	\name_primitive:NN \insertpenalties	\tex_insertpenalties:D
601	\name_primitive:NN \lower	\tex_lower:D
602	\name_primitive:NN \moveleft	\tex_moveleft:D
603	\name_primitive:NN \moveright	\tex_moveright:D
604	\name_primitive:NN \raise	\tex_raise:D
605	\name_primitive:NN \copy	\tex_copy:D
606	\name_primitive:NN \lastbox	\tex_lastbox:D
607	\name_primitive:NN \vsplit	\tex_vsplit:D
608	\name_primitive:NN \unhbox	\tex_unhbox:D

609	\name_primitive:NN \unhcopy	\tex_unhcopy:D
610	\name_primitive:NN \unvbox	\tex_unvbox:D
611	\name_primitive:NN \unvcopy	\tex_unvcopy:D
612	\name_primitive:NN \setbox	\tex_setbox:D
613	\name_primitive:NN \hbox	\tex_hbox:D
614	\name_primitive:NN \vbox	\tex_vbox:D
615	\name_primitive:NN \vtop	\tex_vtop:D
616	\name_primitive:NN \prevdepth	\tex_prevdepth:D
617	\name_primitive:NN \badness	\tex_badness:D
618	\name_primitive:NN \hbadness	\tex_hbadness:D
619	\name_primitive:NN \vbadness	\tex_vbadness:D
620	\name_primitive:NN \hfuzz	\tex_hfuzz:D
621	\name_primitive:NN \vfuzz	\tex_vfuzz:D
622	\name_primitive:NN \overfullrule	\tex_overfullrule:D
623	\name_primitive:NN \boxmaxdepth	\tex_boxmaxdepth:D
624	\name_primitive:NN \splitmaxdepth	\tex_splitmaxdepth:D
625	\name_primitive:NN \splittopskip	\tex_splittopskip:D
626	\name_primitive:NN \everyhbox	\tex_everyhbox:D
627	\name_primitive:NN \everyvbox	\tex_everyvbox:D
628	\name_primitive:NN \nullfont	\tex_nullfont:D
629	\name_primitive:NN \textfont	\tex_textfont:D
630	\name_primitive:NN \scriptfont	\tex_scriptfont:D
631	\name_primitive:NN \scriptscriptfont	\tex_scriptscriptfont:D
632	\name_primitive:NN \fontdimen	\tex_fontdimen:D
633	\name_primitive:NN \hyphenchar	\tex_hyphenchar:D
634	\name_primitive:NN \skewchar	\tex_skewchar:D
635	\name_primitive:NN \defaultthyphenchar	\tex_defaultthyphenchar:D
636	\name_primitive:NN \defaultskewchar	\tex_defaultskewchar:D
637	\name_primitive:NN \number	\tex_number:D
638	\name_primitive:NN \romannumeral	\tex_romannumeral:D
639	\name_primitive:NN \string	\tex_string:D
640	\name_primitive:NN \lowercase	\tex_lowercase:D
641	\name_primitive:NN \uppercase	\tex_uppercase:D
642	\name_primitive:NN \meaning	\tex_meaning:D
643	\name_primitive:NN \penalty	\tex_penalty:D
644	\name_primitive:NN \unpenalty	\tex_unpenalty:D
645	\name_primitive:NN \lastpenalty	\tex_lastpenalty:D
646	\name_primitive:NN \special	\tex_special:D
647	\name_primitive:NN \dump	\tex_dump:D
648	\name_primitive:NN \patterns	\tex_patterns:D
649	\name_primitive:NN \hyphenation	\tex_hyphenation:D
650	\name_primitive:NN \time	\tex_time:D
651	\name_primitive:NN \day	\tex_day:D
652	\name_primitive:NN \month	\tex_month:D
653	\name_primitive:NN \year	\tex_year:D
654	\name_primitive:NN \jobname	\tex_jobname:D
655	\name_primitive:NN \everyjob	\tex_everyjob:D
656	\name_primitive:NN \count	\tex_count:D
657	\name_primitive:NN \dimen	\tex_dimen:D
658	\name_primitive:NN \skip	\tex_skip:D

659	\name_primitive:NN	\toks	\tex_toks:D
660	\name_primitive:NN	\muskip	\tex_muskip:D
661	\name_primitive:NN	\box	\tex_box:D
662	\name_primitive:NN	\wd	\tex_wd:D
663	\name_primitive:NN	\ht	\tex_ht:D
664	\name_primitive:NN	\dp	\tex_dp:D
665	\name_primitive:NN	\catcode	\tex_catcode:D
666	\name_primitive:NN	\delcode	\tex_delcode:D
667	\name_primitive:NN	\sfcode	\tex_sfcode:D
668	\name_primitive:NN	\lccode	\tex_lccode:D
669	\name_primitive:NN	\uccode	\tex_uccode:D
670	\name_primitive:NN	\mathcode	\tex_mathcode:D

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

671	\name_primitive:NN	\ifdefined	\etex_ifdefined:D
672	\name_primitive:NN	\ifcsname	\etex_ifcsname:D
673	\name_primitive:NN	\unless	\etex_unless:D
674	\name_primitive:NN	\eTeXversion	\etex_eTeXversion:D
675	\name_primitive:NN	\eTeXrevision	\etex_eTeXrevision:D
676	\name_primitive:NN	\marks	\etex_marks:D
677	\name_primitive:NN	\topmarks	\etex_topmarks:D
678	\name_primitive:NN	\firstmarks	\etex_firstmarks:D
679	\name_primitive:NN	\botmarks	\etex_botmarks:D
680	\name_primitive:NN	\splitfirstmarks	\etex_splitfirstmarks:D
681	\name_primitive:NN	\splitbotmarks	\etex_splitbotmarks:D
682	\name_primitive:NN	\unexpanded	\etex_unexpanded:D
683	\name_primitive:NN	\detokenize	\etex_detokenize:D
684	\name_primitive:NN	\scantokens	\etex_scantokens:D
685	\name_primitive:NN	\showtokens	\etex_showtokens:D
686	\name_primitive:NN	\readline	\etex_readline:D
687	\name_primitive:NN	\tracingassigns	\etex_tracingassigns:D
688	\name_primitive:NN	\tracingscantokens	\etex_tracingscantokens:D
689	\name_primitive:NN	\tracingnesting	\etex_tracingnesting:D
690	\name_primitive:NN	\tracingifs	\etex_tracingifs:D
691	\name_primitive:NN	\currentiflevel	\etex_currentiflevel:D
692	\name_primitive:NN	\currentifbranch	\etex_currentifbranch:D
693	\name_primitive:NN	\currentifttype	\etex_currentifttype:D
694	\name_primitive:NN	\tracinggroups	\etex_tracinggroups:D
695	\name_primitive:NN	\currentgrouplevel	\etex_currentgrouplevel:D
696	\name_primitive:NN	\currentgrouptype	\etex_currentgrouptype:D
697	\name_primitive:NN	\showgroups	\etex_showgroups:D
698	\name_primitive:NN	\showifs	\etex_showifs:D
699	\name_primitive:NN	\interactionmode	\etex_interactionmode:D
700	\name_primitive:NN	\lastnodetype	\etex_lastnodetype:D
701	\name_primitive:NN	\iffontchar	\etex_iffontchar:D
702	\name_primitive:NN	\fontcharht	\etex_fontcharht:D
703	\name_primitive:NN	\fontchardp	\etex_fontchardp:D
704	\name_primitive:NN	\fontcharwd	\etex_fontcharwd:D
705	\name_primitive:NN	\fontcharic	\etex_fontcharic:D

706	\name_primitive:NN \parshapeindent	\etex_parshapeindent:D
707	\name_primitive:NN \parshapelength	\etex_parshapelength:D
708	\name_primitive:NN \parshapedimen	\etex_parshapedimen:D
709	\name_primitive:NN \numexpr	\etex_numexpr:D
710	\name_primitive:NN \dimexpr	\etex_dimexpr:D
711	\name_primitive:NN \glueexpr	\etex_glueexpr:D
712	\name_primitive:NN \muexpr	\etex_muexpr:D
713	\name_primitive:NN \gluestretch	\etex_gluestretch:D
714	\name_primitive:NN \glueshrink	\etex_glueshrink:D
715	\name_primitive:NN \gluestretchorder	\etex_gluestretchorder:D
716	\name_primitive:NN \glueshrinkorder	\etex_glueshrinkorder:D
717	\name_primitive:NN \gluetomu	\etex_gluetomu:D
718	\name_primitive:NN \mutoglu	\etex_mutoglu:D
719	\name_primitive:NN \lastlinefit	\etex_lastlinefit:D
720	\name_primitive:NN \interlinepenalties	\etex_interlinepenalties:D
721	\name_primitive:NN \clubpenalties	\etex_clubpenalties:D
722	\name_primitive:NN \widowpenalties	\etex_widowpenalties:D
723	\name_primitive:NN \displaywidowpenalties	\etex_displaywidowpenalties:D
724	\name_primitive:NN \middle	\etex_middle:D
725	\name_primitive:NN \savinghyphcodes	\etex_savinghyphcodes:D
726	\name_primitive:NN \savingvdiscards	\etex_savingvdiscards:D
727	\name_primitive:NN \pagediscards	\etex_pagediscards:D
728	\name_primitive:NN \splitdiscards	\etex_splitdiscards:D
729	\name_primitive:NN \TeXstate	\etex_TeXstate:D
730	\name_primitive:NN \beginL	\etex_beginL:D
731	\name_primitive:NN \endL	\etex_endL:D
732	\name_primitive:NN \beginR	\etex_beginR:D
733	\name_primitive:NN \endR	\etex_endR:D
734	\name_primitive:NN \predisplaydirection	\etex_predisplaydirection:D
735	\name_primitive:NN \everyeof	\etex_everyeof:D
736	\name_primitive:NN \protected	\etex_protected:D

The newer primitives are more complex: there are an awful lot of them, and we don't use them all at the moment. So the following is selective. In the case of the pdfTeX primitives, we retain pdf at the start of the names *only* for directly PDF-related primitives, as there are a lot of pdfTeX primitives that start \pdf... but are not related to PDF output. These ones related to PDF output.

737	\name_primitive:NN \pdfcreationdate	\pdfTEX_pdfcreationdate:D
738	\name_primitive:NN \pdfcolorstack	\pdfTEX_pdfcolorstack:D
739	\name_primitive:NN \pdfcompresslevel	\pdfTEX_pdfcompresslevel:D
740	\name_primitive:NN \pdfdecimaldigits	\pdfTEX_pdfdecimaldigits:D
741	\name_primitive:NN \pdfhorigin	\pdfTEX_pdfhorigin:D
742	\name_primitive:NN \pdfinfo	\pdfTEX_pdfinfo:D
743	\name_primitive:NN \pdflastxform	\pdfTEX_pdflastxform:D
744	\name_primitive:NN \pdfliteral	\pdfTEX_pdfliteral:D
745	\name_primitive:NN \pdfminorversion	\pdfTEX_pdfminorversion:D
746	\name_primitive:NN \pdfobjcompresslevel	\pdfTEX_pdfobjcompresslevel:D
747	\name_primitive:NN \pdfoutput	\pdfTEX_pdfoutput:D
748	\name_primitive:NN \pdfrefxform	\pdfTEX_pdfrefxform:D
749	\name_primitive:NN \pdfrestore	\pdfTEX_pdfrestore:D

```

750 \name_primitive:NN \pdfsave \pdfTeX_pdfsave:D
751 \name_primitive:NN \pdfsetmatrix \pdfTeX_pdfsetmatrix:D
752 \name_primitive:NN \pdfpkresolution \pdfTeX_pdfpkresolution:D
753 \name_primitive:NN \pdfTeXrevision \pdfTeX_pdfTeXrevision:D
754 \name_primitive:NN \pdfvorigin \pdfTeX_pdfvorigin:D
755 \name_primitive:NN \pdfxform \pdfTeX_pdfxform:D

```

While these are not.

```

756 \name_primitive:NN \pdfstrcmp \pdfTeX_strcmp:D

```

X_YTeX-specific primitives. Note that X_YTeX’s \strcmp is handled earlier and is “rolled up” into \pdfstrcmp.

```

757 \name_primitive:NN \XeTeXversion \xetex_XeTeXversion:D

```

Primitives from LuaTeX.

```

758 \name_primitive:NN \catcodetable \luaTeX_catcodetable:D
759 \name_primitive:NN \directlua \luaTeX_directlua:D
760 \name_primitive:NN \initcatcodetable \luaTeX_initcatcodetable:D
761 \name_primitive:NN \lattelua \luaTeX_lattelua:D
762 \name_primitive:NN \luaTeXversion \luaTeX_luaTeXversion:D
763 \name_primitive:NN \savecatcodetable \luaTeX_savecatcodetable:D

```

The job is done: close the group (using the primitive renamed!).

```

764 \tex_endgroup:D

```

L^AT_EX 2_ε will have moved a few primitives, so these are sorted out.

```

765 <*package>
766 \tex_let:D \tex_end:D @@end
767 \tex_let:D \tex_everydisplay:D \frozen@everydisplay
768 \tex_let:D \tex_everymath:D \frozen@everymath
769 \tex_let:D \tex_hyphen:D @@hyph
770 \tex_let:D \tex_input:D @@input
771 \tex_let:D \tex_italic_correction:D @@italiccorr
772 \tex_let:D \tex_underline:D @@underline

```

That is also true for the luatex package for L^AT_EX 2_ε.

```

773 \tex_let:D \luaTeX_catcodetable:D \luaTeXcatcodetable
774 \tex_let:D \luaTeX_initcatcodetable:D \luaTeXinitcatcodetable
775 \tex_let:D \luaTeX_lattelua:D \luaTeXlattelua
776 \tex_let:D \luaTeX_savecatcodetable:D \luaTeXsavecatcodetable
777 </package>
778 </initex | package>

```

181 l3basics implementation

```

779 <*initex | package>
780 <*package>
781 \ProvidesExplPackage
782   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
783 \package_check_loaded_expl:

```

784 `\package`

181.1 Renaming some T_EX primitives (again)

Having given all the T_EX primitives a consistent name, we need to give sensible names to the ones we actually want to use. These will be defined as needed in the appropriate modules, but do a few now, just to get started.²

```

\if_true: Then some conditionals.
\if_false: 785 \tex_let:D \if_true:      \tex_iftrue:D
      \or: 786 \tex_let:D \if_false:    \tex_iffalse:D
      \else: 787 \tex_let:D \or:      \tex_or:D
      \fi: 788 \tex_let:D \else:      \tex_else:D
\reverse_if:N 789 \tex_let:D \fi:      \tex_fi:D
      \if:w 790 \tex_let:D \reverse_if:N \etex_unless:D
\if_charcode:w 791 \tex_let:D \if:w      \tex_if:D
\if_catcode:w 792 \tex_let:D \if_charcode:w \tex_if:D
\if_meaning:w 793 \tex_let:D \if_catcode:w \tex_ifcat:D
      794 \tex_let:D \if_meaning:w \tex_ifx:D

```

(End definition for \if_true:. This function is documented on page 22.)

```

\if_mode_math: TEX lets us detect some if its modes.
\if_mode_horizontal: 795 \tex_let:D \if_mode_math:      \tex_ifmmode:D
\if_mode_vertical: 796 \tex_let:D \if_mode_horizontal: \tex_ifhmode:D
\if_mode_inner: 797 \tex_let:D \if_mode_vertical:      \tex_ifvmode:D
      798 \tex_let:D \if_mode_inner:      \tex_ifinner:D

```

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

```

\if_cs_exist:N
\if_cs_exist:w 799 \tex_let:D \if_cs_exist:N      \etex_ifdefined:D
      800 \tex_let:D \if_cs_exist:w      \etex_ifcurname:D

```

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

`\exp_after:wN` The three `\exp_` functions are used in the `l3expan` module where they are described.

```

\exp_not:N 801 \tex_let:D \exp_after:wN      \tex_expandafter:D
\exp_not:n 802 \tex_let:D \exp_not:N      \tex_noexpand:D
      803 \tex_let:D \exp_not:n      \etex_unexpanded:D

```

(End definition for \exp_after:wN. This function is documented on page 29.)

```

\token_to_meaning:N
\token_to_str:N 804 \tex_let:D \token_to_meaning:N \tex_meaning:D
      \cs:w 805 \tex_let:D \token_to_str:N      \tex_string:D
      \cs_end: 806 \tex_let:D \cs:w      \tex_csname:D
\cs_meaning:N 807 \tex_let:D \cs_end:      \tex_endcsname:D
\cs_show:N 808 \tex_let:D \cs_meaning:N      \tex_meaning:D
      809 \tex_let:D \cs_show:N      \tex_show:D

```

²This renaming gets expensive in terms of csname usage, an alternative scheme would be to just use the `\tex...:D` name in the cases where no good alternative exists.

(End definition for `\token_to_meaning:N`. This function is documented on page 15.)

`\scan_stop:` The next three are basic functions for which there also exist versions that are safe inside alignments. These safe versions are defined in the `l3prg` module.

```
\group_begin:
\group_end:
810 \tex_let:D \scan_stop:      \tex_relax:D
811 \tex_let:D \group_begin:    \tex_begingroup:D
812 \tex_let:D \group_end:      \tex_endgroup:D
```

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

```
\if_int_compare:w
\int_to_roman:w
```

```
813 \tex_let:D \if_int_compare:w \tex_ifnum:D
814 \tex_let:D \int_to_roman:w   \tex_romannumeral:D
```

(End definition for `\if_int_compare:w`. This function is documented on page 68.)

```
\group_insert_after:N
```

```
815 \tex_let:D \group_insert_after:N \tex_aftergroup:D
```

(End definition for `\group_insert_after:N`. This function is documented on page 9.)

```
\tex_global:D
\tex_long:D
\tex_protected:D
```

```
816 \tex_let:D \tex_global:D      \tex_global:D
817 \tex_let:D \tex_long:D        \tex_long:D
818 \tex_let:D \tex_protected:D    \etex_protected:D
```

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

`\exp_args:Nc` Discussed in `l3expan`, but needed much earlier.

```
819 \tex_long:D \tex_def:D \exp_args:Nc #1#2 { \exp_after:wN #1 \cs:w #2 \cs_end: }
```

(End definition for `\exp_args:Nc`. This function is documented on page 26.)

`\token_to_str:c` A small number of variants by hand.

```
\cs_meaning:c
\cs_show:c
820 \tex_def:D \cs_meaning:c { \exp_args:Nc \cs_meaning:N }
821 \tex_def:D \token_to_str:c { \exp_args:Nc \token_to_str:N }
822 \tex_def:D \cs_show:c     { \exp_args:Nc \cs_show:N }
```

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

181.2 Defining functions

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

`\cs_set_nopar:Npn` All assignment functions in L^AT_EX3 should be naturally robust; after all, the T_EX 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
823 \tex_let:D \cs_set_nopar:Npn \tex_def:D
824 \tex_let:D \cs_set_nopar:Npx \tex_edef:D
```

```
\cs_set_protected_nopar:Npn
825 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npn
```

```
\cs_set_protected_nopar:Npx
826 { \tex_long:D \cs_set_nopar:Npn }
```

```
\cs_set_protected:Npn
827 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npx
```

```
\cs_set_protected:Npx
828 { \tex_long:D \cs_set_nopar:Npx }
```

```
829 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npn
```

```
830 { \tex_protected:D \cs_set_nopar:Npn }
```

```

831 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npx
832 { \tex_protected:D \cs_set_nopar:Npx }
833 \cs_set_protected_nopar:Npn \cs_set_protected:Npn
834 { \tex_protected:D \tex_long:D \cs_set_nopar:Npn }
835 \cs_set_protected_nopar:Npn \cs_set_protected:Npx
836 { \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 837 \tex_let:D \cs_gset_nopar:Npn          \tex_gdef:D
\cs_gset:Npn        838 \tex_let:D \cs_gset_nopar:Npx          \tex_xdef:D
\cs_gset:Npx        839 \cs_set_protected_nopar:Npn \cs_gset:Npn
\cs_gset_protected_nopar:Npn 840 { \tex_long:D \cs_gset_nopar:Npn }
\cs_gset_protected_nopar:Npx 841 \cs_set_protected_nopar:Npn \cs_gset:Npx
\cs_gset_protected:Npn 842 { \tex_long:D \cs_gset_nopar:Npx }
\cs_gset_protected:Npx 843 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npn
844 { \tex_protected:D \cs_gset_nopar:Npn }
845 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npx
846 { \tex_protected:D \cs_gset_nopar:Npx }
847 \cs_set_protected_nopar:Npn \cs_gset_protected:Npn
848 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npn }
849 \cs_set_protected_nopar:Npn \cs_gset_protected:Npx
850 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npx }
(End definition for \cs_gset_nopar:Npn. This function is documented on page ??.)

```

181.3 Selecting tokens

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

```

851 \cs_set:Npn \use:c #1 { \cs:w #1 \cs_end: }
(End definition for \use:c. This function is documented on page 16.)

```

\use:x Fully expands its argument and passes it to the input stream. Uses \cs_tmp: as a scratch register but does not affect it.

```

852 \cs_set_protected:Npn \use:x #1
853 {
854   \group_begin:
855   \cs_set:Npx \cs_tmp:w {#1}
856   \exp_after:wN
857   \group_end:
858   \cs_tmp:w
859 }
860 \cs_set:Npn \cs_tmp:w { }

```

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

```

\use:nnn 861 \cs_set:Npn \use:n #1 {#1}
\use:nnnn 862 \cs_set:Npn \use:nn #1#2 {#1#2}
863 \cs_set:Npn \use:nnn #1#2#3 {#1#2#3}
864 \cs_set:Npn \use:nnnn #1#2#3#4 {#1#2#3#4}

```

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

```
\use_ii:nn      865 \cs_set:Npn \use_i:nn  #1#2 {#1}
                  866 \cs_set:Npn \use_ii:nn #1#2 {#2}
```

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

```
\use_ii:nnnn      867 \cs_set:Npn \use_i:nnnn  #1#2#3 {#1}
\use_iii:nnnn      868 \cs_set:Npn \use_ii:nnnn #1#2#3 {#2}
\use_i_ii:nnnn      869 \cs_set:Npn \use_iii:nnnn #1#2#3 {#3}
\use_i:nnnnn      870 \cs_set:Npn \use_i_ii:nnnn #1#2#3 {#1#2}
\use_ii:nnnnn      871 \cs_set:Npn \use_i:nnnnn  #1#2#3#4 {#1}
\use_iii:nnnnn      872 \cs_set:Npn \use_ii:nnnnn #1#2#3#4 {#2}
\use_iv:nnnnn      873 \cs_set:Npn \use_iii:nnnnn #1#2#3#4 {#3}
                  874 \cs_set:Npn \use_iv:nnnnn  #1#2#3#4 {#4}
```

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

```
\use_none_delimit_by_q_stop:w      875 \cs_set:Npn \use_none_delimit_by_q_nil:w #1 \q_nil { }
\use_none_delimit_by_q_recursion_stop:w 876 \cs_set:Npn \use_none_delimit_by_q_stop:w #1 \q_stop { }
                  877 \cs_set:Npn \use_none_delimit_by_q_recursion_stop:w #1 \q_recursion_stop { }
```

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

```
\use_i_delimit_by_q_stop:nw      878 \cs_set:Npn \use_i_delimit_by_q_nil:nw #1#2 \q_nil {#1}
\use_i_delimit_by_q_recursion_stop:nw 879 \cs_set:Npn \use_i_delimit_by_q_stop:nw #1#2 \q_stop {#1}
                  880 \cs_set:Npn \use_i_delimit_by_q_recursion_stop:nw #1#2 \q_recursion_stop {#1}
```

181.4 Gobbling tokens from input

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

```
\use_none:nnnnnnn      881 \cs_set:Npn \use_none:n          #1          { }
\use_none:nnnnnnnn      882 \cs_set:Npn \use_none:nn        #1#2        { }
\use_none:nnnnnnnnn      883 \cs_set:Npn \use_none:nnn       #1#2#3       { }
\use_none:nnnnnnnnnn      884 \cs_set:Npn \use_none:nnnn      #1#2#3#4      { }
\use_none:nnnnnnnnnn      885 \cs_set:Npn \use_none:nnnnn     #1#2#3#4#5     { }
\use_none:nnnnnnnnnn      886 \cs_set:Npn \use_none:nnnnnn    #1#2#3#4#5#6    { }
\use_none:nnnnnnnnnn      887 \cs_set:Npn \use_none:nnnnnnn   #1#2#3#4#5#6#7   { }
\use_none:nnnnnnnnnn      888 \cs_set:Npn \use_none:nnnnnnnn  #1#2#3#4#5#6#7#8   { }
\use_none:nnnnnnnnnn      889 \cs_set:Npn \use_none:nnnnnnnnn #1#2#3#4#5#6#7#8#9 { }
```

181.5 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 T_EX 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 T_EX 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 T_EX 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 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.

```
890 \cs_set_nopar:Npn \prg_return_true:
891 { \exp_after:wN \use_i:nn \int_to_roman:w }
892 \cs_set_nopar:Npn \prg_return_false:
893 { \exp_after:wN \use_ii:nn \int_to_roman:w}
```

An extended state space could be implemented by including a more elaborate function in place of `\use_i:nn`/`\use_ii:nn`. Provided two arguments are absorbed then the code will work.

`\prg_set_conditional:Npnn` The user functions for the types using parameter text from the programmer. Call aux function to grab parameters, split the base function into name and signature and then use, e.g., `\cs_set:Npn` to define it with.

```
\prg_new_conditional:Npnn
\prg_set_protected_conditional:Npnn
\prg_new_protected_conditional:Npnn
894 \cs_set_protected:Npn \prg_set_conditional:Npnn #1
895 {
896   \prg_get_parm_aux:nw
897   {
898     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
899     \cs_set:Npn { parm }
900   }
901 }
902 \cs_set_protected:Npn \prg_new_conditional:Npnn #1
903 {
904   \prg_get_parm_aux:nw
905   {
906     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
907     \cs_new:Npn { parm }
908   }
909 }
```

```

910 \cs_set_protected:Npn \prg_set_protected_conditional:Npnn #1
911 {
912   \prg_get_parm_aux:nw{
913     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
914     \cs_set_protected:Npn { parm }
915   }
916 }
917 \cs_set_protected:Npn \prg_new_protected_conditional:Npnn #1
918 {
919   \prg_get_parm_aux:nw
920   {
921     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
922     \cs_new_protected:Npn { parm }
923   }
924 }

```

\prg_set_conditional:Nnn The user functions for the types automatically inserting the correct parameter text based
 \prg_new_conditional:Nnn on the signature. Call aux function after calculating number of arguments, split the base
 \prg_set_protected_conditional:Nnn function into name and signature and then use, *e.g.*, \cs_set:Npn to define it with.
 \prg_new_protected_conditional:Nnn

```

925 \cs_set_protected:Npn \prg_set_conditional:Nnn #1
926 {
927   \exp_args:Nnf \prg_get_count_aux:nn
928   {
929     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
930     \cs_set:Npn { count }
931   }
932   { \cs_get_arg_count_from_signature:N #1 }
933 }
934 \cs_set_protected:Npn \prg_new_conditional:Nnn #1
935 {
936   \exp_args:Nnf \prg_get_count_aux:nn
937   {
938     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
939     \cs_new:Npn { count }
940   }
941   { \cs_get_arg_count_from_signature:N #1 }
942 }
943
944 \cs_set_protected:Npn \prg_set_protected_conditional:Nnn #1{
945   \exp_args:Nnf \prg_get_count_aux:nn{
946     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
947     \cs_set_protected:Npn {count}
948   }{\cs_get_arg_count_from_signature:N #1}
949 }
950
951 \cs_set_protected:Npn \prg_new_protected_conditional:Nnn #1
952 {
953   \exp_args:Nnf \prg_get_count_aux:nn
954   {

```



```

955         \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
956         \cs_new_protected:Npn {count}
957     }
958     { \cs_get_arg_count_from_signature:N #1 }
959 }

```

\prg_set_eq_conditional:NNn The obvious setting-equal functions.

\prg_new_eq_conditional:NNn

```

960 \cs_set_protected:Npn \prg_set_eq_conditional:NNn #1#2#3
961 { \prg_set_eq_conditional_aux:NNNn \cs_set_eq:cc #1#2 {#3} }
962 \cs_set_protected:Npn \prg_new_eq_conditional:NNn #1#2#3
963 { \prg_set_eq_conditional_aux:NNNn \cs_new_eq:cc #1#2 {#3} }

```

\prg_get_parm_aux:nw

\prg_get_count_aux:nn

For the Npnn type we must grab the parameter text before continuing. We make this a very generic function that takes one argument before reading everything up to a left brace. Something similar for the Nnn type.

```

964 \cs_set:Npn \prg_get_count_aux:nn #1#2 { #1 {#2} }
965 \cs_set:Npn \prg_get_parm_aux:nw #1#2# { #1 {#2} }

```

\prg_generate_conditional_parm_aux:nnNNnnnn

\prg_generate_conditional_parm_aux:nw

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.

```

966 \cs_set_protected:Npn \prg_generate_conditional_aux:nnNNnnnn #1#2#3#4#5#6#7#8
967 {
968     \prg_generate_conditional_aux:nnw {#5}
969     {
970         #4 {#1} {#2} {#6} {#8}
971     }
972     #7 , ? , \q_recursion_stop
973 }

```

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.

```

974 \cs_set_protected:Npn \prg_generate_conditional_aux:nnw #1#2#3 ,
975 {
976     \if:w ?#3
977     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
978     \fi:
979     \use:c { prg_generate_#3_form_#1:Nnnnn } #2
980     \prg_generate_conditional_aux:nnw {#1} {#2}
981 }

```

`\prg_generate_p_form_parm:Nnnnn` How to generate the various forms. The `parm` types here takes the following arguments:
`\prg_generate_TF_form_parm:Nnnnn` 1: how to define (an N-type), 2: name, 3: signature, 4: parameter text (or empty), 5:
`\prg_generate_T_form_parm:Nnnnn` replacement. Remember that the logic-returning functions expect two arguments to be
`\prg_generate_F_form_parm:Nnnnn` present after `\c_zero`: notice the construction of the different variants relies on this, and
that the TF variant will be slightly faster than the T version.

```

982 \cs_set_protected:Npn \prg_generate_p_form_parm:Nnnnn #1#2#3#4#5
983 {
984   \exp_args:Nc #1 { #2 _p: #3 } #4
985   {
986     #5 \c_zero
987     \c_true_bool \c_false_bool
988   }
989 }
990 \cs_set_protected:Npn \prg_generate_T_form_parm:Nnnnn #1#2#3#4#5
991 {
992   \exp_args:Nc #1 { #2 : #3 T } #4
993   {
994     #5 \c_zero
995     \use:n \use_none:n
996   }
997 }
998 \cs_set_protected:Npn \prg_generate_F_form_parm:Nnnnn #1#2#3#4#5
999 {
1000   \exp_args:Nc #1 { #2 : #3 F } #4
1001   {
1002     #5 \c_zero
1003     { }
1004   }
1005 }
1006 \cs_set_protected:Npn \prg_generate_TF_form_parm:Nnnnn #1#2#3#4#5
1007 {
1008   \exp_args:Nc #1 { #2 : #3 TF } #4
1009   { #5 \c_zero }
1010 }

```

`\prg_generate_p_form_count:Nnnnn` The `count` form is similar, but of course requires a number rather than a primitive
`\prg_generate_TF_form_count:Nnnnn` argument specification.

```

1011 \cs_set_protected:Npn \prg_generate_p_form_count:Nnnnn #1#2#3#4#5
1012 {
1013   \cs_generate_from_arg_count:cNnn { #2 _p: #3 } #1 {#4}
1014   {
1015     #5 \c_zero
1016     \c_true_bool \c_false_bool
1017   }
1018 }
1019 \cs_set_protected:Npn \prg_generate_T_form_count:Nnnnn #1#2#3#4#5
1020 {
1021   \cs_generate_from_arg_count:cNnn { #2 : #3 T } #1 {#4}
1022   {

```

```

1023         #5 \c_zero
1024         \use:n \use_none:n
1025     }
1026 }
1027 \cs_set_protected:Npn \prg_generate_F_form_count:Nnnnn #1#2#3#4#5
1028 {
1029     \cs_generate_from_arg_count:cNnn { #2 : #3 F } #1 {#4}
1030     {
1031         #5 \c_zero
1032         { }
1033     }
1034 }
1035 \cs_set_protected:Npn \prg_generate_TF_form_count:Nnnnn #1#2#3#4#5
1036 {
1037     \cs_generate_from_arg_count:cNnn { #2 : #3 TF } #1 {#4}
1038     { #5 \c_zero }
1039 }

```

\prg_set_eq_conditional_aux:NNNn

\prg_set_eq_conditional_aux:NNNw

```

1040 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNn #1#2#3#4
1041 { \prg_set_eq_conditional_aux:NNNw #1#2#3#4 , ? , \q_recursion_stop }

```

Manual clist loop over argument #4.

```

1042 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNw #1#2#3#4 ,
1043 {
1044     \if:w ? #4 \scan_stop:
1045     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1046     \fi:
1047     #1
1048     { \exp_args:NNc \cs_split_function:NN #2 { prg_conditional_form_#4:nnn } }
1049     { \exp_args:NNc \cs_split_function:NN #3 { prg_conditional_form_#4:nnn } }
1050     \prg_set_eq_conditional_aux:NNNw #1 {#2} {#3}
1051 }
1052 \cs_set:Npn \prg_conditional_form_p:nnn #1#2#3 { #1 _p : #2 }
1053 \cs_set:Npn \prg_conditional_form_TF:nnn #1#2#3 { #1 : #2 TF }
1054 \cs_set:Npn \prg_conditional_form_T:nnn #1#2#3 { #1 : #2 T }
1055 \cs_set:Npn \prg_conditional_form_F:nnn #1#2#3 { #1 : #2 F }

```

All that is left is to define the canonical boolean true and false. I think Michael originated the idea of expandable boolean tests. At first these were supposed to expand into either TT or TF to be tested using \if:w but this was later changed to 00 and 01, so they could be used in logical operations. Later again they were changed to being numerical constants with values of 1 for true and 0 for false. We need this from the get-go.

\c_true_bool Here are the canonical boolean values.

\c_false_bool

```

1056 \tex_chardef:D \c_true_bool = 1~
1057 \tex_chardef:D \c_false_bool = 0~

```

181.6 Dissecting a control sequence

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

- The usual case of a printable escape character;
- the case of a non-printable escape character, e.g., when the value of `\tex_escapechar:D` is negative;
- when the escape character is a space.

One approach to solve this is to test how many tokens result from `\token_to_str:N \a`. If there are two tokens, then the escape character is printable, while if it is non-printable then only one is present.

However, there is an additional complication: the control sequence itself may start with a space. Clearly that should *not* be lost in the process of converting to a string. So the approach adopted is a little more intricate still. When the escape character is printable, `\token_to_str:N _` yields the escape character itself and a space. The escape sequence will terminate the expansion started by `\int_to_roman:w`, which is a negative number and so will not gobble the escape character even if it's a number. The `\if:w` test will then be `false`, and the naïve approach of gobbling the first character of the `\token_to_str:N` version of the control sequence will work, even if the first character is a space. The second case is that the escape character is itself a space. In this case, the escape character space is consumed terminating the first `\int_to_roman:w`, and `\cs_to_str_aux:w` is expanded. This inserts a space, making the `\if:w` test `true`. The second `\int_to_roman:w` will then execute the `\token_to_str:N`, with the escape-character space being consumed by the `\int_to_roman:w`, and thus leaving the control sequence name in the input stream. The final case is where the escape character is not printable. The flow here starts with the `\token_to_str:N _` giving just a space, which terminates the first `\int_to_roman:w` but leaves no token for the `\if:w` test. This means that the `\int_to_roman:w` is executed before the test is finished. The result is that the `\fi:`, expanded before the `\if:w` is finished, becomes `\scan_stop: \fi:`, and the `\scan_stop:` is then used in the `\if:w` test. In this case, `\token_to_str:N` is therefore used with no gobbling at all, which is exactly what is needed in this case.

```

1058 \cs_set_nopar:Npn \cs_to_str:N
1059 {
1060   \if:w \int_to_roman:w - '0 \token_to_str:N \ %
1061     \cs_to_str_aux:w
1062   \fi:
1063   \exp_after:wN \use_none:n \token_to_str:N
1064 }
1065 \cs_set_nopar:Npn \cs_to_str_aux:w #1 \use_none:n
1066 { ~ \int_to_roman:w - '0 \fi: }
```

`\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 $\langle false \rangle$ is returned with $\langle true \rangle$ for when there is a colon in the function and $\langle false \rangle$ 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`.

```

1067 \group_begin:
1068   \tex_lccode:D '\@ = '\: \scan_stop:
1069   \tex_catcode:D '\@ = 12~
1070   \tex_lowercase:D
1071   {
1072     \group_end:

```

First ensure that we actually get a properly evaluated str as we don't know how many expansions `\cs_to_str:N` requires. Insert extra colon to catch the error cases.

```

1073   \cs_set:Npn \cs_split_function:NN #1#2
1074   {
1075     \exp_after:wN \cs_split_function_aux:w
1076     \int_to_roman:w - '\q \cs_to_str:N #1 @ a \q_stop #2
1077   }

```

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.

```

1078   \cs_set:Npn \cs_split_function_aux:w #1 @ #2#3 \q_stop #4
1079   {
1080     \if_meaning:w a #2
1081       \exp_after:wN \use_i:nn
1082     \else:
1083       \exp_after:wN \use_ii:nn
1084     \fi:
1085     { #4 {#1} { } \c_false_bool }
1086     { \cs_split_function_auxii:w #2#3 \q_stop #4 {#1} }
1087   }
1088   \cs_set:Npn \cs_split_function_auxii:w #1 @a \q_stop #2#3
1089   { #2{#3}{#1}\c_true_bool }

```

End of lowercase

```

1090   }

```

`\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.

```

1091 \cs_set:Npn \cs_get_function_name:N #1
1092 { \cs_split_function:NN #1 \use_i:nnn }
1093 \cs_set:Npn \cs_get_function_signature:N #1
1094 { \cs_split_function:NN #1 \use_ii:nnn }

```

181.7 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:`.

```

1095 \prg_set_conditional:Npnn \cs_if_exist:N #1 { p , T , F , TF }
1096 {
1097   \if_meaning:w #1 \scan_stop:
1098     \prg_return_false:
1099   \else:
1100     \if_cs_exist:N #1
1101       \prg_return_true:
1102     \else:
1103       \prg_return_false:
1104     \fi:
1105   \fi:
1106 }

```

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.

```

1107 \prg_set_conditional:Npnn \cs_if_exist:c #1 { p , T , F , TF }
1108 {
1109   \if_cs_exist:w #1 \cs_end:
1110     \exp_after:wN \use_i:nn
1111   \else:
1112     \exp_after:wN \use_ii:nn
1113   \fi:
1114   {
1115     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1116     \prg_return_false:
1117   \else:
1118     \prg_return_true:
1119   \fi:
1120 }
1121 \prg_return_false:
1122 }

```

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

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

```

\cs_if_free:c 1123 \prg_set_conditional:Npnn \cs_if_free:N #1 { p , T , F , TF }
1124 {
1125   \if_meaning:w #1 \scan_stop:

```

```

1126     \prg_return_true:
1127 \else:
1128     \if_cs_exist:N #1
1129     \prg_return_false:
1130 \else:
1131     \prg_return_true:
1132 \fi:
1133 \fi:
1134 }
1135 \prg_set_conditional:Npnn \cs_if_free:c #1 { p , T , F , TF }
1136 {
1137     \if_cs_exist:w #1 \cs_end:
1138     \exp_after:wN \use_i:nn
1139 \else:
1140     \exp_after:wN \use_ii:nn
1141 \fi:
1142 {
1143     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1144     \prg_return_true:
1145 \else:
1146     \prg_return_false:
1147 \fi:
1148 }
1149 { \prg_return_true: }
1150 }

```

??.) (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.

```

1151 \cs_set:Npn \cs_if_exist_use:NTF #1#2
1152 { \cs_if_exist:NTF #1 { #1 #2 } }
1153 \cs_set:Npn \cs_if_exist_use:NF #1
1154 { \cs_if_exist:NTF #1 { #1 } }
1155 \cs_set:Npn \cs_if_exist_use:NT #1 #2
1156 { \cs_if_exist:NTF #1 { #1#2 } { } }
1157 \cs_set:Npn \cs_if_exist_use:N #1
1158 { \cs_if_exist:NTF #1 { #1 } { } }
1159 \cs_set:Npn \cs_if_exist_use:cTF #1#2
1160 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } }
1161 \cs_set:Npn \cs_if_exist_use:cF #1
1162 { \cs_if_exist:cTF {#1} { \use:c {#1} } }
1163 \cs_set:Npn \cs_if_exist_use:cT #1#2
1164 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } { } }
1165 \cs_set:Npn \cs_if_exist_use:c #1
1166 { \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

181.8 Defining and checking (new) functions

`\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`
`\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`.

```

1167 \*package>
1168 \tex_let:D \c_minus_one \m@ne
1169 \*package>
1170 \*initex>
1171 \tex_countdef:D \c_minus_one = 10 ~
1172 \c_minus_one = -1 ~
1173 \*initex>
1174 \tex_chardef:D \c_sixteen = 16~
1175 \tex_chardef:D \c_zero = 0~
1176 \tex_chardef:D \c_six = 6~
1177 \tex_chardef:D \c_seven = 7~
1178 \tex_chardef:D \c_twelve = 12~

```

(End definition for `\c_minus_one`, `\c_zero`, and `\c_sixteen`. These functions are documented on page 67.)

`\c_max_register_int` This is here as this particular integer is needed both in package mode and to bootstrap `l3alloc`

```

1179 \tex_mathchardef:D \c_max_register_int = 32 767 \scan_stop:

```

(End definition for `\c_max_register_int`. This function is documented on page 67.)

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`.

```

1180 \cs_set_protected_nopar:Npn \iow_log:x
1181 { \tex_immediate:D \tex_write:D \c_minus_one }
1182 \cs_set_protected_nopar:Npn \iow_term:x
1183 { \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 \LaTeX 3 has loaded `l3msg` then the code should issue a
`\msg_kernel_error:nxx` usable if terse error message and halt. This can only happen if a coding error is made by
`\msg_kernel_error:nn` the team, so this is a reasonable response.

```

1184 \cs_set_protected_nopar:Npn \msg_kernel_error:nxxx #1#2#3#4
1185 {

```



```

1186 \tex_errmessage:D
1187 {
1188     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!~! ^^J
1189     Argh,~internal~LaTeX3~error! ^^J ^^J
1190     Module ~ #1 , ~ message-name-~"#2": ^^J
1191     Arguments~'#3'~and~'#4' ^^J ^^J
1192     This~is~one~for~The~LaTeX3~Project::~bailing~out
1193 }
1194 \tex_end:D
1195 }
1196 \cs_set_protected_nopar:Npn \msg_kernel_error:nxx #1#2#3
1197 { \msg_kernel_error:nxxx {#1} {#2} {#3} { } }
1198 \cs_set_protected_nopar:Npn \msg_kernel_error:nn #1#2
1199 { \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.

```

1200 \cs_set_nopar:Npn \msg_line_context:
1201 { on~line~\tex_the:D \tex_inputlineno:D }

```

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

\chk_if_free_cs:N This command is called by \cs_new_nopar:Npn and \cs_new_eq:NN etc. to make sure
\chk_if_free_cs:c that the argument sequence is not already in use. If it is, an error is signalled. It checks
if $\langle csname \rangle$ 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!

```

1202 \cs_set_protected_nopar:Npn \chk_if_free_cs:N #1
1203 {
1204     \cs_if_free:NF #1
1205     {
1206         \msg_kernel_error:nxxx { kernel } { command-already-defined }
1207         { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1208     }
1209 }
1210 \<package>
1211 \tex_ifodd:D \l@expl@log@functions@bool
1212 \cs_set_protected_nopar:Npn \chk_if_free_cs:N #1
1213 {
1214     \cs_if_free:NF #1
1215     {
1216         \msg_kernel_error:nxxx { kernel } { command-already-defined }
1217         { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1218     }
1219     \iow_log:x { Defining~\token_to_str:N #1~ \msg_line_context: }
1220 }
1221 \fi:
1222 \</package>
1223 \cs_set_protected_nopar:Npn \chk_if_free_cs:c
1224 { \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.

```

1225 \cs_set_protected_nopar:Npn \chk_if_exist_cs:N #1
1226 {
1227   \cs_if_exist:NF #1
1228   {
1229     \msg_kernel_error:nnxx { kernel } { command-not-defined }
1230     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1231   }
1232 }
1233 \cs_set_protected_nopar:Npn \chk_if_exist_cs:c
1234 { \exp_args:Nc \chk_if_exist_cs:N }

```

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

181.9 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
1235 \cs_set:Npn \cs_tmp:w #1#2
1236 {
1237   \cs_set_protected_nopar:Npn #1 ##1
1238   {
1239     \chk_if_free_cs:N ##1
1240     #2 ##1
1241   }
1242 }
1243 \cs_tmp:w \cs_new_nopar:Npn \cs_gset_nopar:Npn
1244 \cs_tmp:w \cs_new_nopar:Npx \cs_gset_nopar:Npx
1245 \cs_tmp:w \cs_new:Npn \cs_gset:Npn
1246 \cs_tmp:w \cs_new:Npx \cs_gset:Npx
1247 \cs_tmp:w \cs_new_protected_nopar:Npn \cs_gset_protected_nopar:Npn
1248 \cs_tmp:w \cs_new_protected_nopar:Npx \cs_gset_protected_nopar:Npx
1249 \cs_tmp:w \cs_new_protected:Npn \cs_gset_protected:Npn
1250 \cs_tmp:w \cs_new_protected:Npx \cs_gset_protected:Npx

```

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

`\cs_set_nopar:cpn` Like `\cs_set_nopar:Npn` and `\cs_new_nopar:Npn`, except that the first argument consists of the sequence of characters that should be used to form the name of the desired control sequence (the `c` stands for csname argument, see the expansion module). Global versions are also provided.

`\cs_set_nopar:cpn` `\cs_set_nopar:cpn⟨string⟩⟨rep-text⟩` will turn `⟨string⟩` into a csname and then assign `⟨rep-text⟩` to it by using `\cs_set_nopar:Npn`. This means that there might be a parameter string between the two arguments.

```

1251 \cs_set:Npn \cs_tmp:w #1#2
1252 { \cs_new_protected_nopar:Npn #1 { \exp_args:Nc #2 } }

```

```

1253 \cs_tmp:w \cs_set_nopar:cpn \cs_set_nopar:Npn
1254 \cs_tmp:w \cs_set_nopar:cpx \cs_set_nopar:Npx
1255 \cs_tmp:w \cs_gset_nopar:cpn \cs_gset_nopar:Npn
1256 \cs_tmp:w \cs_gset_nopar:cpx \cs_gset_nopar:Npx
1257 \cs_tmp:w \cs_new_nopar:cpn \cs_new_nopar:Npn
1258 \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 1259 \cs_tmp:w \cs_set:cpn \cs_set:Npn
\cs_gset:cpx 1260 \cs_tmp:w \cs_set:cpx \cs_set:Npx
\cs_new:cpn 1261 \cs_tmp:w \cs_gset:cpn \cs_gset:Npn
\cs_new:cpx 1262 \cs_tmp:w \cs_gset:cpx \cs_gset:Npx
1263 \cs_tmp:w \cs_new:cpn \cs_new:Npn
1264 \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.

```

1265 \cs_tmp:w \cs_set_protected_nopar:cpn \cs_set_protected_nopar:Npn
1266 \cs_tmp:w \cs_set_protected_nopar:cpx \cs_set_protected_nopar:Npx
1267 \cs_tmp:w \cs_gset_protected_nopar:cpn \cs_gset_protected_nopar:Npn
1268 \cs_tmp:w \cs_gset_protected_nopar:cpx \cs_gset_protected_nopar:Npx
1269 \cs_tmp:w \cs_new_protected_nopar:cpn \cs_new_protected_nopar:Npn
1270 \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 1271 \cs_tmp:w \cs_set_protected:cpn \cs_set_protected:Npn
\cs_gset_protected:cpx 1272 \cs_tmp:w \cs_set_protected:cpx \cs_set_protected:Npx
\cs_new_protected:cpn 1273 \cs_tmp:w \cs_gset_protected:cpn \cs_gset_protected:Npn
\cs_new_protected:cpx 1274 \cs_tmp:w \cs_gset_protected:cpx \cs_gset_protected:Npx
1275 \cs_tmp:w \cs_new_protected:cpn \cs_new_protected:Npn
1276 \cs_tmp:w \cs_new_protected:cpx \cs_new_protected:Npx

```

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

181.10 Copying definitions

\cs_set_eq:NN These macros allow us to copy the definition of a control sequence to another control sequence.

\cs_set_eq:cN The = sign allows us to define funny char tokens like = itself or \sqcup with this function.
 \cs_set_eq:Nc 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”.

```

1277 \cs_new_protected:Npn \cs_set_eq:NN #1 { \tex_let:D #1 =~ }
1278 \cs_new_protected_nopar:Npn \cs_set_eq:cN { \exp_args:Nc \cs_set_eq:NN }
1279 \cs_new_protected_nopar:Npn \cs_set_eq:Nc { \exp_args:NNc \cs_set_eq:NN }
1280 \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 1281 \cs_new_protected:Npn \cs_new_eq:NN #1
\cs_new_eq:Nc 1282 {
\cs_new_eq:cc 1283   \chk_if_free_cs:N #1
1284   \tex_global:D \cs_set_eq:NN #1
1285 }
1286 \cs_new_protected_nopar:Npn \cs_new_eq:cN { \exp_args:Nc \cs_new_eq:NN }
1287 \cs_new_protected_nopar:Npn \cs_new_eq:Nc { \exp_args:NNc \cs_new_eq:NN }
1288 \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 1289 \cs_new_protected_nopar:Npn \cs_gset_eq:NN { \tex_global:D \cs_set_eq:NN }
\cs_gset_eq:Nc 1290 \cs_new_protected_nopar:Npn \cs_gset_eq:Nc { \exp_args:NNc \cs_gset_eq:NN }
\cs_gset_eq:cc 1291 \cs_new_protected_nopar:Npn \cs_gset_eq:cN { \exp_args:Nc \cs_gset_eq:NN }
1292 \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 ??.)

```

181.11 undefining functions

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.

```

1293 \cs_new_protected_nopar:Npn \cs_undefine:N #1
1294 { \cs_gset_eq:NN #1 \c_undefined:D }
1295 \cs_new_protected_nopar:Npn \cs_undefine:c #1
1296 {
1297   \if_cs_exist:w #1 \cs_end:
1298     \exp_after:wN \use:n
1299   \else:
1300     \exp_after:wN \use_none:n
1301   \fi:
1302   { \cs_gset_eq:cN {#1} \c_undefined:D }
1303 }
      (End definition for \cs_undefine:N and \cs_undefine:c. These functions are documented on
page ??.)

```

181.12 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.

```

1304 \cs_new:Npn \cs_get_arg_count_from_signature:N #1
1305 { \cs_split_function:NN #1 \cs_get_arg_count_from_signature_aux:nnN }
1306 \cs_new:Npn \cs_get_arg_count_from_signature_aux:nnN #1#2#3
1307 {
1308   \if_meaning:w \c_true_bool #3
1309   \exp_after:wN \use_i:nn
1310   \else:
1311   \exp_after:wN\use_ii:nn
1312   \fi:
1313   {
1314     \exp_after:wN \cs_get_arg_count_from_signature_auxii:w
1315     \use_none:nnnnnnnnn #2 9876543210 \q_stop
1316   }
1317   { -1 }
1318 }
1319 \cs_new:Npn \cs_get_arg_count_from_signature_auxii:w #1#2 \q_stop {#1}

```

A variant form we need right away.

```

1320 \cs_new_nopar:Npn \cs_get_arg_count_from_signature:c
1321 { \exp_args:Nc \cs_get_arg_count_from_signature:N }

```

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

\cs_generate_from_arg_count:NNnn
\cs_generate_from_arg_count_error_msg:Nn
\cs_generate_from_arg_count_aux:nwn

We provide a constructor function for defining functions with a given number of arguments. For this we need to choose the correct parameter text and then use that when defining. Since TeX supports from zero to nine arguments, we use a simple switch to choose the correct parameter text, ensuring the result is returned after finishing the conditional. If it is not between zero and nine, we throw an error.

1: function to define, 2: with what to define it, 3: the number of args it requires and 4: the replacement text

```

1322 \cs_new_protected:Npn \cs_generate_from_arg_count:NNnn #1#2#3#4
1323 {
1324   \if_case:w \int_eval:w #3 \int_eval_end:
1325   \cs_generate_from_arg_count_aux:nwn {}
1326   \or: \cs_generate_from_arg_count_aux:nwn {##1}
1327   \or: \cs_generate_from_arg_count_aux:nwn {##1##2}
1328   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3}
1329   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4}
1330   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5}
1331   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6}

```

```

1332 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7}
1333 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8}
1334 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8##9}
1335 \else:
1336   \cs_generate_from_arg_count_error_msg:Nn #1 {#3}
1337   \use_i:nnn
1338 \fi:
1339 {#2#1}
1340 {#4}
1341 }
1342 \cs_new_protected_nopar:Npn
1343 \cs_generate_from_arg_count_aux:nwn #1 #2 \fi: #3
1344 { \fi: #3 #1 }

```

A variant form we need right away.

```

1345 \cs_new_nopar:Npn \cs_generate_from_arg_count:cNnn
1346 { \exp_args:Nc \cs_generate_from_arg_count:NNnn }

```

The error message. Elsewhere we use the value of -1 to signal a missing colon in a function, so provide a hint for help on this.

```

1347 \cs_new:Npn \cs_generate_from_arg_count_error_msg:Nn #1#2
1348 {
1349   \msg_kernel_error:nnxx { kernel } { bad-number-of-arguments }
1350   { \token_to_str:N #1 } { \int_eval:n {#2} }
1351 }

```

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

181.13 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.

```

\cs_set:Nn We want to define \cs_set:Nn as
\cs_set:Nx
\cs_set_nopar:Nn \cs_set_protected:Npn \cs_set:Nn #1#2
\cs_set_nopar:Nx {
\cs_set_protected:Nn \cs_generate_from_arg_count:NNnn #1 \cs_set:Npn
\cs_set_protected:Nx { \cs_get_arg_count_from_signature:N #1 } {#2}
\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

```

```

1352 \cs_set:Npn \cs_tmp:w #1#2#3
1353 {
1354   \cs_set_protected:cpx { cs_ #1 : #2 } ##1##2
1355   {
1356     \exp_not:N \cs_generate_from_arg_count:NNnn ##1
1357     \exp_after:wN \exp_not:N \cs:w cs_#1 : #3 \cs_end:

```

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.

```

1358         { \exp_not:N\cs_get_arg_count_from_signature:N ##1 }{##2}
1359     }
1360 }

```

Then we define the 32 variants beginning with N.

```

1361 \cs_tmp:w { set } { Nn } { Npn }
1362 \cs_tmp:w { set } { Nx } { Npx }
1363 \cs_tmp:w { set_nopar } { Nn } { Npn }
1364 \cs_tmp:w { set_nopar } { Nx } { Npx }
1365 \cs_tmp:w { set_protected } { Nn } { Npn }
1366 \cs_tmp:w { set_protected } { Nx } { Npx }
1367 \cs_tmp:w { set_protected_nopar } { Nn } { Npn }
1368 \cs_tmp:w { set_protected_nopar } { Nx } { Npx }
1369 \cs_tmp:w { gset } { Nn } { Npn }
1370 \cs_tmp:w { gset } { Nx } { Npx }
1371 \cs_tmp:w { gset_nopar } { Nn } { Npn }
1372 \cs_tmp:w { gset_nopar } { Nx } { Npx }
1373 \cs_tmp:w { gset_protected } { Nn } { Npn }
1374 \cs_tmp:w { gset_protected } { Nx } { Npx }
1375 \cs_tmp:w { gset_protected_nopar } { Nn } { Npn }
1376 \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
1377 \cs_tmp:w { new } { Nn } { Npn }
1378 \cs_tmp:w { new } { Nx } { Npx }
1379 \cs_tmp:w { new_nopar } { Nn } { Npn }
1380 \cs_tmp:w { new_nopar } { Nx } { Npx }
1381 \cs_tmp:w { new_protected } { Nn } { Npn }
1382 \cs_tmp:w { new_protected } { Nx } { Npx }
1383 \cs_tmp:w { new_protected_nopar } { Nn } { Npn }
1384 \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}
}

1385 \cs_set:Npn \cs_tmp:w #1#2#3
1386 {
1387   \cs_set_protected:cpx {cs_#1:#2}##1##2{
1388     \exp_not:N\cs_generate_from_arg_count:cNnn {##1}
1389     \exp_after:wN \exp_not:N \cs:w cs_#1:#3 \cs_end:
1390     { \exp_not:N \cs_get_arg_count_from_signature:c {##1} } {##2}
1391   }
1392 }

```

\cs_set:cn The 32 c variants.

\cs_set:cx	1393	\cs_tmp:w { set }	{ cn } { Npn }
\cs_set_nopar:cn	1394	\cs_tmp:w { set }	{ cx } { Npx }
\cs_set_nopar:cx	1395	\cs_tmp:w { set_nopar }	{ cn } { Npn }
\cs_set_protected:cn	1396	\cs_tmp:w { set_nopar }	{ cx } { Npx }
\cs_set_protected:cx	1397	\cs_tmp:w { set_protected }	{ cn } { Npn }
\cs_set_protected_nopar:cn	1398	\cs_tmp:w { set_protected }	{ cx } { Npx }
\cs_set_protected_nopar:cx	1399	\cs_tmp:w { set_protected_nopar }	{ cn } { Npn }
\cs_gset:cn	1400	\cs_tmp:w { set_protected_nopar }	{ cx } { Npx }
\cs_gset:cx	1401	\cs_tmp:w { gset }	{ cn } { Npn }
\cs_gset_nopar:cn	1402	\cs_tmp:w { gset }	{ cx } { Npx }
\cs_gset_nopar:cx	1403	\cs_tmp:w { gset_nopar }	{ cn } { Npn }
\cs_gset_protected:cn	1404	\cs_tmp:w { gset_nopar }	{ cx } { Npx }
\cs_gset_protected:cx	1405	\cs_tmp:w { gset_protected }	{ cn } { Npn }
\cs_gset_protected_nopar:cn	1406	\cs_tmp:w { gset_protected }	{ cx } { Npx }
\cs_gset_protected_nopar:cx	1407	\cs_tmp:w { gset_protected_nopar }	{ cn } { Npn }
	1408	\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	1409	\cs_tmp:w { new }	{ cn } { Npn }
\cs_new_nopar:cn	1410	\cs_tmp:w { new }	{ cx } { Npx }
\cs_new_nopar:cx	1411	\cs_tmp:w { new_nopar }	{ cn } { Npn }
\cs_new_protected:cn	1412	\cs_tmp:w { new_nopar }	{ cx } { Npx }
\cs_new_protected:cx	1413	\cs_tmp:w { new_protected }	{ cn } { Npn }
\cs_new_protected_nopar:cn	1414	\cs_tmp:w { new_protected }	{ cx } { Npx }
\cs_new_protected_nopar:cx	1415	\cs_tmp:w { new_protected_nopar }	{ cn } { Npn }
	1416	\cs_tmp:w { new_protected_nopar }	{ cx } { Npx }

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

181.14 Checking control sequence equality

\cs_if_eq:NN Check if two control sequences are identical.

\cs_if_eq:cN	1417	\prg_new_conditional:Npnn \cs_if_eq:NN #1#2 { p , T , F , TF }
\cs_if_eq:Nc	1418	{
\cs_if_eq:cc	1419	\if_meaning:w #1#2
	1420	\prg_return_true: \else: \prg_return_false: \fi:
	1421	}
	1422	\cs_new_nopar:Npn \cs_if_eq_p:cN { \exp_args:Nc \cs_if_eq_p:NN }
	1423	\cs_new_nopar:Npn \cs_if_eq:cNTF { \exp_args:Nc \cs_if_eq:NNTF }
	1424	\cs_new_nopar:Npn \cs_if_eq:cNT { \exp_args:Nc \cs_if_eq:NNT }
	1425	\cs_new_nopar:Npn \cs_if_eq:cNF { \exp_args:Nc \cs_if_eq:NNF }
	1426	\cs_new_nopar:Npn \cs_if_eq_p:Nc { \exp_args:NNc \cs_if_eq_p:NN }
	1427	\cs_new_nopar:Npn \cs_if_eq:NcTF { \exp_args:NNc \cs_if_eq:NNTF }
	1428	\cs_new_nopar:Npn \cs_if_eq:NcT { \exp_args:NNc \cs_if_eq:NNT }
	1429	\cs_new_nopar:Npn \cs_if_eq:NcF { \exp_args:NNc \cs_if_eq:NNF }
	1430	\cs_new_nopar:Npn \cs_if_eq_p:cc { \exp_args:Ncc \cs_if_eq_p:NN }
	1431	\cs_new_nopar:Npn \cs_if_eq:ccTF { \exp_args:Ncc \cs_if_eq:NNTF }
	1432	\cs_new_nopar:Npn \cs_if_eq:ccT { \exp_args:Ncc \cs_if_eq:NNT }


```

1433 \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 ??.)

```

181.15 Diagnostic wrapper functions

```

\kernel_register_show:N
\kernel_register_show:c
1434 \cs_new_nopar:Npn \kernel_register_show:N #1
1435 {
1436   \cs_if_exist:NTF #1
1437   { \tex_showthe:D #1 }
1438   {
1439     \msg_kernel_error:nnx { kernel } { variable-not-defined }
1440     { \token_to_str:N #1 }
1441   }
1442 }
1443 \cs_new_nopar:Npn \kernel_register_show:c { \exp_args:Nc \int_show:N }
      (End definition for \kernel_register_show:N and \kernel_register_show:c. These functions are
      documented on page ??.)

```

181.16 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.

```

\pdfTeX_if_engine:
1444 \cs_new_eq:NN \luatex_if_engine:T \use_none:n
1445 \cs_new_eq:NN \luatex_if_engine:F \use:n
1446 \cs_new_eq:NN \luatex_if_engine:TF \use_ii:nn
1447 \cs_new_eq:NN \pdfTeX_if_engine:T \use:n
1448 \cs_new_eq:NN \pdfTeX_if_engine:F \use_none:n
1449 \cs_new_eq:NN \pdfTeX_if_engine:TF \use_i:nn
1450 \cs_new_eq:NN \xetex_if_engine:T \use_none:n
1451 \cs_new_eq:NN \xetex_if_engine:F \use:n
1452 \cs_new_eq:NN \xetex_if_engine:TF \use_ii:nn
1453 \cs_new_eq:NN \luatex_if_engine_p: \c_false_bool
1454 \cs_new_eq:NN \pdfTeX_if_engine_p: \c_true_bool
1455 \cs_new_eq:NN \xetex_if_engine_p: \c_false_bool
1456 \cs_if_exist:NT \xetex_XeTeXversion:D
1457 {
1458   \cs_set_eq:NN \pdfTeX_if_engine:T \use_none:n
1459   \cs_set_eq:NN \pdfTeX_if_engine:F \use:n
1460   \cs_set_eq:NN \pdfTeX_if_engine:TF \use_ii:nn
1461   \cs_set_eq:NN \xetex_if_engine:T \use:n
1462   \cs_set_eq:NN \xetex_if_engine:F \use_none:n
1463   \cs_set_eq:NN \xetex_if_engine:TF \use_i:nn
1464   \cs_set_eq:NN \pdfTeX_if_engine_p: \c_false_bool
1465   \cs_set_eq:NN \xetex_if_engine_p: \c_true_bool
1466 }
1467 \cs_if_exist:NT \luatex_directlua:D
1468 {
1469   \cs_set_eq:NN \luatex_if_engine:T \use:n

```

```

1470 \cs_set_eq:NN \luatex_if_engine:F \use_none:n
1471 \cs_set_eq:NN \luatex_if_engine:TF \use_i:nn
1472 \cs_set_eq:NN \pdfTeX_if_engine:T \use_none:n
1473 \cs_set_eq:NN \pdfTeX_if_engine:F \use:n
1474 \cs_set_eq:NN \pdfTeX_if_engine:TF \use_ii:nn
1475 \cs_set_eq:NN \luatex_if_engine_p: \c_true_bool
1476 \cs_set_eq:NN \pdfTeX_if_engine_p: \c_false_bool
1477 }

```

(End definition for `\xetex_if_engine:`, `\luatex_if_engine:`, and `\pdfTeX_if_engine:`. These functions are documented on page ??.)

181.17 Doing nothing functions

`\prg_do_nothing:` This does not fit anywhere else!

```

1478 \cs_new_nopar:Npn \prg_do_nothing: { }

```

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

181.18 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.

```

1479 \prg_new_conditional:Npnn \str_if_eq:nn #1#2 { p , T , F , TF }
1480 {
1481   \if_int_compare:w \pdfTeX_strcmp:D { \exp_not:n {#1} } { \exp_not:n {#2} }
1482   = \c_zero
1483   \prg_return_true: \else: \prg_return_false: \fi:
1484 }
1485 \prg_new_conditional:Npnn \str_if_eq:xx #1#2 { p , T , F , TF }
1486 {
1487   \if_int_compare:w \pdfTeX_strcmp:D {#1} {#2} = \c_zero
1488   \prg_return_true: \else: \prg_return_false: \fi:
1489 }

```

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

181.19 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

```

1490 (*deprecated)
1491 \cs_new_eq:NN \cs_gnew_nopar:Npn \cs_new_nopar:Npn
1492 \cs_new_eq:NN \cs_gnew:Npn \cs_new:Npn
1493 \cs_new_eq:NN \cs_gnew_protected_nopar:Npn \cs_new_protected_nopar:Npn
1494 \cs_new_eq:NN \cs_gnew_protected:Npn \cs_new_protected:Npn
1495 \cs_new_eq:NN \cs_gnew_nopar:Npx \cs_new_nopar:Npx
1496 \cs_new_eq:NN \cs_gnew:Npx \cs_new:Npx
1497 \cs_new_eq:NN \cs_gnew_protected_nopar:Npx \cs_new_protected_nopar:Npx
1498 \cs_new_eq:NN \cs_gnew_protected:Npx \cs_new_protected:Npx
1499 \cs_new_eq:NN \cs_gnew_nopar:cpn \cs_new_nopar:cpn

```

```

1500 \cs_new_eq:NN \cs_gnew:cpn \cs_new:cpn
1501 \cs_new_eq:NN \cs_gnew_protected_nopar:cpn \cs_new_protected_nopar:cpn
1502 \cs_new_eq:NN \cs_gnew_protected:cpn \cs_new_protected:cpn
1503 \cs_new_eq:NN \cs_gnew_nopar:cpx \cs_new_nopar:cpx
1504 \cs_new_eq:NN \cs_gnew:cpx \cs_new:cpx
1505 \cs_new_eq:NN \cs_gnew_protected_nopar:cpx \cs_new_protected_nopar:cpx
1506 \cs_new_eq:NN \cs_gnew_protected:cpx \cs_new_protected:cpx
1507 </deprecated>

1508 <*deprecated>
1509 \cs_new_eq:NN \cs_gnew_eq:NN \cs_new_eq:NN
1510 \cs_new_eq:NN \cs_gnew_eq:cN \cs_new_eq:cN
1511 \cs_new_eq:NN \cs_gnew_eq:Nc \cs_new_eq:Nc
1512 \cs_new_eq:NN \cs_gnew_eq:cc \cs_new_eq:cc
1513 </deprecated>

1514 <*deprecated>
1515 \cs_new_eq:NN \cs_gundefine:N \cs_undefine:N
1516 \cs_new_eq:NN \cs_gundefine:c \cs_undefine:c
1517 </deprecated>

1518 <*deprecated>
1519 \cs_new_eq:NN \group_execute_after:N \group_insert_after:N
1520 </deprecated>

```

Deprecated 2011-09-06, for removal by 2012-09-05.

```

\c_pdftex_is_engine_bool
\c_luatex_is_engine_bool
\c_xetex_is_engine_bool

```

Predicates are better

```

1521 \cs_new_eq:NN \c_luatex_is_engine_bool \luatex_if_engine_p:
1522 \cs_new_eq:NN \c_pdftex_is_engine_bool \pdftex_if_engine_p:
1523 \cs_new_eq:NN \c_xetex_is_engine_bool \xetex_if_engine_p:

```

(End definition for \c_pdftex_is_engine_bool, \c_luatex_is_engine_bool, and \c_xetex_is_engine_bool. These functions are documented on page ??.)

Deprecated 2011-09-06, for removal by 2012-10-06.

```

\use_i_after_fi:nw
\use_i_after_else:nw
\use_i_after_or:nw
\use_i_after_orelse:nw

```

These functions return the first argument after ending the conditional. This is rather specialized, and we want to de-emphasize the use of primitive T_EX conditionals.

```

1524 \cs_set:Npn \use_i_after_fi:nw #1 \fi: { \fi: #1 }
1525 \cs_set:Npn \use_i_after_else:nw #1 \else: #2 \fi: { \fi: #1 }
1526 \cs_set:Npn \use_i_after_or:nw #1 \or: #2 \fi: { \fi: #1 }
1527 \cs_set:Npn \use_i_after_orelse:nw #1#2#3 \fi: { \fi: #1 }

```

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

Deprecated 2011-09-07, for removal by 2011-10-07.

```

\cs_set_eq:NwN

```

```

1528 \tex_let:D \cs_set_eq:NwN \tex_let:D
      (End definition for \cs_set_eq:NwN. This function is documented on page ??.)
1529 </initex | package>

```

182 l3expan implementation

1530 `*initex | package)`

We start by ensuring that the required packages are loaded.

1531 `*package)`

1532 `\ProvidesExplPackage`

1533 `{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}`

1534 `\package_check_loaded_expl:`

1535 `\package)`

`\exp_after:wN` These are defined in l3basics.

`\exp_not:N` (End definition for `\exp_after:wN`. This function is documented on page 29.)

`\exp_not:n`

182.1 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 182.3. In section 182.2 some common cases are coded by a more direct method for efficiency, typically using calls to `\exp_after:wN`.

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

1536 `\cs_new_nopar:Npn \l_exp_tl { }`

(End definition for `\l_exp_tl`. This function is documented on page 30.)

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.

1537 `\cs_new:Npn \exp_arg_next:nnn #1#2#3 { #2 \::: { #3 {#1} } }`

1538 `\cs_new:Npn \exp_arg_next_nobrace:nnn #1#2#3 { #2 \::: { #3 #1 } }`

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

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.

`\:::` The end marker is just another name for the identity function.

```

1539 \cs_new:Npn \::: #1 {#1}
      (End definition for \:::. This function is documented on page 30.)

```

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

```

1540 \cs_new:Npn \::n #1 \::: #2#3 { #1 \::: { #2 {#3} } }
      (End definition for \::n. This function is documented on page 30.)

```

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

```

1541 \cs_new:Npn \::N #1 \::: #2#3 { #1 \::: {#2#3} }
      (End definition for \::N. This function is documented on page 30.)

```

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

```

1542 \cs_new:Npn \::c #1 \::: #2#3
1543 { \exp_after:wN \exp_arg_next:nobrace:nnn \cs:w #3 \cs_end: {#1} {#2} }
      (End definition for \::c. This function is documented on page 30.)

```

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

```

1544 \cs_new:Npn \::o #1 \::: #2#3
1545 { \exp_after:wN \exp_arg_next:nnn \exp_after:wN {#3} {#1} {#2} }
      (End definition for \::o. This function is documented on page 30.)

```

`\::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 `TEX` 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 `TEX` has not tried to execute any of the non-expandable tokens. This is what differentiates this function from the `x` argument type.

```

1546 \cs_new:Npn \::f #1 \::: #2#3
1547 {
1548   \exp_after:wN \exp_arg_next:nnn
1549   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1550   {#1} {#2}
1551 }
1552 \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.

```

1553 \cs_new_protected:Npn \::x #1 \::: #2#3
1554 {
1555   \cs_set_nopar:Npx \l_exp_tl { {#3} }
1556   \exp_after:wN \exp_arg_next:nnn \l_exp_tl {#1} {#2}
1557 }

```

(End definition for `\::x`. This function is documented on page 30.)

`\::v` These functions return the value of a register, i.e., one of `tl`, `num`, `int`, `skip`, `dim` and `muskip`. The `V` version expects a single token whereas `v` like `c` creates a `cname` 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.

```

1558 \cs_new:Npn \::V #1 \::: #2#3
1559 {
1560   \exp_after:wN \exp_arg_next:nnn
1561   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1562   {#1} {#2}
1563 }
1564 \cs_new:Npn \::v # 1\::: #2#3
1565 {
1566   \exp_after:wN \exp_arg_next:nnn
1567   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1568   {#1} {#2}
1569 }

```

(End definition for `\::v`. This function is documented on page 30.)

`\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:`.

```

1570 \cs_new_nopar:Npn \exp_eval_register:N #1
1571 {
1572   \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.

```

1573   \if_meaning:w \scan_stop: #1
1574   \exp_eval_error_msg:w
1575   \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`.

```

1576     \else:
1577       \exp_after:wN \use_i_ii:nnn
1578     \fi:
1579     \exp_after:wN \c_zero \tex_the:D #1
1580   }
1581   \cs_new_nopar:Npn \exp_eval_register:c #1
1582     { \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}

1583 \cs_new:Npn \exp_eval_error_msg:w #1 \tex_the:D #2
1584 {
1585   \fi:
1586   \fi:
1587   \msg_expandable_error:n { Erroneous ~ variable ~ #2 used! }
1588   \c_zero
1589 }

```

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

182.2 Hand-tuned definitions

One of the most important features of these functions is that they are fully expandable and therefore allow to prefix them with `\tex_global:D` for example.

```

\exp_args:No Those lovely runs of expansion!
\exp_args:NNo 1590 \cs_new:Npn \exp_args:No #1#2 { \exp_after:wN #1 \exp_after:wN {#2} }
\exp_args:NNNo 1591 \cs_new:Npn \exp_args:NNo #1#2#3
                { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN {#3} }
                1592
                1593 \cs_new:Npn \exp_args:NNNo #1#2#3#4
                { \exp_after:wN #1 \exp_after:wN#2 \exp_after:wN #3 \exp_after:wN {#4} }
                1594
                (End definition for \exp_args:No. This function is documented on page 28.)

```

`\exp_args:Nc` In `l3basics`
(End definition for `\exp_args:Nc`. This function is documented on page 26.)

`\exp_args:cc` Here are the functions that turn their argument into csnames but are expandable.

```

\exp_args:NNc 1595 \cs_new:Npn \exp_args:cc #1#2
\exp_args:Ncc 1596 { \cs:w #1 \exp_after:wN \cs_end: \cs:w #2 \cs_end: }
\exp_args:Nccc 1597 \cs_new:Npn \exp_args:NNc #1#2#3
1598 { \exp_after:wN #1 \exp_after:wN #2 \cs:w # 3\cs_end: }
1599 \cs_new:Npn \exp_args:Ncc #1#2#3
1600 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: \cs:w #3 \cs_end: }
1601 \cs_new:Npn \exp_args:Nccc #1#2#3#4
1602 {
1603   \exp_after:wN #1
1604   \cs:w #2 \exp_after:wN \cs_end:
1605   \cs:w #3 \exp_after:wN \cs_end:
1606   \cs:w #4 \cs_end:
1607 }

```

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

```

\exp_args:Nf
\exp_args:NV 1608 \cs_new:Npn \exp_args:Nf #1#2
\exp_args:Nv 1609 { \exp_after:wN #1 \exp_after:wN { \tex_romannumeral:D -'0 #2 } }
\exp_args:Nx 1610 \cs_new:Npn \exp_args:Nv #1#2
1611 {
1612   \exp_after:wN #1 \exp_after:wN
1613   { \tex_romannumeral:D \exp_eval_register:c {#2} }
1614 }
1615 \cs_new:Npn \exp_args:NV #1#2
1616 {
1617   \exp_after:wN #1 \exp_after:wN
1618   { \tex_romannumeral:D \exp_eval_register:N #2 }
1619 }

```

(End definition for \exp_args:Nf and others. These functions are documented on page 27.)

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

```

\exp_args:NNf 1620 \cs_new:Npn \exp_args:NNf #1#2#3
\exp_args:NVV 1621 {
\exp_args:Ncf 1622   \exp_after:wN #1
1623   \exp_after:wN #2
1624   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1625 }
1626 \cs_new:Npn \exp_args:NNv #1#2#3
1627 {
1628   \exp_after:wN #1
1629   \exp_after:wN #2
1630   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1631 }
1632 \cs_new:Npn \exp_args:NNV #1#2#3
1633 {
1634   \exp_after:wN #1

```



```

1635     \exp_after:wN #2
1636     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1637   }
1638   \cs_new:Npn \exp_args:Nco #1#2#3
1639   {
1640     \exp_after:wN #1
1641     \cs:w #2 \exp_after:wN \cs_end:
1642     \exp_after:wN {#3}
1643   }
1644   \cs_new:Npn \exp_args:Ncf #1#2#3
1645   {
1646     \exp_after:wN #1
1647     \cs:w #2 \exp_after:wN \cs_end:
1648     \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1649   }
1650   \cs_new_nopar:Npn \exp_args:NVV #1#2#3
1651   {
1652     \exp_after:wN #1
1653     \exp_after:wN { \tex_romannumeral:D \exp_after:wN
1654       \exp_eval_register:N \exp_after:wN #2 \exp_after:wN }
1655     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1656   }

```

(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 1657 \cs_new:Npn \exp_args:NNNV #1#2#3#4
\exp_args:NcNo 1658 {
\exp_args:NNNV 1659   \exp_after:wN #1
1660   \exp_after:wN #2
1661   \exp_after:wN #3
1662   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #4 }
1663 }
1664 \cs_new:Npn \exp_args:NcNc #1#2#3#4
1665 {
1666   \exp_after:wN #1
1667   \cs:w #2 \exp_after:wN \cs_end:
1668   \exp_after:wN #3
1669   \cs:w #4 \cs_end:
1670 }
1671 \cs_new:Npn \exp_args:NcNo #1#2#3#4
1672 {
1673   \exp_after:wN #1
1674   \cs:w #2 \exp_after:wN \cs_end:
1675   \exp_after:wN #3
1676   \exp_after:wN {#4}
1677 }
1678 \cs_new:Npn \exp_args:Ncco #1#2#3#4
1679 {
1680   \exp_after:wN #1
1681   \cs:w #2 \exp_after:wN \cs_end:

```

```

1682 \cs:w #3 \exp_after:wN \cs_end:
1683 \exp_after:wN {#4}
1684 }

```

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

182.3 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

```

1685 \cs_new_protected_nopar:Npn \exp_args:Nx { \::x \::: }

```

(End definition for \exp_args:Nx. This function is documented on page 27.)

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

```

\exp_args:Nnc 1686 \cs_new_nopar:Npn \exp_args:Nnc { \::n \::c \::: }
\exp_args:Ncx 1687 \cs_new_nopar:Npn \exp_args:Nfo { \::f \::o \::: }
\exp_args:Nfo 1688 \cs_new_nopar:Npn \exp_args:Nff { \::f \::f \::: }
\exp_args:Nff 1689 \cs_new_nopar:Npn \exp_args:Nnf { \::n \::f \::: }
\exp_args:Nnf 1690 \cs_new_nopar:Npn \exp_args:Nno { \::n \::o \::: }
\exp_args:Nno 1691 \cs_new_nopar:Npn \exp_args:NnV { \::n \::V \::: }
\exp_args:NnV 1692 \cs_new_nopar:Npn \exp_args:Noc { \::o \::c \::: }
\exp_args:NnV 1693 \cs_new_nopar:Npn \exp_args:Noo { \::o \::o \::: }
\exp_args:Nnx 1694 \cs_new_protected_nopar:Npn \exp_args:NNx { \::N \::x \::: }
\exp_args:Noo 1695 \cs_new_protected_nopar:Npn \exp_args:Ncx { \::c \::x \::: }
\exp_args:Noc 1696 \cs_new_protected_nopar:Npn \exp_args:Nnx { \::n \::x \::: }
\exp_args:Nox 1697 \cs_new_protected_nopar:Npn \exp_args:Nox { \::o \::x \::: }
\exp_args:Nxo 1698 \cs_new_protected_nopar:Npn \exp_args:Nxo { \::x \::o \::: }
\exp_args:Nxx 1699 \cs_new_protected_nopar:Npn \exp_args:Nxx { \::x \::x \::: }

```

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

\exp_args:Nccx

```

\exp_args:Ncnx 1700 \cs_new_nopar:Npn \exp_args:NNno { \::N \::n \::o \::: }
\exp_args:NNno 1701 \cs_new_nopar:Npn \exp_args:NNoo { \::N \::o \::o \::: }
\exp_args:Nnno 1702 \cs_new_nopar:Npn \exp_args:Nnnc { \::n \::n \::c \::: }
\exp_args:Nnnx 1703 \cs_new_nopar:Npn \exp_args:Nnno { \::n \::n \::o \::: }
\exp_args:Nnox 1704 \cs_new_nopar:Npn \exp_args:Nooo { \::o \::o \::o \::: }
\exp_args:Nooo 1705 \cs_new_protected_nopar:Npn \exp_args:NNnx { \::N \::n \::x \::: }
\exp_args:Noox 1706 \cs_new_protected_nopar:Npn \exp_args:NNox { \::N \::o \::x \::: }
\exp_args:Nnnx 1707 \cs_new_protected_nopar:Npn \exp_args:Nnnx { \::n \::n \::x \::: }
\exp_args:Nnnc 1708 \cs_new_protected_nopar:Npn \exp_args:Nnox { \::n \::o \::x \::: }
\exp_args:NNnx 1709 \cs_new_protected_nopar:Npn \exp_args:Nccx { \::c \::c \::x \::: }
\exp_args:NNoo 1710 \cs_new_protected_nopar:Npn \exp_args:Ncnx { \::c \::n \::x \::: }
\exp_args:NNox 1711 \cs_new_protected_nopar:Npn \exp_args:Noox { \::o \::o \::x \::: }

```

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

182.4 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
\::V_unbraced
\::v_unbraced
1712 \cs_new:Npn \exp_arg_last_unbraced:nn #1#2 { #2#1 }
1713 \cs_new:Npn \::f_unbraced \::: #1#2
1714 {
1715     \exp_after:wN \exp_arg_last_unbraced:nn
1716     \exp_after:wN { \tex_romannumeral:D -'0 #2 } {#1}
1717 }
1718 \cs_new:Npn \::o_unbraced \::: #1#2
1719 { \exp_after:wN \exp_arg_last_unbraced:nn \exp_after:wN {#2} {#1} }
1720 \cs_new:Npn \::V_unbraced \::: #1#2
1721 {
1722     \exp_after:wN \exp_arg_last_unbraced:nn
1723     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #2 } {#1}
1724 }
1725 \cs_new:Npn \::v_unbraced \::: #1#2
1726 {
1727     \exp_after:wN \exp_arg_last_unbraced:nn
1728     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#2} } {#1}
1729 }

```

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

`\exp_last_unbraced:NV` Now the business end: most of these are hand-tuned for speed, but the general system is in place.

```

\exp_last_unbraced:Nv
\exp_last_unbraced:Nf
\exp_last_unbraced:No
\exp_last_unbraced:NcV
\exp_last_unbraced:NNV
\exp_last_unbraced:NNo
\exp_last_unbraced:Nno
\exp_last_unbraced:Noo
\exp_last_unbraced:Nfo
\exp_last_unbraced:NNNV
\exp_last_unbraced:NNNo
1730 \cs_new:Npn \exp_last_unbraced:NV #1#2
1731 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:N #2 }
1732 \cs_new:Npn \exp_last_unbraced:Nv #1#2
1733 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:c {#2} }
1734 \cs_new:Npn \exp_last_unbraced:No #1#2 { \exp_after:wN #1 #2 }
1735 \cs_new:Npn \exp_last_unbraced:Nf #1#2
1736 { \exp_after:wN #1 \tex_romannumeral:D -'0 #2 }
1737 \cs_new:Npn \exp_last_unbraced:NcV #1#2#3
1738 {
1739     \exp_after:wN #1
1740     \cs:w #2 \exp_after:wN \cs_end:
1741     \tex_romannumeral:D \exp_eval_register:N #3
1742 }
1743 \cs_new:Npn \exp_last_unbraced:NNV #1#2#3
1744 {
1745     \exp_after:wN #1
1746     \exp_after:wN #2
1747     \tex_romannumeral:D \exp_eval_register:N #3
1748 }
1749 \cs_new:Npn \exp_last_unbraced:NNo #1#2#3
1750 { \exp_after:wN #1 \exp_after:wN #2 #3 }
1751 \cs_new_nopar:Npn \exp_last_unbraced:Nno { \::n \::o_unbraced \::: }
1752 \cs_new_nopar:Npn \exp_last_unbraced:Noo { \::o \::o_unbraced \::: }

```

```

1753 \cs_new_nopar:Npn \exp_last_unbraced:Nfo { \::f \::o_unbraced \::: }
1754 \cs_new:Npn \exp_last_unbraced:NNNV #1#2#3#4
1755 {
1756   \exp_after:wN #1
1757   \exp_after:wN #2
1758   \exp_after:wN #3
1759   \tex_romannumeral:D \exp_eval_register:N #4
1760 }
1761 \cs_new:Npn \exp_last_unbraced:NNNo #1#2#3#4
1762 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN #3 #4 }

```

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

`\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.

```

1763 \cs_new:Npn \exp_last_two_unbraced:Noo #1#2#3
1764 { \exp_after:wN \exp_last_two_unbraced_aux:noN \exp_after:wN {#3} {#2} #1 }
1765 \cs_new:Npn \exp_last_two_unbraced_aux:noN #1#2#3
1766 { \exp_after:wN #3 #2 #1 }

```

(End definition for \exp_last_two_unbraced:Noo. This function is documented on page 29.)

182.5 Preventing expansion

```

\exp_not:o
\exp_not:f 1767 \cs_new:Npn \exp_not:o #1 { \etex_unexpanded:D \exp_after:wN {#1} }
\exp_not:V 1768 \cs_new:Npn \exp_not:f #1
\exp_not:v 1769 { \etex_unexpanded:D \exp_after:wN { \tex_romannumeral:D -'0 #1 } }
1770 \cs_new:Npn \exp_not:V #1
1771 {
1772   \etex_unexpanded:D \exp_after:wN
1773   { \tex_romannumeral:D \exp_eval_register:N #1 }
1774 }
1775 \cs_new:Npn \exp_not:v #1
1776 {
1777   \etex_unexpanded:D \exp_after:wN
1778   { \tex_romannumeral:D \exp_eval_register:c {#1} }
1779 }

```

(End definition for \exp_not:o. This function is documented on page 30.)

`\exp_not:c` A helper function.

```

1780 \cs_new:Npn \exp_not:c #1 { \exp_after:wN \exp_not:N \cs:w #1 \cs_end: }

```

(End definition for \exp_not:c. This function is documented on page 29.)

182.6 Defining function variants

`\cs_generate_variant:Nn` #1 : Base form of a function; *e.g.*, `\tl_set:Nn`
`\cs_generate_variant_aux:nnNNn` #2 : One or more variant argument specifiers; *e.g.*, `{Nx,c,cx}`
`\cs_generate_variant_aux:Nnnw` Test whether the base function is protected or not and define `\cs_tmp:w` as either
`\cs_generate_variant_aux:NNn` `\cs_new_nopar:Npx` or `\cs_new_protected_nopar:Npx`, then used to define all the 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`.

```

1781 \cs_new_protected:Npn \cs_generate_variant:Nn #1
1782 {
1783   \chk_if_exist_cs:N #1
1784   \cs_generate_variant_aux:N #1
1785   \cs_split_function:NN #1 \cs_generate_variant_aux:nnNNn
1786   #1
1787 }

```

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.

```

1788 \cs_new:Npn \cs_generate_variant_aux:nnNNn #1#2#3#4#5
1789 { \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.

```

1790 \cs_new:Npn \cs_generate_variant_aux:Nnnw #1#2#3#4 ,
1791 {
1792   \if:w ? #4
1793     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1794   \fi:
1795   \exp_args:Nnc \cs_generate_variant_aux:NNn
1796   #1
1797   {
1798     #2 : #4
1799     \exp_after:wN \use_i_delimit_by_q_stop:nw
1800     \use_none:nnnnnnnnn #4
1801     \use_none:nnnnnnnnn
1802     \use_none:nnnnnnnnn
1803     \use_none:nnnnnnnnn
1804     \use_none:nnnnnnn

```

```

1805         \use_none:nnnnn
1806         \use_none:nnnn
1807         \use_none:nnn
1808         \use_none:nn
1809         \use_none:n
1810         { }
1811         \q_stop
1812         #3
1813     }
1814     {#4}
1815     \cs_generate_variant_aux:Nnnw #1 {#2} {#3}
1816 }

```

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.

```

1817 \cs_new:Npn \cs_generate_variant_aux:NNn #1 #2 #3
1818 {
1819     \cs_if_free:NTF #2
1820     {
1821         \cs_tmp:w #2 { \exp_not:c { exp_args:N #3 } \exp_not:N #1 }
1822         \cs_generate_internal_variant:n {#3}
1823     }
1824     {
1825         \iow_log:x
1826         {
1827             Variant~\token_to_str:N #2~%
1828             already~defined;~ not~ changing~ it~on~line~%
1829             \tex_the:D \tex_inputlineno:D
1830         }
1831     }
1832 }

```

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

`\cs_generate_variant_aux:N` The idea here is to pick up protected parent functions, using the nature of the meaning 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.

```

1833 \group_begin:
1834     \tex_lccode:D ‘\Z = ‘\d \scan_stop:
1835     \tex_lccode:D ‘\? = ‘\ \ \scan_stop:
1836     \tex_catcode:D ‘\P = 12 \scan_stop:
1837     \tex_catcode:D ‘\R = 12 \scan_stop:
1838     \tex_catcode:D ‘\O = 12 \scan_stop:
1839     \tex_catcode:D ‘\T = 12 \scan_stop:
1840     \tex_catcode:D ‘\E = 12 \scan_stop:
1841     \tex_catcode:D ‘\C = 12 \scan_stop:
1842     \tex_catcode:D ‘\Z = 12 \scan_stop:
1843     \tex_lowercase:D

```

```

1844 {
1845   \group_end:
1846   \cs_new_nopar:Npn \cs_generate_variant_aux:N #1
1847   {
1848     \exp_after:wN \cs_generate_variant_aux:w
1849     \token_to_meaning:N #1
1850     \q_mark \cs_new_protected_nopar:Npx
1851     ? PROTECTEZ
1852     \q_mark \cs_new_nopar:Npx
1853     \q_stop
1854   }
1855   \cs_new:Npn \cs_generate_variant_aux:w
1856   #1 ? PROTECTEZ #2 \q_mark #3 #4 \q_stop
1857   {
1858     \cs_set_eq:NN \cs_tmp:w #3
1859   }
1860 }

```

(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`

```

1861 \cs_new_protected:Npn \cs_generate_internal_variant:n #1
1862 {
1863   \cs_if_free:cT { exp_args:N #1 }
1864   {
1865     \cs_new:cpx { exp_args:N #1 }
1866     { \cs_generate_internal_variant_aux:N #1 : }
1867   }
1868 }

```

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.

```

1869 \cs_new:Npn \cs_generate_internal_variant_aux:N #1
1870 {
1871   \exp_not:c { :: #1 }
1872   \if_meaning:w : #1
1873   \exp_after:wN \use_none:n
1874   \fi:
1875   \cs_generate_internal_variant_aux:N
1876 }

```

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

182.7 Variants which cannot be created earlier

\str_if_eq:Vn These cannot come earlier as they need \cs_generate_variant:Nn.

```

1877 \cs_generate_variant:Nn \str_if_eq_p:nn { V , o }
1878 \cs_generate_variant:Nn \str_if_eq_p:nn { nV , no , VV }
1879 \cs_generate_variant:Nn \str_if_eq:nnT { V , o }

```

\str_if_eq:Vn
\str_if_eq:on
\str_if_eq:nV
\str_if_eq:no
\str_if_eq:VV

```

1880 \cs_generate_variant:Nn \str_if_eq:nnT { nV , no , VV }
1881 \cs_generate_variant:Nn \str_if_eq:nnF { V , o }
1882 \cs_generate_variant:Nn \str_if_eq:nnF { nV , no , VV }
1883 \cs_generate_variant:Nn \str_if_eq:nnTF { V , o }
1884 \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 ??.)
1885 \</initex | package>

```

183 l3prg implementation

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

```

1886 \<*initex | package>
1887 \<*package>
1888 \ProvidesExplPackage
1889   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1890 \package_check_loaded_expl:
1891 \</package>

```

183.1 Primitive conditionals

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

```

1892 \tex_let:D \if_bool:N          \tex_ifodd:D
1893 \tex_let:D \if_predicate:w      \tex_ifodd:D
      (End definition for \if_bool:N. This function is documented on page 41.)

```

183.2 Defining a set of conditional functions

`\prg_set_conditional:Npnn` 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 ??.)

183.3 The boolean data type

`\prg_new_conditional:Npnn` Boolean variables have to be initiated when they are created. Other than that there is not much to say here.

```

1894 \cs_new_protected_nopar:Npn \bool_new:N #1 { \cs_new_eq:NN #1 \c_false_bool }
1895 \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.

```

\bool_set_true:N
\bool_set_true:c
\bool_gset_true:N
\bool_gset_true:c
\bool_set_false:N
\bool_set_false:c
\bool_gset_false:N
\bool_gset_false:c

```

```

1896 \cs_new_protected_nopar:Npn \bool_set_true:N #1
1897   { \cs_set_eq:NN #1 \c_true_bool }
1898 \cs_new_protected_nopar:Npn \bool_set_false:N #1
1899   { \cs_set_eq:NN #1 \c_false_bool }

```



```

1900 \cs_new_protected_nopar:Npn \bool_gset_true:N #1
1901   { \cs_gset_eq:NN #1 \c_true_bool }
1902 \cs_new_protected_nopar:Npn \bool_gset_false:N #1
1903   { \cs_gset_eq:NN #1 \c_false_bool }
1904 \cs_generate_variant:Nn \bool_set_true:N { c }
1905 \cs_generate_variant:Nn \bool_set_false:N { c }
1906 \cs_generate_variant:Nn \bool_gset_true:N { c }
1907 \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 1908 \cs_new_eq:NN \bool_set_eq:NN \cs_set_eq:NN
\bool_set_eq:Nc 1909 \cs_new_eq:NN \bool_set_eq:Nc \cs_set_eq:Nc
\bool_set_eq:cc 1910 \cs_new_eq:NN \bool_set_eq:cN \cs_set_eq:cN
\bool_gset_eq:NN 1911 \cs_new_eq:NN \bool_set_eq:cc \cs_set_eq:cc
\bool_gset_eq:cN 1912 \cs_new_eq:NN \bool_gset_eq:NN \cs_gset_eq:NN
\bool_gset_eq:Nc 1913 \cs_new_eq:NN \bool_gset_eq:Nc \cs_gset_eq:Nc
\bool_gset_eq:cN 1914 \cs_new_eq:NN \bool_gset_eq:cN \cs_gset_eq:cN
\bool_gset_eq:cc 1915 \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 1916 \cs_new:Npn \bool_set:Nn #1#2
\bool_gset:cn 1917   { \tex_chardef:D #1 = \bool_if_p:n {#2} }
1918 \cs_new:Npn \bool_gset:Nn #1#2
1919   { \tex_global:D \tex_chardef:D #1 = \bool_if_p:n {#2} }
1920 \cs_generate_variant:Nn \bool_set:Nn { c }
1921 \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.

```

1922 \prg_new_conditional:Npnn \bool_if:N #1 { p , T , F , TF }
1923 {
1924   \if_meaning:w \c_true_bool #1
1925   \prg_return_true:
1926   \else:
1927   \prg_return_false:
1928   \fi:
1929 }
1930 \cs_generate_variant:Nn \bool_if_p:N { c }
1931 \cs_generate_variant:Nn \bool_if:NT { c }
1932 \cs_generate_variant:Nn \bool_if:NF { c }
1933 \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 1934 \bool_new:N \l_tmpa_bool
1935 \bool_new:N \g_tmpa_bool

```

(End definition for `\l_tmpa_bool` and `\g_tmpa_bool`. These functions are documented on page 36.)

183.4 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` **$\langle true \rangle$ 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` **$\langle false \rangle$ 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 $\langle false \rangle$.

`\bool_I_0:w` **$\langle true \rangle$ 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 $\langle true \rangle$.

`\bool_)_0:w` **$\langle false \rangle$ Or** Current truth value is false, logical Or seen, continue with `GetNext` to examine truth value of next boolean (sub-)expression.

`\bool_)_1:w` **$\langle true \rangle$ Close** Current truth value is true, Close seen, return $\langle true \rangle$.

`\bool_S_0:w` **$\langle false \rangle$ Close** Current truth value is false, Close seen, return $\langle false \rangle$.

`\bool_S_1:w` We introduce an additional Stop operation with the following semantics:

$\langle true \rangle$ Stop Current truth value is true, return $\langle true \rangle$.

$\langle false \rangle$ Stop Current truth value is false, return $\langle false \rangle$.

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 \TeX . 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.

```

1936 \prg_new_conditional:Npnn \bool_if:n #1 { T , F , TF }
1937 {
1938   \if_predicate:w \bool_if_p:n {#1}
1939   \prg_return_true:
1940   \else:
1941     \prg_return_false:
1942   \fi:
1943 }
1944 \cs_new:Npn \bool_if_p:n #1
1945 {
1946   \group_align_safe_begin:
1947   \bool_get_next:N ( #1 ) S
1948 }

```

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

```

1949 \cs_new:Npn \bool_get_next:N #1
1950 {
1951   \use:c
1952   {
1953     bool_
1954     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1955     :w
1956   }
1957   #1
1958 }

```

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.

```

1959 \cs_new:Npn \bool_get_not_next:N #1
1960 {
1961   \use:c
1962   {
1963     bool_not_
1964     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1965     :w
1966   }
1967   #1
1968 }

```

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

```

1969 \cs_new:Npn \bool_get_next:NN #1#2 { \bool_get_next:N #2 }
1970 \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.

```

1971 \cs_new:cpn { bool_!:w } #1#2
1972 {
1973   \if_meaning:w ( #2
1974     \exp_after:wN \bool_Not:w
1975   \else:
1976     \if_meaning:w ! #2
1977     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_next:NN
1978   \else:
1979     \exp_after:wN \exp_after:wN \exp_after:wN \bool_Not:N
1980   \fi:
1981   \fi:
1982   #2
1983 }

```

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

```

1984 \cs_new:cpn { bool_not_!:w } #1#2
1985 {
1986   \if_meaning:w ( #2
1987     \exp_after:wN \bool_not_Not:w
1988   \else:
1989     \if_meaning:w ! #2
1990     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_not_next:NN
1991   \else:
1992     \exp_after:wN \exp_after:wN \exp_after:wN \bool_not_Not:N
1993   \fi:
1994   \fi:
1995   #2
1996 }

```

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.

```

1997 \cs_new:Npn \bool_Not:w { \exp_after:wN \int_value:w \bool_get_not_next:N }
1998 \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.

```

1999 \cs_new:Npn \bool_Not:N #1
2000 {
2001   \exp_after:wN \bool_p:w
2002   \if_meaning:w #1 \c_true_bool
2003     \c_false_bool
2004   \else:
2005     \c_true_bool
2006   \fi:
2007 }
2008 \cs_new:Npn \bool_not_Not:N #1
2009 {
2010   \exp_after:wN \bool_p:w
2011   \if_meaning:w #1 \c_true_bool
2012     \c_true_bool
2013   \else:

```

```

2014         \c_false_bool
2015     \fi:
2016 }

```

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 `!(...)`.

```

2017 \cs_new:cpn { bool_( :w } #1
2018 { \exp_after:wN \bool_cleanup:N \int_value:w \bool_get_next:N }
2019 \cs_new:cpn { bool_not_( :w } #1
2020 { \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.

```

2021 \cs_new:cpn { bool_p:w } { \exp_after:wN \bool_cleanup:N \int_value:w }
2022 \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.

```

2023 \cs_new:Npn \bool_cleanup:N #1
2024 {
2025     \exp_after:wN \bool_choose:NN \exp_after:wN #1
2026     \int_to_roman:w - '\q
2027 }
2028 \cs_new:Npn \bool_not_cleanup:N #1
2029 {
2030     \exp_after:wN \bool_not_choose:NN \exp_after:wN #1
2031     \int_to_roman:w - '\q
2032 }

```

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

```

2033 \cs_new_nopar:Npn \bool_choose:NN #1#2 { \use:c { bool_ #2 _ #1 :w } }
2034 \cs_new_nopar:Npn \bool_not_choose:NN #1#2 { \use:c { bool_not_ #2 _ #1 :w } }

```

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

```

2035 \cs_new_nopar:cpn { bool_&_1:w } & { \bool_get_next:N }
2036 \cs_new_nopar:cpn { bool_|_0:w } | { \bool_get_next:N }
2037 \cs_new_nopar:cpn { bool_not_&_0:w } & { \bool_get_next:N }
2038 \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.

```

2039 \cs_new_nopar:cpn { bool_)_0:w } { \c_false_bool }
2040 \cs_new_nopar:cpn { bool_)_1:w } { \c_true_bool }
2041 \cs_new_nopar:cpn { bool_not_)_0:w } { \c_true_bool }
2042 \cs_new_nopar:cpn { bool_not_)_1:w } { \c_false_bool }
2043 \cs_new_nopar:cpn { bool_S_0:w } { \group_align_safe_end: \c_false_bool }
2044 \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.

```

2045 \cs_new_nopar:cpn { bool_&_0:w } & { \bool_eval_skip_to_end:Nw \c_false_bool }

```

```

2046 \cs_new_nopar:cpn { bool_|_1:w } | { \bool_eval_skip_to_end:Nw \c_true_bool }
2047 \cs_new_nopar:cpn { bool_not_&_1:w } &
2048   { \bool_eval_skip_to_end:Nw \c_false_bool }
2049 \cs_new_nopar:cpn { bool_not_|_0:w } |
2050   { \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.

```

2051 %% (
2052 \cs_new:Npn \bool_eval_skip_to_end:Nw #1#2 )
2053 {
2054   \bool_eval_skip_to_end_aux:Nw #1#2 ( % )
2055   \q_no_value \q_stop
2056   {#2}
2057 }

```

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

```

2058 \cs_new:Npn \bool_eval_skip_to_end_aux:Nw #1#2 ( #3#4 \q_stop #5 % )
2059 {
2060   \quark_if_no_value:NTF #3
2061   {#1}
2062   { \bool_eval_skip_to_end_aux_ii:Nw #1 #5 }
2063 }

```

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!

```

2064 \cs_new:Npn \bool_eval_skip_to_end_aux_ii:Nw #1#2 ( #3 )
2065 { % (
2066   \bool_eval_skip_to_end:Nw #1#3 )
2067 }

```

`\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.

```

2068 \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.

```

2069 \cs_new:Npn \bool_xor_p:nn #1#2
2070 {
2071   \int_compare:nNnTF { \bool_if_p:n {#1} } = { \bool_if_p:n {#2} }
2072   \c_false_bool
2073   \c_true_bool
2074 }

```

183.5 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
2075 \cs_new:Npn \bool_while_do:Nn #1#2
2076 { \bool_if:NT #1 { #2 \bool_while_do:Nn #1 {#2} } }
2077 \cs_new:Npn \bool_until_do:Nn #1#2
2078 { \bool_if:NF #1 { #2 \bool_until_do:Nn #1 {#2} } }

```

```

2079 \cs_generate_variant:Nn \bool_while_do:Nn { c }
2080 \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
\bool_do_while:cn after executing the body. Otherwise identical to the above functions.
\bool_do_until:Nn
\bool_do_until:cn
2081 \cs_new:Npn \bool_do_while:Nn #1#2
2082 { #2 \bool_if:NT #1 { \bool_do_while:Nn #1 {#2} } }
2083 \cs_new:Npn \bool_do_until:Nn #1#2
2084 { #2 \bool_if:NF #1 { \bool_do_until:Nn #1 {#2} } }
2085 \cs_generate_variant:Nn \bool_do_while:Nn { c }
2086 \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
2087 \cs_new:Npn \bool_while_do:nn #1#2
2088 {
2089   \bool_if:nT {#1}
2090   {
2091     #2
2092     \bool_while_do:nn {#1} {#2}
2093   }
2094 }
2095 \cs_new:Npn \bool_do_while:nn #1#2
2096 {
2097   #2
2098   \bool_if:nT {#1} { \bool_do_while:nn {#1} {#2} }
2099 }
2100 \cs_new:Npn \bool_until_do:nn #1#2
2101 {
2102   \bool_if:nF {#1}
2103   {
2104     #2
2105     \bool_until_do:nn {#1} {#2}
2106   }
2107 }
2108 \cs_new:Npn \bool_do_until:nn #1#2
2109 {
2110   #2
2111   \bool_if:nF {#1} { \bool_do_until:nn {#1} {#2} }
2112 }

```

183.6 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.

```

2113 \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 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`.

```

2114 \cs_new:Npn \prg_case_int:nnn #1
2115 {
2116   \tex_romannumeral:D
2117   \exp_args:Nf \prg_case_int_aux:nnn { \int_eval:n {#1} }
2118 }
2119 \cs_new:Npn \prg_case_int_aux:nnn #1 #2 #3
2120 { \prg_case_int_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2121 \cs_new:Npn \prg_case_int_aux:nw #1#2#3
2122 {
2123   \int_compare:nNnTF {#1} = {#2}
2124   { \prg_case_end:nw {#3} }
2125   { \prg_case_int_aux:nw {#1} }
2126 }

```

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

```

2127 \cs_new:Npn \prg_case_dim:nnn #1
2128 {
2129   \tex_romannumeral:D
2130   \exp_args:Nf \prg_case_dim_aux:nnn { \dim_eval:n {#1} }
2131 }
2132 \cs_new:Npn \prg_case_dim_aux:nnn #1 #2 #3
2133 { \prg_case_dim_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2134 \cs_new:Npn \prg_case_dim_aux:nw #1#2#3
2135 {
2136   \dim_compare:nNnTF {#1} = {#2}
2137   { \prg_case_end:nw {#3} }
2138   { \prg_case_dim_aux:nw {#1} }
2139 }

```

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

```

2140 \cs_new:Npn \prg_case_str:nnn #1#2#3
2141 {
2142   \tex_romannumeral:D
2143   \prg_case_str_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2144 }
2145 \cs_new:Npn \prg_case_str_aux:nw #1#2#3
2146 {
2147   \str_if_eq:nnTF {#1} {#2}
2148   { \prg_case_end:nw {#3} }
2149   { \prg_case_str_aux:nw {#1} }
2150 }
2151 \cs_generate_variant:Nn \prg_case_str:nnn { o }
2152 \cs_new:Npn \prg_case_str:xxn #1#2#3
2153 {

```

```

2154 \tex_romannumeral:D
2155 \prg_case_str_x_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2156 }
2157 \cs_new:Npn \prg_case_str_x_aux:nw #1#2#3
2158 {
2159 \str_if_eq:xxTF {#1} {#2}
2160 { \prg_case_end:nw {#3} }
2161 { \prg_case_str_x_aux:nw {#1} }
2162 }

```

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

```

2163 \cs_new:Npn \prg_case_tl:Nnn #1#2#3
2164 {
2165 \tex_romannumeral:D
2166 \prg_case_tl_aux:Nw #1 #2 #1 {#3} \q_recursion_stop
2167 }
2168 \cs_new:Npn \prg_case_tl_aux:Nw #1#2#3
2169 {
2170 \tl_if_eq:NNTF #1 #2
2171 { \prg_case_end:nw {#3} }
2172 { \prg_case_tl_aux:Nw #1 }
2173 }
2174 \cs_generate_variant:Nn \prg_case_tl:Nnn { c }

```

183.7 Producing n copies

\prg_replicate:nn This function uses a cascading csname technique by David Kastrup (who else :-)
\prg_replicate_aux:N The idea is to make the input 25 result in first adding five, and then 20 copies of
\prg_replicate_first_aux:N the code to be replicated. The technique uses cascading csnames which means that we
\prg_replicate_ start building several csnames so we end up with a list of functions to be called in reverse
\prg_replicate_0:n order. This is important here (and other places) because it means that we can for instance
\prg_replicate_1:n make the function that inserts five copies of something to also hand down ten to the next
\prg_replicate_2:n function in line. This is exactly what happens here: in the example with 25 then the
\prg_replicate_3:n next function is the one that inserts two copies but it sees the ten copies handed down by
\prg_replicate_4:n the previous function. In order to avoid the last function to insert say, 100 copies of the
\prg_replicate_5:n original argument just to gobble them again we define separate functions to be inserted
\prg_replicate_6:n first. These functions also close the expansion of \int_to_roman:w, which ensures that
\prg_replicate_7:n \prg_replicate:nn only requires two steps of expansion.
\prg_replicate_8:n
\prg_replicate_9:n

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*}
\prg_replicate_first_0:n An alternative approach is to create a string of m's with \int_to_roman:w which can be
\prg_replicate_first_1:n done with just four macros but that method has its own problems since it can exhaust
\prg_replicate_first_2:n the string pool. Also, it is considerably slower than what we use here so the few extra
\prg_replicate_first_3:n csnames are well spent I would say.
\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

```

2175 \cs_new_nopar:Npn \prg_replicate:nn #1

```

Then comes all the functions that do the hard work of inserting all the copies.

Users shouldn't ask for something to be replicated once or even not at all but...

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

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.

243

```

2218 { \msg_expandable_error:n { Zero-step-size-for-stepwise-function. } }
2219 {
2220   \int_compare:nNnTF {#2} > \c_zero
2221     { \exp_args:Nf \prg_stepwise_function_incr:nnnN }
2222     { \exp_args:Nf \prg_stepwise_function_decr:nnnN }
2223     { \int_eval:n {#1} } {#2} {#3} #4
2224   }
2225 }
2226 \cs_new:Npn \prg_stepwise_function_incr:nnnN #1#2#3#4
2227 {
2228   \int_compare:nNnF {#1} > {#3}
2229   {
2230     #4 {#1}
2231     \exp_args:Nf \prg_stepwise_function_incr:nnnN
2232     { \int_eval:n { #1 + #2 } } {#2} {#3} #4
2233   }
2234 }
2235 \cs_new:Npn \prg_stepwise_function_decr:nnnN #1#2#3#4
2236 {
2237   \int_compare:nNnF {#1} < {#3}
2238   {
2239     #4 {#1}
2240     \exp_args:Nf \prg_stepwise_function_decr:nnnN
2241     { \int_eval:n { #1 + #2 } } {#2} {#3} #4
2242   }
2243 }

```

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

\g_prg_stepwise_level_int For nesting, the usual approach of using a counter.

```

2244 \int_new:N \g_prg_stepwise_level_int

```

(End definition for \g_prg_stepwise_level_int. 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 nesting safe (as seen in other in line functions), and map that function using \prg_stepwise_variable:nnnNn \prg_stepwise_aux:NNnnnn stepwise_function:nnnN.

```

2245 \cs_new_protected:Npn \prg_stepwise_inline:nnnn
2246 {
2247   \exp_args:NNc \prg_stepwise_aux:NNnnnn
2248   \cs_gset_nopar:Npn
2249   { g_prg_stepwise_ \int_use:N \g_prg_stepwise_level_int :n }
2250 }
2251 \cs_new_protected:Npn \prg_stepwise_variable:nnnNn #1#2#3#4#5
2252 {
2253   \exp_args:NNc \prg_stepwise_aux:NNnnnn
2254   \cs_gset_nopar:Npx
2255   { g_prg_stepwise_ \int_use:N \g_prg_stepwise_level_int :n }
2256   {#1}{#2}{#3}
2257   {
2258     \tl_set:Nn \exp_not:N #4 {##1}

```

```

2259         \exp_not:n {#5}
2260     }
2261 }
2262 \cs_new_protected:Npn \prg_stepwise_aux:NNnnnn #1#2#3#4#5#6
2263 {
2264     #1 #2 ##1 {#6}
2265     \int_gincr:N \g_pr_g_stepwise_level_int
2266     \prg_stepwise_function:nnnN {#3}{#4}{#5} #2
2267     \int_gdecr:N \g_pr_g_stepwise_level_int
2268 }

```

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

183.8 Detecting TeX'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.

```

2269 \prg_new_conditional:Npnn \mode_if_vertical: { p , T , F , TF }
2270 { \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.

```

2271 \prg_new_conditional:Npnn \mode_if_horizontal: { p , T , F , TF }
2272 { \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.

```

2273 \prg_new_conditional:Npnn \mode_if_inner: { p , T , F , TF }
2274 { \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.

```

2275 \prg_new_conditional:Npnn \mode_if_math: { p , T , F , TF }
2276 { \if_mode_math: \prg_return_true: \else: \prg_return_false: \fi: }

```

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

183.9 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.

```
2277 \cs_new_nopar:Npn \group_align_safe_begin:
2278 { \if_int_compare:w \if_false: { \fi: ' } = \c_zero \fi: }
2279 \cs_new_nopar:Npn \group_align_safe_end:
2280 { \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 T_EX is in the beginning of an align cell (right after the `\cr`) it is in a somewhat strange mode as it is looking ahead to find an `\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 T_EX's scanning ahead. On the other hand we don't want to insert a `\scan_stop:` every time as that will destroy kerning between 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: }
    }
  }
}
```

⁴Unless we enforce an extra pass with an appropriate value of `\pretolerance`.

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.

```
2281 \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.

```
\prg_variable_get_scope_aux:w
\prg_variable_get_type:N
\prg_variable_get_type:w
2282 \group_begin:
2283   \tex_lccode:D '\& = '\g \scan_stop:
2284   \tex_catcode:D '\& = \c_twelve
2285   \tl_to_lowercase:n
2286   {
2287     \group_end:
2288     \cs_new_nopar:Npn \prg_variable_get_scope:N #1
2289     {
2290       \exp_last_unbraced:Nf \prg_variable_get_scope_aux:w
2291       { \cs_to_str:N #1 \exp_stop_f: \q_stop }
2292     }
2293     \cs_new_nopar:Npn \prg_variable_get_scope_aux:w #1#2 \q_stop
2294     { \token_if_eq_meaning:NNT & #1 { g } }
2295   }
2296 \group_begin:
2297   \tex_lccode:D '\& = '\_ \scan_stop:
2298   \tex_catcode:D '\& = \c_twelve
2299   \tl_to_lowercase:n
2300   {
2301     \group_end:
2302     \cs_new_nopar:Npn \prg_variable_get_type:N #1
2303     {
2304       \exp_after:wN \prg_variable_get_type_aux:w
2305       \token_to_str:N #1 & a \q_stop
2306     }
2307     \cs_new_nopar:Npn \prg_variable_get_type_aux:w #1 & #2#3 \q_stop
2308     {
2309       \token_if_eq_meaning:NNTF a #2
2310       {#1}
2311       { \prg_variable_get_type_aux:w #2#3 \q_stop }
2312     }
2313   }
      (End definition for \prg_variable_get_scope:N. This function is documented on page ??.)
```

183.10 Experimental programmings functions

`\prg_define_quicksort:nnn` `#1` is the name, `#2` and `#3` are the tokens enclosing the argument. For the somewhat strange `<clist>` type which doesn't enclose the items but uses a separator we define it by hand afterwards. When doing the first pass, the algorithm wraps all elements in braces and then uses a generic quicksort which works on token lists.

As an example

```
\prg_define_quicksort:nnn{seq}{\seq_elt:w}{\seq_elt_end:w}
```

defines the user function `\seq_quicksort:n` and furthermore expects to use the two functions `\seq_quicksort_compare:nnTF` which compares the items and `\seq_quicksort_function:n` which is placed before each sorted item. It is up to the programmer to define these functions when needed. For the `seq` type a sequence is a token list variable, so one additionally has to define

```
\cs_set_nopar:Npn \seq_quicksort:N{\exp_args:No\seq_quicksort:n}
```

For details on the implementation see “Sorting in TeX’s Mouth” by Bernd Raichle. Firstly we define the function for parsing the initial list and then the braced list afterwards.

```
2314 \cs_new_protected_nopar:Npn \prg_define_quicksort:nnn #1#2#3 {
2315   \cs_set:cpx{#1_quicksort:n}##1{
2316     \exp_not:c{#1_quicksort_start_partition:w} ##1
2317     \exp_not:n{#2\q_nil#3\q_stop}
2318   }
2319   \cs_set:cpx{#1_quicksort_braced:n}##1{
2320     \exp_not:c{#1_quicksort_start_partition_braced:n} ##1
2321     \exp_not:N\q_nil\exp_not:N\q_stop
2322   }
2323   \cs_set:cpx {#1_quicksort_start_partition:w} #2 ##1 #3{
2324     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2325     \exp_not:c{#1_quicksort_do_partition_i:nnnw} {##1}{-}{-}
2326   }
2327   \cs_set:cpx {#1_quicksort_start_partition_braced:n} ##1 {
2328     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2329     \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn} {##1}{-}{-}
2330   }
2331 }
```

Now for doing the partitions.

```
2331 \cs_set:cpx {#1_quicksort_do_partition_i:nnnw} ##1##2##3 #2 ##4 #3 {
2332   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2333   {
2334     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2335     \exp_not:c{#1_quicksort_partition_greater_ii:nnnn}
2336     \exp_not:c{#1_quicksort_partition_less_ii:nnnn}
2337   }
2338   {##1}{##2}{##3}{##4}
2339 }
2340 \cs_set:cpx {#1_quicksort_do_partition_i_braced:nnnn} ##1##2##3##4 {
2341   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2342   {
2343     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2344     \exp_not:c{#1_quicksort_partition_greater_ii_braced:nnnn}
2345     \exp_not:c{#1_quicksort_partition_less_ii_braced:nnnn}
2346   }
2347   {##1}{##2}{##3}{##4}
2348 }
```



```

2349 \cs_set:cpx {#1_quicksort_do_partition_ii:nnnw} ##1##2##3 #2 ##4 #3 {
2350   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2351   {
2352     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2353     \exp_not:c{#1_quicksort_partition_less_i:nnnn}
2354     \exp_not:c{#1_quicksort_partition_greater_i:nnnn}
2355   }
2356   {##1}{##2}{##3}{##4}
2357 }
2358 \cs_set:cpx {#1_quicksort_do_partition_ii_braced:nnnn} ##1##2##3##4 {
2359   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2360   {
2361     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2362     \exp_not:c{#1_quicksort_partition_less_i_braced:nnnn}
2363     \exp_not:c{#1_quicksort_partition_greater_i_braced:nnnn}
2364   }
2365   {##1}{##2}{##3}{##4}
2366 }

```

This part of the code handles the two branches in each sorting. Again we will also have to do it braced.

```

2367 \cs_set:cpx {#1_quicksort_partition_less_i:nnnn} ##1##2##3##4{
2368   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}
2369 \cs_set:cpx {#1_quicksort_partition_less_ii:nnnn} ##1##2##3##4{
2370   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}}
2371 \cs_set:cpx {#1_quicksort_partition_greater_i:nnnn} ##1##2##3##4{
2372   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}
2373 \cs_set:cpx {#1_quicksort_partition_greater_ii:nnnn} ##1##2##3##4{
2374   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}
2375 \cs_set:cpx {#1_quicksort_partition_less_i_braced:nnnn} ##1##2##3##4{
2376   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##3}{##4}
2377 \cs_set:cpx {#1_quicksort_partition_less_ii_braced:nnnn} ##1##2##3##4{
2378   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##3}{##4}
2379 \cs_set:cpx {#1_quicksort_partition_greater_i_braced:nnnn} ##1##2##3##4{
2380   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##3}{##4}
2381 \cs_set:cpx {#1_quicksort_partition_greater_ii_braced:nnnn} ##1##2##3##4{
2382   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##3}{##4}

```

Finally, the big kahuna! This is where the sub-lists are sorted.

```

2383 \cs_set:cpx {#1_do_quicksort_braced:nnnnw} ##1##2##3##4\q_stop {
2384   \exp_not:c{#1_quicksort_braced:n}{##2}
2385   \exp_not:c{#1_quicksort_function:n}{##1}
2386   \exp_not:c{#1_quicksort_braced:n}{##3}
2387 }
2388 }

```

(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.

```

2389 \prg_define_quicksort:nnn {prg}{\}{\}
      (End definition for \prg_quicksort:n. This function is documented on page 42.)

\prg_quicksort_function:n
\prg_quicksort_compare:nnTF
2390 \cs_set:Npn \prg_quicksort_function:n {\ERROR}
2391 \cs_set:Npn \prg_quicksort_compare:nnTF {\ERROR}
      (End definition for \prg_quicksort_function:n. This function is documented on page 42.)

```

183.11 Deprecated functions

These were deprecated on 2011-05-27 and will be removed entirely by 2011-08-31.

```

\prg_new_map_functions:Nn
\prg_set_map_functions:Nn
2392 \*deprecated
2393 \cs_new_protected:Npn \prg_new_map_functions:Nn #1#2 { \deprecated }
2394 \cs_new_protected:Npn \prg_set_map_functions:Nn #1#2 { \deprecated }
2395 \*deprecated
      (End definition for \prg_new_map_functions:Nn. This function is documented on page ??.)
2396 \*initex | package)

```

184 l3quark implementation

The following test files are used for this code: *m3quark001.lvt*.

```

2397 \*initex | package)
2398 \*package)
2399 \ProvidesExplPackage
2400   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2401 \package_check_loaded_expl:
2402 \*package)

\quark_new:N Allocate a new quark.
2403 \cs_new_protected_nopar:Npn \quark_new:N #1 { \tl_const:Nn #1 {#1} }
      (End definition for \quark_new:N. This function is documented on page 43.)

\q_nil Some “public” quarks. \q_stop is an “end of argument” marker, \q_nil is a empty value
\q_mark and \q_no_value marks an empty argument.
\q_no_value
\q_stop
2404 \quark_new:N \q_nil
2405 \quark_new:N \q_mark
2406 \quark_new:N \q_no_value
2407 \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.

```
2408 \quark_new:N \q_recursion_tail
2409 \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` 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. `\quark_if_recursion_tail_stop_do:Nn` 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.

```
2410 \cs_new:Npn \quark_if_recursion_tail_stop:N #1
2411 {
2412   \if_meaning:w #1 \q_recursion_tail
2413   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2414   \fi:
2415 }
2416 \cs_new:Npn \quark_if_recursion_tail_stop_do:Nn #1
2417 {
2418   \if_meaning:w #1 \q_recursion_tail
2419   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2420   \else:
2421     \exp_after:wN \use_none:n
2422   \fi:
2423 }
```

(End definition for `\quark_if_recursion_tail_stop:N`. This function is documented on page 45.)

`\quark_if_recursion_tail_stop:n` The same idea applies when testing multiple tokens, but here we just compare the token list to `\q_recursion_tail` as a string.

```
\quark_if_recursion_tail_stop:o
\quark_if_recursion_tail_stop_do:nn
\quark_if_recursion_tail_stop_do:on
\quark_if_recursion_tail_aux:w
2424 \cs_new:Npn \quark_if_recursion_tail_stop:n #1
2425 {
2426   \if_int_compare:w \pdfTeX_strcmp:D
2427     { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2428   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2429   \fi:
2430 }
2431 \cs_new:Npn \quark_if_recursion_tail_stop_do:nn #1
2432 {
2433   \if_int_compare:w \pdfTeX_strcmp:D
2434     { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2435   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2436   \else:
2437     \exp_after:wN \use_none:n
2438   \fi:
```

```

2439 }
2440 \cs_generate_variant:Nn \quark_if_recursion_tail_stop:n { o }
2441 \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_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

```

2442 \prg_new_conditional:Nnn \quark_if_nil:N { p, T, F, TF }
2443 {
2444   \if_meaning:w \q_nil #1
2445   \prg_return_true:
2446   \else:
2447   \prg_return_false:
2448   \fi:
2449 }
2450 \prg_new_conditional:Nnn \quark_if_no_value:N { p, T, F, TF }
2451 {
2452   \if_meaning:w \q_no_value #1
2453   \prg_return_true:
2454   \else:
2455   \prg_return_false:
2456   \fi:
2457 }
2458 \cs_generate_variant:Nn \quark_if_no_value_p:N { c }
2459 \cs_generate_variant:Nn \quark_if_no_value:NT { c }
2460 \cs_generate_variant:Nn \quark_if_no_value:NF { c }
2461 \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

```

2462 \prg_new_conditional:Nnn \quark_if_nil:n { p, T, F, TF }
2463 {
2464   \if_int_compare:w \pdfTeX_strcmp:D
2465   { \exp_not:N \q_nil } { \exp_not:n {#1} } = \c_zero
2466   \prg_return_true:
2467   \else:
2468   \prg_return_false:
2469   \fi:
2470 }
2471 \prg_new_conditional:Nnn \quark_if_no_value:n { p, T, F, TF }
2472 {
2473   \if_int_compare:w \pdfTeX_strcmp:D
2474   { \exp_not:N \q_no_value } { \exp_not:n {#1} } = \c_zero
2475   \prg_return_true:
2476   \else:

```

⁵It may still loop in special circumstances however!

```

2477     \prg_return_false:
2478     \fi:
2479   }
2480   \cs_generate_variant:Nn \quark_if_nil_p:n { V , o }
2481   \cs_generate_variant:Nn \quark_if_nil:nTF { V , o }
2482   \cs_generate_variant:Nn \quark_if_nil:nT { V , o }
2483   \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.

```

2484   \quark_new:N \q_tl_act_mark
2485   \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 95.)

```

2486 \</initex | package>

```

185 l3token implementation

```

2487 <*initex | package>
2488 <*package>
2489 \ProvidesExplPackage
2490   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2491   \package_check_loaded_expl:
2492 </package>

```

185.1 Character tokens

\char_set_catcode:nn Category code changes.

```

\char_value_catcode:n
\char_show_value_catcode:n

```

```

2493 \cs_new_protected_nopar:Npn \char_set_catcode:nn #1#2
2494   { \tex_catcode:D #1 = \int_eval:w #2 \int_eval_end: }
2495 \cs_new_nopar:Npn \char_value_catcode:n #1
2496   { \tex_the:D \tex_catcode:D \int_eval:w #1 \int_eval_end: }
2497 \cs_new_nopar:Npn \char_show_value_catcode:n #1
2498   { \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 48.)

```

\char_set_catcode_escape:N
\char_set_catcode_group_begin:N
\char_set_catcode_group_end:N
\char_set_catcode_math_toggle:N
\char_set_catcode_alignment:N
\char_set_catcode_end_line:N
\char_set_catcode_parameter:N
\char_set_catcode_math_superscript:N
\char_set_catcode_math_subscript:N
\char_set_catcode_ignore:N
\char_set_catcode_space:N
\char_set_catcode_letter:N
\char_set_catcode_other:N
\char_set_catcode_active:N
\char_set_catcode_comment:N
\char_set_catcode_invalid:N

```

```

2499 \cs_new_protected_nopar:Npn \char_set_catcode_escape:N #1
2500   { \char_set_catcode:nn { '#1 } \c_zero }
2501 \cs_new_protected_nopar:Npn \char_set_catcode_group_begin:N #1
2502   { \char_set_catcode:nn { '#1 } \c_one }
2503 \cs_new_protected_nopar:Npn \char_set_catcode_group_end:N #1
2504   { \char_set_catcode:nn { '#1 } \c_two }
2505 \cs_new_protected_nopar:Npn \char_set_catcode_math_toggle:N #1
2506   { \char_set_catcode:nn { '#1 } \c_three }
2507 \cs_new_protected_nopar:Npn \char_set_catcode_alignment:N #1

```

```

2508 { \char_set_catcode:nn { '#1 } \c_four }
2509 \cs_new_protected_nopar:Npn \char_set_catcode_end_line:N #1
2510 { \char_set_catcode:nn { '#1 } \c_five }
2511 \cs_new_protected_nopar:Npn \char_set_catcode_parameter:N #1
2512 { \char_set_catcode:nn { '#1 } \c_six }
2513 \cs_new_protected_nopar:Npn \char_set_catcode_math_superscript:N #1
2514 { \char_set_catcode:nn { '#1 } \c_seven }
2515 \cs_new_protected_nopar:Npn \char_set_catcode_math_subscript:N #1
2516 { \char_set_catcode:nn { '#1 } \c_eight }
2517 \cs_new_protected_nopar:Npn \char_set_catcode_ignore:N #1
2518 { \char_set_catcode:nn { '#1 } \c_nine }
2519 \cs_new_protected_nopar:Npn \char_set_catcode_space:N #1
2520 { \char_set_catcode:nn { '#1 } \c_ten }
2521 \cs_new_protected_nopar:Npn \char_set_catcode_letter:N #1
2522 { \char_set_catcode:nn { '#1 } \c_eleven }
2523 \cs_new_protected_nopar:Npn \char_set_catcode_other:N #1
2524 { \char_set_catcode:nn { '#1 } \c_twelve }
2525 \cs_new_protected_nopar:Npn \char_set_catcode_active:N #1
2526 { \char_set_catcode:nn { '#1 } \c_thirteen }
2527 \cs_new_protected_nopar:Npn \char_set_catcode_comment:N #1
2528 { \char_set_catcode:nn { '#1 } \c_fourteen }
2529 \cs_new_protected_nopar:Npn \char_set_catcode_invalid:N #1
2530 { \char_set_catcode:nn { '#1 } \c_fifteen }

```

(End definition for \char_set_catcode_escape:N and others. These functions are documented on page 47.)

```

\char_set_catcode_escape:n
  \char_set_catcode_group_begin:n 2531 \cs_new_protected_nopar:Npn \char_set_catcode_escape:n #1
  \char_set_catcode_group_end:n    2532 { \char_set_catcode:nn {#1} \c_zero }
  \char_set_catcode_math_toggle:n  2533 \cs_new_protected_nopar:Npn \char_set_catcode_group_begin:n #1
  \char_set_catcode_alignment:n     2534 { \char_set_catcode:nn {#1} \c_one }
\char_set_catcode_end_line:n        2535 \cs_new_protected_nopar:Npn \char_set_catcode_group_end:n #1
  \char_set_catcode_parameter:n     2536 { \char_set_catcode:nn {#1} \c_two }
  \char_set_catcode_math_superscript:n 2537 \cs_new_protected_nopar:Npn \char_set_catcode_math_toggle:n #1
  \char_set_catcode_math_subscript:n 2538 { \char_set_catcode:nn {#1} \c_three }
  \char_set_catcode_ignore:n         2539 \cs_new_protected_nopar:Npn \char_set_catcode_alignment:n #1
  \char_set_catcode_space:n          2540 { \char_set_catcode:nn {#1} \c_four }
  \char_set_catcode_letter:n         2541 \cs_new_protected_nopar:Npn \char_set_catcode_end_line:n #1
  \char_set_catcode_other:n          2542 { \char_set_catcode:nn {#1} \c_five }
  \char_set_catcode_active:n         2543 \cs_new_protected_nopar:Npn \char_set_catcode_parameter:n #1
  \char_set_catcode_comment:n        2544 { \char_set_catcode:nn {#1} \c_six }
  \char_set_catcode_invalid:n        2545 \cs_new_protected_nopar:Npn \char_set_catcode_math_superscript:n #1
  \char_set_catcode_invalid:n        2546 { \char_set_catcode:nn {#1} \c_seven }
  \char_set_catcode_invalid:n        2547 \cs_new_protected_nopar:Npn \char_set_catcode_math_subscript:n #1
  \char_set_catcode_invalid:n        2548 { \char_set_catcode:nn {#1} \c_eight }
  \char_set_catcode_invalid:n        2549 \cs_new_protected_nopar:Npn \char_set_catcode_ignore:n #1
  \char_set_catcode_invalid:n        2550 { \char_set_catcode:nn {#1} \c_nine }
  \char_set_catcode_invalid:n        2551 \cs_new_protected_nopar:Npn \char_set_catcode_space:n #1
  \char_set_catcode_invalid:n        2552 { \char_set_catcode:nn {#1} \c_ten }
  \char_set_catcode_invalid:n        2553 \cs_new_protected_nopar:Npn \char_set_catcode_letter:n #1

```

```

2554 { \char_set_catcode:nn {#1} \c_eleven }
2555 \cs_new_protected_nopar:Npn \char_set_catcode_other:n #1
2556 { \char_set_catcode:nn {#1} \c_twelve }
2557 \cs_new_protected_nopar:Npn \char_set_catcode_active:n #1
2558 { \char_set_catcode:nn {#1} \c_thirteen }
2559 \cs_new_protected_nopar:Npn \char_set_catcode_comment:n #1
2560 { \char_set_catcode:nn {#1} \c_fourteen }
2561 \cs_new_protected_nopar:Npn \char_set_catcode_invalid:n #1
2562 { \char_set_catcode:nn {#1} \c_fifteen }

```

(End definition for `\char_set_catcode_escape:n` and others. These functions are documented on page 47.)

```

\char_set_mathcode:nn Pretty repetitive, but necessary!
\char_value_mathcode:n 2563 \cs_new_protected_nopar:Npn \char_set_mathcode:nn #1#2
\char_show_value_mathcode:n 2564 { \tex_mathcode:D #1 = \int_eval:w #2 \int_eval_end: }
\char_set_lccode:nn 2565 \cs_new_nopar:Npn \char_value_mathcode:n #1
\char_value_lccode:n 2566 { \tex_the:D \tex_mathcode:D \int_eval:w #1\int_eval_end: }
\char_show_value_lccode:n 2567 \cs_new_nopar:Npn \char_show_value_mathcode:n #1
\char_set_uccode:nn 2568 { \tex_showthe:D \tex_mathcode:D \int_eval:w #1 \int_eval_end: }
\char_value_uccode:n 2569 \cs_new_protected_nopar:Npn \char_set_lccode:nn #1#2
\char_show_value_uccode:n 2570 { \tex_lccode:D #1 = \int_eval:w #2 \int_eval_end: }
\char_set_sfcode:nn 2571 \cs_new_nopar:Npn \char_value_lccode:n #1
\char_value_sfcode:n 2572 { \tex_the:D \tex_lccode:D \int_eval:w #1\int_eval_end: }
\char_show_value_sfcode:n 2573 \cs_new_nopar:Npn \char_show_value_lccode:n #1
2574 { \tex_showthe:D \tex_lccode:D \int_eval:w #1 \int_eval_end: }
2575 \cs_new_protected_nopar:Npn \char_set_uccode:nn #1#2
2576 { \tex_uccode:D #1 = \int_eval:w #2 \int_eval_end: }
2577 \cs_new_nopar:Npn \char_value_uccode:n #1
2578 { \tex_the:D \tex_uccode:D \int_eval:w #1\int_eval_end: }
2579 \cs_new_nopar:Npn \char_show_value_uccode:n #1
2580 { \tex_showthe:D \tex_uccode:D \int_eval:w #1 \int_eval_end: }
2581 \cs_new_protected_nopar:Npn \char_set_sfcode:nn #1#2
2582 { \tex_sfcode:D #1 = \int_eval:w #2 \int_eval_end: }
2583 \cs_new_nopar:Npn \char_value_sfcode:n #1
2584 { \tex_the:D \tex_sfcode:D \int_eval:w #1\int_eval_end: }
2585 \cs_new_nopar:Npn \char_show_value_sfcode:n #1
2586 { \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 50.)

185.2 Generic tokens

`\token_new:Nn` Creates a new token.

```

2587 \cs_new_protected_nopar:Npn \token_new:Nn #1#2 { \cs_new_eq:NN #1 #2 }

```

(End definition for `\token_new:Nn`. This function is documented on page 50.)

`\c_group_begin_token` We define these useful tokens. We have to do it by hand with the brace tokens for obvious reasons.

```

\c_group_end_token
\c_math_toggle_token 2588 \cs_new_eq:NN \c_group_begin_token {
\c_alignment_token 2589 \cs_new_eq:NN \c_group_end_token }
\c_parameter_token

```

```

\c_math_superscript_token
\c_math_subscript_token
\c_space_token
\c_catcode_letter_token
\c_catcode_other_token

```

```

2590 \group_begin:
2591   \char_set_catcode_math_toggle:N \*
2592   \token_new:Nn \c_math_toggle_token { * }
2593   \char_set_catcode_alignment:N \*
2594   \token_new:Nn \c_alignment_token { * }
2595   \token_new:Nn \c_parameter_token { # }
2596   \token_new:Nn \c_math_superscript_token { ^ }
2597   \char_set_catcode_math_subscript:N \*
2598   \token_new:Nn \c_math_subscript_token { * }
2599   \token_new:Nn \c_space_token { ~ }
2600   \token_new:Nn \c_catcode_letter_token { a }
2601   \token_new:Nn \c_catcode_other_token { 1 }
2602 \group_end:

```

(End definition for `\c_group_begin_token` and others. These functions are documented on page 50.)

`\c_catcode_active_tl` Not an implicit token!

```

2603 \group_begin:
2604   \char_set_catcode_active:N \*
2605   \tl_const:Nn \c_catcode_active_tl { \exp_not:N * }
2606 \group_end:

```

(End definition for `\c_catcode_active_tl`. This function is documented on page 50.)

185.3 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.

```

2607 \prg_new_conditional:Npnn \token_if_group_begin:N #1 { p , T , F , TF }
2608 {
2609   \if_catcode:w \exp_not:N #1 \c_group_begin_token
2610   \prg_return_true: \else: \prg_return_false: \fi:
2611 }

```

(End definition for `\token_if_group_begin:N`. This function is documented on page 51.)

`\token_if_group_end:N` Check if token is a end group token. We use the constant `\c_group_end_token` for this.

```

2612 \prg_new_conditional:Npnn \token_if_group_end:N #1 { p , T , F , TF }
2613 {
2614   \if_catcode:w \exp_not:N #1 \c_group_end_token
2615   \prg_return_true: \else: \prg_return_false: \fi:
2616 }

```

(End definition for `\token_if_group_end:N`. This function is documented on page 51.)

`\token_if_math_toggle:N` Check if token is a math shift token. We use the constant `\c_math_toggle_token` for this.

```

2617 \prg_new_conditional:Npnn \token_if_math_toggle:N #1 { p , T , F , TF }
2618 {
2619   \if_catcode:w \exp_not:N #1 \c_math_toggle_token
2620   \prg_return_true: \else: \prg_return_false: \fi:
2621 }

```


(End definition for \token_if_math_toggle:N. This function is documented on page 51.)

`\token_if_alignment:N` Check if token is an alignment tab token. We use the constant `\c_alignment_tab_token` for this.

```
2622 \prg_new_conditional:Npnn \token_if_alignment:N #1 { p , T , F , TF }
2623 {
2624   \if_catcode:w \exp_not:N #1 \c_alignment_token
2625   \prg_return_true: \else: \prg_return_false: \fi:
2626 }
```

(End definition for \token_if_alignment:N. This function is documented on page 51.)

`\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.

```
2627 \group_begin:
2628 \cs_set_eq:NN \c_parameter_token \scan_stop:
2629 \prg_new_conditional:Npnn \token_if_parameter:N #1 { p , T , F , TF }
2630 {
2631   \if_catcode:w \exp_not:N #1 \c_parameter_token
2632   \prg_return_true: \else: \prg_return_false: \fi:
2633 }
2634 \group_end:
```

(End definition for \token_if_parameter:N. This function is documented on page 51.)

`\token_if_math_superscript:N` Check if token is a math superscript token. We use the constant `\c_superscript_token` for this.

```
2635 \prg_new_conditional:Npnn \token_if_math_superscript:N #1 { p , T , F , TF }
2636 {
2637   \if_catcode:w \exp_not:N #1 \c_math_superscript_token
2638   \prg_return_true: \else: \prg_return_false: \fi:
2639 }
```

(End definition for \token_if_math_superscript:N. This function is documented on page 51.)

`\token_if_math_subscript:N` Check if token is a math subscript token. We use the constant `\c_subscript_token` for this.

```
2640 \prg_new_conditional:Npnn \token_if_math_subscript:N #1 { p , T , F , TF }
2641 {
2642   \if_catcode:w \exp_not:N #1 \c_math_subscript_token
2643   \prg_return_true: \else: \prg_return_false: \fi:
2644 }
```

(End definition for \token_if_math_subscript:N. This function is documented on page 52.)

`\token_if_space:N` Check if token is a space token. We use the constant `\c_space_token` for this.

```
2645 \prg_new_conditional:Npnn \token_if_space:N #1 { p , T , F , TF }
2646 {
2647   \if_catcode:w \exp_not:N #1 \c_space_token
2648   \prg_return_true: \else: \prg_return_false: \fi:
2649 }
```

(End definition for \token_if_space:N. This function is documented on page 52.)

`\token_if_letter:N` Check if token is a letter token. We use the constant `\c_letter_token` for this.

```
2650 \prg_new_conditional:Npnn \token_if_letter:N #1 { p , T , F , TF }
2651 {
2652   \if_catcode:w \exp_not:N #1 \c_catcode_letter_token
2653   \prg_return_true: \else: \prg_return_false: \fi:
2654 }
```

(End definition for \token_if_letter:N. This function is documented on page 52.)

`\token_if_other:N` Check if token is an other char token. We use the constant `\c_other_char_token` for this.

```
2655 \prg_new_conditional:Npnn \token_if_other:N #1 { p , T , F , TF }
2656 {
2657   \if_catcode:w \exp_not:N #1 \c_catcode_other_token
2658   \prg_return_true: \else: \prg_return_false: \fi:
2659 }
```

(End definition for \token_if_other:N. This function is documented on page 52.)

`\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.

```
2660 \prg_new_conditional:Npnn \token_if_active:N #1 { p , T , F , TF }
2661 {
2662   \if_catcode:w \exp_not:N #1 \c_catcode_active_tl
2663   \prg_return_true: \else: \prg_return_false: \fi:
2664 }
```

(End definition for \token_if_active:N. This function is documented on page 52.)

`\token_if_eq_meaning:NN` Check if the tokens #1 and #2 have same meaning.

```
2665 \prg_new_conditional:Npnn \token_if_eq_meaning:NN #1#2 { p , T , F , TF }
2666 {
2667   \if_meaning:w #1 #2
2668   \prg_return_true: \else: \prg_return_false: \fi:
2669 }
```

(End definition for \token_if_eq_meaning:NN. This function is documented on page 52.)

`\token_if_eq_catcode:NN` Check if the tokens #1 and #2 have same category code.

```
2670 \prg_new_conditional:Npnn \token_if_eq_catcode:NN #1#2 { p , T , F , TF }
2671 {
2672   \if_catcode:w \exp_not:N #1 \exp_not:N #2
2673   \prg_return_true: \else: \prg_return_false: \fi:
2674 }
```

(End definition for \token_if_eq_catcode:NN. This function is documented on page 52.)

`\token_if_eq_charcode:NN` Check if the tokens #1 and #2 have same character code.

```

2675 \prg_new_conditional:Npnn \token_if_eq_charcode:NN #1#2 { p , T , F , TF }
2676 {
2677   \if_charcode:w \exp_not:N #1 \exp_not:N #2
2678   \prg_return_true: \else: \prg_return_false: \fi:
2679 }

```

(End definition for `\token_if_eq_charcode:NN`. This function is documented on page 52.)

`\token_if_macro:N` When a token is a macro, `\token_to_meaning:N` will always output something like
`\token_if_macro_p_aux:w` `\long macro:#1->#1` so we could naively check to see if the meaning contains `->`.
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.

```

2680 \group_begin:
2681 \char_set_catcode_other:N \M
2682 \char_set_catcode_other:N \A
2683 \char_set_lccode:nn { '\; } { '\: }
2684 \char_set_lccode:nn { '\T } { '\T }
2685 \char_set_lccode:nn { '\F } { '\F }
2686 \tl_to_lowercase:n
2687 {
2688   \group_end:
2689   \prg_new_conditional:Npnn \token_if_macro:N #1 { p , T , F , TF }
2690   {
2691     \exp_after:wN \token_if_macro_p_aux:w
2692     \token_to_meaning:N #1 MA; \q_stop
2693   }
2694   \cs_new_nopar:Npn \token_if_macro_p_aux:w #1 MA #2 ; #3 \q_stop
2695   {
2696     \if_int_compare:w \pdfTeX_strcmp:D { #2 } { cro } = \c_zero
2697     \prg_return_true:
2698   \else:
2699     \prg_return_false:
2700   \fi:
2701   }
2702 }

```

(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.

```

2703 \prg_new_conditional:Npnn \token_if_cs:N #1 { p , T , F , TF }
2704 {
2705     \if_catcode:w \exp_not:N #1 \scan_stop:
2706     \prg_return_true: \else: \prg_return_false: \fi:
2707 }

```

(End definition for `\token_if_cs:N`. This function is documented on page 52.)

`\token_if_expandable:N` Check if token is expandable. We use the fact that T_EX will temporarily convert `\exp_not:N` $\langle token \rangle$ into `\scan_stop:` if $\langle token \rangle$ is expandable.

```

2708 \prg_new_conditional:Npnn \token_if_expandable:N #1 { p , T , F , TF }
2709 {
2710     \cs_if_exist:NTF #1
2711     {
2712         \exp_after:wN \if_meaning:w \exp_not:N #1 #1
2713         \prg_return_false: \else: \prg_return_true: \fi:
2714     }
2715     { \prg_return_false: }
2716 }

```

(End definition for `\token_if_expandable:N`. This function is documented on page 53.)

`\token_if_chardef:N` Most of these functions have to check the meaning of the token in question so we need to do some checkups on which characters are output by `\token_to_meaning:N`. As usual, these characters have catcode 12 so we must do some serious substitutions in the code below...

```

\token_if_protected_macro:N
\token_if_dim_register:N
\token_if_skip_register:N
\token_if_int_register:N
\token_if_toks_register:N
\token_if_chardef_p_aux:w
\token_if_mathchardef_p_aux:w
\token_if_int_register_p_aux:w
\token_if_skip_register_p_aux:w
\token_if_dim_register_p_aux:w
\token_if_toks_register_p_aux:w
\token_if_protected_macro_p_aux:w
\token_if_long_macro_p_aux:w
\token_if_protected_long_macro_p_aux:w
2717 \group_begin:
2718 \char_set_lccode:nn { '\T } { '\T }
2719 \char_set_lccode:nn { '\F } { '\F }
2720 \char_set_lccode:nn { '\X } { '\n }
2721 \char_set_lccode:nn { '\Y } { '\t }
2722 \char_set_lccode:nn { '\Z } { '\d }
2723 \char_set_lccode:nn { '\? } { '\\ }
2724 \tl_map_inline:nn { \X \Y \Z \M \C \H \A \R \O \U \S \K \I \P \L \G \P \E }
2725 { \char_set_catcode:nn { '#1 } \c_twelve }

```

We convert the token list to lower case and restore the catcode and lowercase code changes.

```

2726 \tl_to_lowercase:n
2727 {
2728     \group_end:

```

First up is checking if something has been defined with `\chardef` or `\mathchardef`. This is easy since T_EX thinks of such tokens as hexadecimal so it stores them as `\char"⟨hex number⟩` or `\mathchar"⟨hex number⟩`.

```

2729 \prg_new_conditional:Npnn \token_if_chardef:N #1 { p , T , F , TF }
2730 {
2731     \exp_after:wN \token_if_chardef_aux:w
2732     \token_to_meaning:N #1 ?CHAR" \q_stop

```

```

2733     }
2734     \cs_new_nopar:Npn \token_if_chardef_aux:w #1 ?CHAR" #2 \q_stop
2735     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
2736     \prg_new_conditional:Npnn \token_if_mathchardef:N #1 { p , T , F , TF }
2737     {
2738         \exp_after:wN \token_if_mathchardef_aux:w
2739         \token_to_meaning:N #1 ?MAYHCHAR" \q_stop
2740     }
2741     \cs_new_nopar:Npn \token_if_mathchardef_aux:w #1 ?MAYHCHAR" #2 \q_stop
2742     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Integer registers are a little more difficult since they expand to `\count⟨number⟩` and there is also a primitive `\countdef`. So we have to check for that primitive as well.

```

2743     \prg_new_conditional:Npnn \token_if_int_register:N #1 { p , T , F , TF }
2744     {
2745         \if_meaning:w \tex_countdef:D #1
2746         \prg_return_false:
2747     \else:
2748         \exp_after:wN \token_if_int_register_aux:w
2749         \token_to_meaning:N #1 ?COUXY \q_stop
2750     \fi:
2751     }
2752     \cs_new_nopar:Npn \token_if_int_register_aux:w #1 ?COUXY #2 \q_stop
2753     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Skip registers are done the same way as the integer registers.

```

2754     \prg_new_conditional:Npnn \token_if_skip_register:N #1 { p , T , F , TF }
2755     {
2756         \if_meaning:w \tex_skipdef:D #1
2757         \prg_return_false:
2758     \else:
2759         \exp_after:wN \token_if_skip_register_aux:w
2760         \token_to_meaning:N #1 ?SKIP \q_stop
2761     \fi:
2762     }
2763     \cs_new_nopar:Npn \token_if_skip_register_aux:w #1 ?SKIP #2 \q_stop
2764     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Dim registers. No news here

```

2765     \prg_new_conditional:Npnn \token_if_dim_register:N #1 { p , T , F , TF }
2766     {
2767         \if_meaning:w \tex_dimendef:D #1
2768         \c_false_bool
2769     \else:
2770         \exp_after:wN \token_if_dim_register_aux:w
2771         \token_to_meaning:N #1 ?ZIMEX \q_stop
2772     \fi:
2773     }
2774     \cs_new_nopar:Npn \token_if_dim_register_aux:w #1 ?ZIMEX #2 \q_stop
2775     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Toks registers.

```

2776 \prg_new_conditional:Npnn \token_if_toks_register:N #1 { p , T , F , TF }
2777 {
2778   \if_meaning:w \tex_toksdef:D #1
2779   \prg_return_false:
2780   \else:
2781     \exp_after:wN \token_if_toks_register_aux:w
2782     \token_to_meaning:N #1 ?YOKS \q_stop
2783   \fi:
2784 }
2785 \cs_new_nopar:Npn \token_if_toks_register_aux:w #1 ?YOKS #2 \q_stop
2786 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Protected macros.

```

2787 \prg_new_conditional:Npnn \token_if_protected_macro:N #1
2788 { p , T , F , TF }
2789 {
2790   \exp_after:wN \token_if_protected_macro_aux:w
2791   \token_to_meaning:N #1 ?PROYECYEZ~MACRO \q_stop
2792 }
2793 \cs_new_nopar:Npn \token_if_protected_macro_aux:w
2794 #1 ?PROYECYEZ~MACRO #2 \q_stop
2795 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Long macros.

```

2796 \prg_new_conditional:Npnn \token_if_long_macro:N #1 { p , T , F , TF }
2797 {
2798   \exp_after:wN \token_if_long_macro_aux:w
2799   \token_to_meaning:N #1 ?LOXG~MACRO \q_stop
2800 }
2801 \cs_new_nopar:Npn \token_if_long_macro_aux:w #1 ?LOXG~MACRO #2 \q_stop
2802 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Finally protected long macros where we for once don't have to add an extra test since there is no primitive for the combined prefixes.

```

2803 \prg_new_conditional:Npnn \token_if_protected_long_macro:N #1
2804 { p , T , F , TF }
2805 {
2806   \exp_after:wN \token_if_protected_long_macro_aux:w
2807   \token_to_meaning:N #1 ?PROYECYEZ?LOXG~MACRO \q_stop
2808 }
2809 \cs_new_nopar:Npn \token_if_protected_long_macro_aux:w
2810 #1 ?PROYECYEZ?LOXG~MACRO #2 \q_stop
2811 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Finally the `\tl_to_lowercase:n` ends!

```

2812 }

```

(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.

```

2813 \tex_chardef:D \c_token_A_int = 'A ~ %
2814 \group_begin:
2815 \char_set_catcode_other:N \;
2816 \char_set_lccode:nn { '\; } { '\: }
2817 \char_set_lccode:nn { '\T } { '\T }
2818 \char_set_lccode:nn { '\F } { '\F }
2819 \tl_to_lowercase:n {
2820   \group_end:
2821   \prg_new_conditional:Npnn \token_if_primitive:N #1 { p , T , F , TF }
2822   {
2823     \token_if_macro:NTF #1
2824     \prg_return_false:
2825     {
2826       \exp_after:wN \token_if_primitive_aux:NNw
2827       \token_to_meaning:N #1 ; ; ; \q_stop #1
2828     }
2829   }
2830 \cs_new_nopar:Npn \token_if_primitive_aux:NNw #1#2 #3 ; #4 \q_stop
2831 {
2832   \tl_if_empty:oTF { \token_if_primitive_aux_space:w #3 ~ }
2833   { \token_if_primitive_aux_loop:N #3 ; \q_stop }
2834   { \token_if_primitive_aux_nullfont:N }
2835 }
2836 }

```

```

2837 \cs_new_nopar:Npn \token_if_primitive_aux_space:w #1 ~ { }
2838 \cs_new:Npn \token_if_primitive_aux_nullfont:N #1
2839 {
2840   \if_meaning:w \tex_nullfont:D #1
2841     \prg_return_true:
2842   \else:
2843     \prg_return_false:
2844   \fi:
2845 }
2846 \cs_new_nopar:Npn \token_if_primitive_aux_loop:N #1
2847 {
2848   \if_num:w '#1 < \c_token_A_int %
2849     \exp_after:wN \token_if_primitive_auxii:Nw
2850     \exp_after:wN #1
2851   \else:
2852     \exp_after:wN \token_if_primitive_aux_loop:N
2853   \fi:
2854 }
2855 \cs_new_nopar:Npn \token_if_primitive_auxii:Nw #1 #2 \q_stop
2856 {
2857   \if:w : #1
2858     \exp_after:wN \token_if_primitive_aux_undefined:N
2859   \else:
2860     \prg_return_false:
2861     \exp_after:wN \use_none:n
2862   \fi:
2863 }
2864 \cs_new:Npn \token_if_primitive_aux_undefined:N #1
2865 {
2866   \if_cs_exist:N #1
2867     \prg_return_true:
2868   \else:
2869     \prg_return_false:
2870   \fi:
2871 }

```

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

185.4 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` 2872 `\cs_new_eq:NN \l_peek_token ?`
2873 `\cs_new_eq:NN \g_peek_token ?`
(End definition for \l_peek_token. This function is documented on page 54.)

`\l_peek_search_token` The token to search for as an implicit token: cf. `\l_peek_search_tl`.
2874 `\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`.
2875 `\cs_new_nopar:Npn \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` 2876 `\cs_new_nopar:Npn \peek_true:w { }`

`\peek_false:w` 2877 `\cs_new_nopar:Npn \peek_true_aux:w { }`

`\peek_tmp:w` 2878 `\cs_new_nopar:Npn \peek_false:w { }`
2879 `\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` 2880 `\cs_new_protected_nopar:Npn \peek_after:Nw`
2881 `{ \tex_futurelet:D \l_peek_token }`
2882 `\cs_new_protected_nopar:Npn \peek_gafter:Nw`
2883 `{ \tex_global:D \tex_futurelet:D \g_peek_token }`
(End definition for \peek_after:Nw. This function is documented on page 54.)

`\peek_true_remove:w` A function to remove the next token and then regain control.
2884 `\cs_new_protected:Npn \peek_true_remove:w`
2885 `{`
2886 `\group_align_safe_end:`
2887 `\tex_afterassignment:D \peek_true_aux:w`
2888 `\cs_set_eq:NN \peek_tmp:w`
2889 `}`
(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.
2890 `\cs_new_protected:Npn \peek_token_generic:NTTF #1#2#3#4`
2891 `{`
2892 `\cs_set_eq:NN \l_peek_search_token #2`
2893 `\tl_set:Nn \l_peek_search_tl {#2}`
2894 `\cs_set_nopar:Npx \peek_true:w`
2895 `{`
2896 `\exp_not:N \group_align_safe_end:`
2897 `\exp_not:n {#3}`
2898 `}`

```

2899     \cs_set_nopar:Npx \peek_false:w
2900     {
2901         \exp_not:N \group_align_safe_end:
2902         \exp_not:n {#4}
2903     }
2904     \group_align_safe_begin:
2905     \peek_after:Nw #1
2906 }
2907 \cs_new_protected:Npn \peek_token_generic:NNT #1#2#3
2908 { \peek_token_generic:NNTF #1 #2 {#3} { } }
2909 \cs_new_protected:Npn \peek_token_generic:NNF #1#2#3
2910 { \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.

```

2911 \cs_new_protected:Npn \peek_token_remove_generic:NNTF #1#2#3#4
2912 {
2913     \cs_set_eq:NN \l_peek_search_token #2
2914     \tl_set:Nn \l_peek_search_tl {#2}
2915     \cs_set_eq:NN \peek_true:w \peek_true_remove:w
2916     \cs_set_nopar:Npx \peek_true_aux:w { \exp_not:n {#3} }
2917     \cs_set_nopar:Npx \peek_false:w
2918     {
2919         \exp_not:N \group_align_safe_end:
2920         \exp_not:n {#4}
2921     }
2922     \group_align_safe_begin:
2923     \peek_after:Nw #1
2924 }
2925 \cs_new_protected:Npn \peek_token_remove_generic:NNT #1#2#3
2926 { \peek_token_remove_generic:NNTF #1 #2 {#3} { } }
2927 \cs_new_protected:Npn \peek_token_remove_generic:NNF #1#2#3
2928 { \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:
2929 \cs_new_nopar:Npn \peek_execute_branches_catcode:
2930 {
2931     \if_catcode:w
2932         \exp_not:N \l_peek_token \exp_not:N \l_peek_search_token
2933         \exp_after:wN \peek_true:w
2934     \else:
2935         \exp_after:wN \peek_false:w
2936     \fi:
2937 }
2938 \cs_new_nopar:Npn \peek_execute_branches_meaning:
2939 {
2940     \if_meaning:w \l_peek_token \l_peek_search_token
2941         \exp_after:wN \peek_true:w

```

```

2942     \else:
2943       \exp_after:wN \peek_false:w
2944     \fi:
2945   }

```

*(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 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.

```

2946 \cs_new_nopar:Npn \peek_execute_branches_charcode:
2947 {
2948   \bool_if:nTF
2949   {
2950     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token
2951     || \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token
2952     || \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
2953   }
2954   { \peek_false:w }
2955   {
2956     \exp_after:wN \peek_execute_branches_charcode_aux:NN
2957     \l_peek_search_tl
2958   }
2959 }
2960 \cs_new:Npn \peek_execute_branches_charcode_aux:NN #1#2
2961 {
2962   \if:w \exp_not:N #1 \exp_not:N #2
2963     \exp_after:wN \peek_true:w
2964   \else:
2965     \exp_after:wN \peek_false:w
2966   \fi:
2967   #2
2968 }

```

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

\peek_ignore_spaces_execute_branches: This function removes one token at a time with a mechanism that can be applied to things other than spaces.

```

2969 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches:
2970 {
2971   \token_if_eq_meaning:NNTF \l_peek_token \c_space_token
2972   {
2973     \tex_afterassignment:D \peek_ignore_spaces_execute_branches_aux:
2974     \cs_set_eq:NN \peek_tmp:w
2975   }
2976   { \peek_execute_branches: }
2977 }
2978 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches_aux:
2979 { \peek_after:Nw \peek_ignore_spaces_execute_branches: }

```

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

`\peek_def:nnnn` The public functions themselves cannot be defined using `\prg_set_conditional:Npn` 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.

```

2980 \group_begin:
2981   \cs_set_nopar:Npn \peek_def:nnnn #1#2#3#4
2982   {
2983     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { TF }
2984     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { T }
2985     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { F }
2986   }
2987   \cs_set_nopar:Npn \peek_def_aux:nnnnn #1#2#3#4#5
2988   {
2989     \cs_gset_nopar:cpx { #1 #5 }
2990     {
2991       \tl_if_empty:nF {#2}
2992       { \exp_not:n { \cs_set_eq:NN \peek_execute_branches: #2 } }
2993       \exp_not:c { #3 #5 }
2994       \exp_not:n {#4}
2995     }
2996   }

```

(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 2997 \peek_def:nnnn { peek_catcode:N }
\peek_catcode_remove:N         2998 { }
\peek_catcode_remove_ignore_spaces:N 2999 { peek_token_generic:NN }
                                3000 { \peek_execute_branches_catcode: }
                                3001 \peek_def:nnnn { peek_catcode_ignore_spaces:N }
                                3002 { \peek_execute_branches_catcode: }
                                3003 { peek_token_generic:NN }
                                3004 { \peek_ignore_spaces_execute_branches: }
                                3005 \peek_def:nnnn { peek_catcode_remove:N }
                                3006 { }
                                3007 { peek_token_remove_generic:NN }
                                3008 { \peek_execute_branches_catcode: }
                                3009 \peek_def:nnnn { peek_catcode_remove_ignore_spaces:N }
                                3010 { \peek_execute_branches_catcode: }
                                3011 { peek_token_remove_generic:NN }
                                3012 { \peek_ignore_spaces_execute_branches: }

```

(End definition for `\peek_catcode:N` and others. These functions are documented on page 55.)

`\peek_charcode:N` Then for character codes.

```

\peek_charcode_ignore_spaces:N 3013 \peek_def:nnnn { peek_charcode:N }
\peek_charcode_remove:N         3014 { }
\peek_charcode_remove_ignore_spaces:N 3015 { peek_token_generic:NN }
                                3016 { \peek_execute_branches_charcode: }

```

```

3017 \peek_def:nnnn { peek_charcode_ignore_spaces:N }
3018 { \peek_execute_branches_charcode: }
3019 { peek_token_generic:NN }
3020 { \peek_ignore_spaces_execute_branches: }
3021 \peek_def:nnnn { peek_charcode_remove:N }
3022 { }
3023 { peek_token_remove_generic:NN }
3024 { \peek_execute_branches_charcode: }
3025 \peek_def:nnnn { peek_charcode_remove_ignore_spaces:N }
3026 { \peek_execute_branches_charcode: }
3027 { peek_token_remove_generic:NN }
3028 { \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_charcode:N and others. These functions are documented on page 55.)

```

\peek_meaning:N Finally for meaning, with the group closed to remove the temporary definition functions.
\peek_meaning_ignore_spaces:N 3029 \peek_def:nnnn { peek_meaning:N }
\peek_meaning_remove:N 3030 { }
\peek_meaning_remove_ignore_spaces:N 3031 { peek_token_generic:NN }
3032 { \peek_execute_branches_meaning: }
3033 \peek_def:nnnn { peek_meaning_ignore_spaces:N }
3034 { \peek_execute_branches_meaning: }
3035 { peek_token_generic:NN }
3036 { \peek_ignore_spaces_execute_branches: }
3037 \peek_def:nnnn { peek_meaning_remove:N }
3038 { }
3039 { peek_token_remove_generic:NN }
3040 { \peek_execute_branches_meaning: }
3041 \peek_def:nnnn { peek_meaning_remove_ignore_spaces:N }
3042 { \peek_execute_branches_meaning: }
3043 { peek_token_remove_generic:NN }
3044 { \peek_ignore_spaces_execute_branches: }
3045 \group_end:

```

(End definition for \peek_meaning:N and others. These functions are documented on page 56.)

185.5 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. However, we cannot just expand the macro blindly as it may have arguments and none might be present. Therefore we define these functions to pick either the prefix(es), the argument specification, or the replacement text from a macro. All of this information is returned as characters with catcode 12. If the token in question isn't a macro, the token `\scan_stop:` is returned instead.

```

3046 \exp_args:Nno \use:nn
3047 { \cs_new_nopar:Npn \token_get_prefix_arg_replacement_aux:wN #1 }
3048 { \tl_to_str:n { macro : } #2 -> #3 \q_stop #4 }
3049 { #4 {#1} {#2} {#3} }
3050 \cs_new:Npn \token_get_prefix_spec:N #1
3051 {

```

```

3052     \token_if_macro:NTF #1
3053     {
3054         \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3055         \token_to_meaning:N #1 \q_stop \use_i:nnn
3056     }
3057     { \scan_stop: }
3058 }
3059 \cs_new:Npn \token_get_arg_spec:N #1
3060 {
3061     \token_if_macro:NTF #1
3062     {
3063         \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3064         \token_to_meaning:N #1 \q_stop \use_ii:nnn
3065     }
3066     { \scan_stop: }
3067 }
3068 \cs_new:Npn \token_get_replacement_spec:N #1
3069 {
3070     \token_if_macro:NTF #1
3071     {
3072         \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3073         \token_to_meaning:N #1 \q_stop \use_iii:nnn
3074     }
3075     { \scan_stop: }
3076 }

```

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

185.6 Experimental token functions

```

\char_active_set:Npn
\char_active_set:Npx
\char_active_set:Npn
\char_active_set:Npx
\char_active_set_eq:NN
\char_active_gset_eq:NN
3077 \group_begin:
3078     \char_set_catcode_active:N ^^@
3079     \cs_set:Npn \char_tmp:NN #1#2
3080     {
3081         \cs_new:Npn #1 ##1
3082         {
3083             \char_set_catcode_active:n { '##1 }
3084             \group_begin:
3085             \char_set_lccode:nn { '\^^@ } { '##1 }
3086             \tl_to_lowercase:n { \group_end: #2 ^^@ }
3087         }
3088     }
3089     \char_tmp:NN \char_active_set:Npn     \cs_set:Npn
3090     \char_tmp:NN \char_active_set:Npx     \cs_set:Npx
3091     \char_tmp:NN \char_active_gset:Npn    \cs_gset:Npn
3092     \char_tmp:NN \char_active_gset:Npx    \cs_gset:Npx
3093     \char_tmp:NN \char_active_set_eq:NN    \cs_set_eq:NN
3094     \char_tmp:NN \char_active_gset_eq:NN \cs_gset_eq:NN
3095 \group_end:

```

(End definition for `\char_active_set:Npn` and `\char_active_set:Npx`. These functions are documented on page 58.)

`\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.

```

3096 \cs_new_protected_nopar:Npn \peek_execute_branches_N_type:
3097 {
3098   \bool_if:nTF
3099   {
3100     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
3101     \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token ||
3102     \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3103   }
3104   { \peek_false:w }
3105   { \peek_true:w }
3106 }
3107 \cs_new_protected_nopar:Npn \peek_N_type:TF
3108 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }
3109 \cs_new_protected_nopar:Npn \peek_N_type:T
3110 { \peek_token_generic:NNT \peek_execute_branches_N_type: \scan_stop: }
3111 \cs_new_protected_nopar:Npn \peek_N_type:F
3112 { \peek_token_generic:NNF \peek_execute_branches_N_type: \scan_stop: }

```

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

185.7 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\char_set_catcode:w` Primitives renamed.

```

\char_set_mathcode:w 3113 {*deprecated}
\char_set_lccode:w    3114 \cs_new_eq:NN \char_set_catcode:w \tex_catcode:D
\char_set_uccode:w    3115 \cs_new_eq:NN \char_set_mathcode:w \tex_mathcode:D
\char_set_sfcode:w    3116 \cs_new_eq:NN \char_set_lccode:w \tex_lccode:D
                     3117 \cs_new_eq:NN \char_set_uccode:w \tex_uccode:D
                     3118 \cs_new_eq:NN \char_set_sfcode:w \tex_sfcode:D
                     3119 {/deprecated}

```

(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 3120 {*deprecated}
\char_value_mathcode:w     3121 \cs_new_nopar:Npn \char_value_catcode:w { \tex_the:D \char_set_catcode:w }
\char_show_value_mathcode:w 3122 \cs_new_nopar:Npn \char_show_value_catcode:w
\char_value_lccode:w       3123 { \tex_showthe:D \char_set_catcode:w }
\char_show_value_lccode:w   3124 \cs_new_nopar:Npn \char_value_mathcode:w { \tex_the:D \char_set_mathcode:w }
\char_value_uccode:w       3125 \cs_new_nopar:Npn \char_show_value_mathcode:w
\char_show_value_uccode:w   3126 { \tex_showthe:D \char_set_mathcode:w }
\char_value_sfcode:w       3127 \cs_new_nopar:Npn \char_value_lccode:w { \tex_the:D \char_set_lccode:w }
\char_show_value_sfcode:w

```

```

3128 \cs_new_nopar:Npn \char_show_value_lccode:w
3129 { \tex_showthe:D \char_set_lccode:w }
3130 \cs_new_nopar:Npn \char_value_uccode:w { \tex_the:D \char_set_uccode:w }
3131 \cs_new_nopar:Npn \char_show_value_uccode:w
3132 { \tex_showthe:D \char_set_uccode:w }
3133 \cs_new_nopar:Npn \char_value_sfcode:w { \tex_the:D \char_set_sfcode:w }
3134 \cs_new_nopar:Npn \char_show_value_sfcode:w
3135 { \tex_showthe:D \char_set_sfcode:w }
3136 \deprecated

```

(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
3137 \*deprecated
3138 \cs_new_eq:NN \peek_after:NN \peek_after:Nw
3139 \cs_new_eq:NN \peek_gafter:NN \peek_gafter:Nw
3140 \deprecated

```

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

Functions deprecated 2011-05-28 for removal by 2011-08-31.

```

\c_alignment_tab_token
\c_math_shift_token
\c_letter_token
\c_other_char_token
3141 \*deprecated
3142 \cs_new_eq:NN \c_alignment_tab_token \c_alignment_token
3143 \cs_new_eq:NN \c_math_shift_token \c_math_toggle_token
3144 \cs_new_eq:NN \c_letter_token \c_catcode_letter_token
3145 \cs_new_eq:NN \c_other_char_token \c_catcode_other_token
3146 \deprecated

```

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

\c_active_char_token An odd one: this was never a token!

```

3147 \*deprecated
3148 \cs_new_eq:NN \c_active_char_token \c_catcode_active_tl
3149 \deprecated

```

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

\char_make_escape:N Two renames in one block!

```

\char_make_group_begin:N
\char_make_group_end:N
\char_make_math_toggle:N
\char_make_alignment:N
\char_make_end_line:N
\char_make_parameter:N
\char_make_math_superscript:N
\char_make_math_subscript:N
\char_make_ignore:N
\char_make_space:N
\char_make_letter:N
\char_make_other:N
\char_make_active:N
\char_make_comment:N
\char_make_invalid:N
\char_make_escape:n
\char_make_group_begin:n
\char_make_group_end:n
\char_make_math_toggle:n
\char_make_alignment:n
\char_make_end_line:n
\char_make_parameter:n
\char make math superscript:n
3150 \*deprecated
3151 \cs_new_eq:NN \char_make_escape:N \char_set_catcode_escape:N
3152 \cs_new_eq:NN \char_make_begin_group:N \char_set_catcode_group_begin:N
3153 \cs_new_eq:NN \char_make_end_group:N \char_set_catcode_group_end:N
3154 \cs_new_eq:NN \char_make_math_shift:N \char_set_catcode_math_toggle:N
3155 \cs_new_eq:NN \char_make_alignment_tab:N \char_set_catcode_alignment:N
3156 \cs_new_eq:NN \char_make_end_line:N \char_set_catcode_end_line:N
3157 \cs_new_eq:NN \char_make_parameter:N \char_set_catcode_parameter:N
3158 \cs_new_eq:NN \char_make_math_superscript:N
3159 \char_set_catcode_math_superscript:N
3160 \cs_new_eq:NN \char_make_math_subscript:N
3161 \char_set_catcode_math_subscript:N
3162 \cs_new_eq:NN \char_make_ignore:N \char_set_catcode_ignore:N
3163 \cs_new_eq:NN \char_make_space:N \char_set_catcode_space:N

```



```

3164 \cs_new_eq:NN \char_make_letter:N \char_set_catcode_letter:N
3165 \cs_new_eq:NN \char_make_other:N \char_set_catcode_other:N
3166 \cs_new_eq:NN \char_make_active:N \char_set_catcode_active:N
3167 \cs_new_eq:NN \char_make_comment:N \char_set_catcode_comment:N
3168 \cs_new_eq:NN \char_make_invalid:N \char_set_catcode_invalid:N
3169 \cs_new_eq:NN \char_make_escape:n \char_set_catcode_escape:n
3170 \cs_new_eq:NN \char_make_begin_group:n \char_set_catcode_group_begin:n
3171 \cs_new_eq:NN \char_make_end_group:n \char_set_catcode_group_end:n
3172 \cs_new_eq:NN \char_make_math_shift:n \char_set_catcode_math_toggle:n
3173 \cs_new_eq:NN \char_make_alignment_tab:n \char_set_catcode_alignment:n
3174 \cs_new_eq:NN \char_make_end_line:n \char_set_catcode_end_line:n
3175 \cs_new_eq:NN \char_make_parameter:n \char_set_catcode_parameter:n
3176 \cs_new_eq:NN \char_make_math_superscript:n
3177 \char_set_catcode_math_superscript:n
3178 \cs_new_eq:NN \char_make_math_subscript:n
3179 \char_set_catcode_math_subscript:n
3180 \cs_new_eq:NN \char_make_ignore:n \char_set_catcode_ignore:n
3181 \cs_new_eq:NN \char_make_space:n \char_set_catcode_space:n
3182 \cs_new_eq:NN \char_make_letter:n \char_set_catcode_letter:n
3183 \cs_new_eq:NN \char_make_other:n \char_set_catcode_other:n
3184 \cs_new_eq:NN \char_make_active:n \char_set_catcode_active:n
3185 \cs_new_eq:NN \char_make_comment:n \char_set_catcode_comment:n
3186 \cs_new_eq:NN \char_make_invalid:n \char_set_catcode_invalid:n
3187 \</deprecat

```

(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

```

```

3188 \<*/deprecat
3189 \cs_new_eq:NN \token_if_alignment_tab_p:N \token_if_alignment_p:N
3190 \cs_new_eq:NN \token_if_alignment_tab_NT \token_if_alignment_NT
3191 \cs_new_eq:NN \token_if_alignment_tab_NF \token_if_alignment_NF
3192 \cs_new_eq:NN \token_if_alignment_tab_NTF \token_if_alignment_NTF
3193 \cs_new_eq:NN \token_if_math_shift_p:N \token_if_math_toggle_p:N
3194 \cs_new_eq:NN \token_if_math_shift_NT \token_if_math_toggle_NT
3195 \cs_new_eq:NN \token_if_math_shift_NF \token_if_math_toggle_NF
3196 \cs_new_eq:NN \token_if_math_shift_NTF \token_if_math_toggle_NTF
3197 \cs_new_eq:NN \token_if_other_char_p:N \token_if_other_p:N
3198 \cs_new_eq:NN \token_if_other_char_NT \token_if_other_NT
3199 \cs_new_eq:NN \token_if_other_char_NF \token_if_other_NF
3200 \cs_new_eq:NN \token_if_other_char_NTF \token_if_other_NTF
3201 \cs_new_eq:NN \token_if_active_char_p:N \token_if_active_p:N
3202 \cs_new_eq:NN \token_if_active_char_NT \token_if_active_NT
3203 \cs_new_eq:NN \token_if_active_char_NF \token_if_active_NF
3204 \cs_new_eq:NN \token_if_active_char_NTF \token_if_active_NTF
3205 \</deprecat

```

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

```

3206 \</initex | package)

```

186 l3int implementation

```

3207 <*initex | package>
      The following test files are used for this code: m3int001,m3int002,m3int03.
3208 <*package>
3209 \ProvidesExplPackage
3210   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3211 \package_check_loaded_expl:
3212 </package>

\int_to_roman:w Done in l3basics.
\if_int_compare:w (End definition for \int_to_roman:w. This function is documented on page 68.)

\int_value:w Here are the remaining primitives for number comparisons and expressions.
\int_eval:w 3213 \cs_new_eq:NN \int_value:w \tex_number:D
\int_eval_end: 3214 \cs_new_eq:NN \int_eval:w \etex_numexpr:D
\if_num:w 3215 \cs_new_eq:NN \int_eval_end: \tex_relax:D
\if_int_odd:w 3216 \cs_new_eq:NN \if_num:w \tex_ifnum:D
\if_case:w 3217 \cs_new_eq:NN \if_int_odd:w \tex_ifodd:D
3218 \cs_new_eq:NN \if_case:w \tex_ifcase:D
      (End definition for \int_value:w. This function is documented on page 69.)

```

186.1 Integer expressions

`\int_eval:n` Wrapper for `\int_eval:w`. Can be used in an integer expression or directly in the input stream. In format mode, there is already a definition in `l3alloc` for bookstrapping, which is therefore corrected to the “real” version here.

```

3219 <*initex>
3220 \cs_set:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3221 </initex>
3222 <*package>
3223 \cs_new:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3224 </package>
      (End definition for \int_eval:n. This function is documented on page 59.)

\int_max:nn Functions for min, max, and absolute value.
\int_min:nn
\int_abs:n 3225 \cs_new:Npn \int_abs:n #1
3226   {
3227     \int_value:w
3228     \if_int_compare:w \int_eval:w #1 < \c_zero
3229       -
3230     \fi:
3231     \int_eval:w #1 \int_eval_end:
3232   }
3233 \cs_new:Npn \int_max:nn #1#2
3234   {
3235     \int_value:w \int_eval:w
3236     \if_int_compare:w
3237       \int_eval:w #1 > \int_eval:w #2 \int_eval_end:

```

```

3238         #1
3239     \else:
3240         #2
3241     \fi:
3242 \int_eval_end:
3243 }
3244 \cs_new:Npn \int_min:nn #1#2
3245 {
3246     \int_value:w \int_eval:w
3247     \if_int_compare:w
3248         \int_eval:w #1 < \int_eval:w #2 \int_eval_end:
3249         #1
3250     \else:
3251         #2
3252     \fi:
3253 \int_eval_end:
3254 }

```

(End definition for `\int_max:nn`. This function is documented on page 59.)

`\int_div_truncate:nn` As `\int_eval:w` rounds the result of a division we also provide a version that truncates the result. This version is thanks to Heiko Oberdiek: getting things right in all cases is not so easy.

```

3255 \cs_new:Npn \int_div_truncate:nn #1#2
3256 {
3257     \int_value:w \int_eval:w
3258     \if_int_compare:w \int_eval:w #1 = \c_zero
3259         0
3260     \else:
3261         ( #1 % )
3262     \if_int_compare:w \int_eval:w #1 < \c_zero
3263         \if_int_compare:w \int_eval:w #2 < \c_zero
3264             - ( #2 + % )
3265         \else:
3266             + ( #2 - % )
3267         \fi:
3268     \else:
3269         \if_int_compare:w \int_eval:w #2 < \c_zero
3270             + ( #2 + % )
3271         \else:
3272             - ( #2 - % )
3273         \fi:
3274     \fi: % ( (
3275         1 ) / 2 )
3276     \fi:
3277     / ( #2 )
3278 \int_eval_end:
3279 }

```

For the sake of completeness:

```

3280 \cs_new:Npn \int_div_round:nn #1#2 { \int_eval:n { ( #1 ) / ( #2 ) } }

```

Finally there's the modulus operation.

```

3281 \cs_new:Npn \int_mod:nn #1#2
3282 {
3283   \int_value:w \int_eval:w
3284     #1 - \int_div_truncate:nn {#1} {#2} * ( #2 )
3285   \int_eval_end:
3286 }

```

(End definition for `\int_div_truncate:nn`. This function is documented on page 60.)

186.2 Creating and initialising integers

`\int_new:N` Two ways to do this: one for the format and one for the $\text{\LaTeX} 2_{\epsilon}$ package.

```

\int_new:c 3287 <*package>
3288 \cs_new_protected_nopar:Npn \int_new:N #1
3289 {
3290   \chk_if_free_cs:N #1
3291   \newcount #1
3292 }
3293 </package>
3294 \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.

```

\int_const:cn 3295 \cs_new_protected:Npn \int_const:Nn #1#2
3296 {
3297   \int_compare:nNnTF {#2} > \c_minus_one
3298   {
3299     \int_compare:nNnTF {#2} > \c_max_register_int
3300     {
3301       \int_new:N #1
3302       \int_gset:Nn #1 {#2}
3303     }
3304     {
3305       \chk_if_free_cs:N #1
3306       \tex_global:D \tex_mathchardef:D #1 =
3307         \int_eval:w #2 \int_eval_end:
3308     }
3309   }
3310   {
3311     \int_new:N #1
3312     \int_gset:Nn #1 {#2}
3313   }
3314 }
3315 \cs_generate_variant:Nn \int_const:Nn { c }

```

(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` 3316 `\cs_new_protected_nopar:Npn \int_zero:N #1 { #1 = \c_zero }`

`\int_gzero:N` 3317 `\cs_new_protected_nopar:Npn \int_gzero:N #1 { \tex_global:D #1 = \c_zero }`

`\int_gzero:c` 3318 `\cs_generate_variant:Nn \int_zero:N { c }`

3319 `\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_set_eq:NN` Setting equal means using one integer inside the set function of another.

`\int_set_eq:cN` 3320 `\cs_new_protected_nopar:Npn \int_set_eq:NN #1#2 { #1 = #2 }`

`\int_set_eq:Nc` 3321 `\cs_generate_variant:Nn \int_set_eq:NN { c }`

`\int_set_eq:cc` 3322 `\cs_generate_variant:Nn \int_set_eq:NN { Nc , cc }`

`\int_gset_eq:NN` 3323 `\cs_new_protected_nopar:Npn \int_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }`

`\int_gset_eq:cN` 3324 `\cs_generate_variant:Nn \int_gset_eq:NN { c }`

`\int_gset_eq:Nc` 3325 `\cs_generate_variant:Nn \int_gset_eq:NN { Nc , cc }`

`\int_gset_eq:cc` (End definition for `\int_set_eq:NN` and others. These functions are documented on page ??.)

186.3 Setting and incrementing integers

`\int_add:Nn` Adding and subtracting to and from a counter ...

`\int_add:cn` 3326 `\cs_new_protected:Npn \int_add:Nn #1#2`

`\int_gadd:Nn` 3327 `{ \tex_advance:D #1 by \int_eval:w #2 \int_eval_end: }`

`\int_gadd:cn` 3328 `\cs_new_protected:Npn \int_sub:Nn #1#2`

`\int_sub:Nn` 3329 `{ \tex_advance:D #1 by - \int_eval:w #2 \int_eval_end: }`

`\int_sub:cn` 3330 `\cs_new_protected_nopar:Npn \int_gadd:Nn`

`\int_gsub:Nn` 3331 `{ \tex_global:D \int_add:Nn }`

`\int_gsub:cn` 3332 `\cs_new_protected_nopar:Npn \int_gsub:Nn`

3333 `{ \tex_global:D \int_sub:Nn }`

3334 `\cs_generate_variant:Nn \int_add:Nn { c }`

3335 `\cs_generate_variant:Nn \int_gadd:Nn { c }`

3336 `\cs_generate_variant:Nn \int_sub:Nn { c }`

3337 `\cs_generate_variant:Nn \int_gsub:Nn { c }`

(End definition for `\int_add:Nn` and `\int_add:cn`. These functions are documented on page ??.)

`\int_incr:N` Incrementing and decrementing of integer registers is done with the following functions.

`\int_incr:c` 3338 `\cs_new_protected_nopar:Npn \int_incr:N #1`

`\int_gincr:N` 3339 `{ \tex_advance:D #1 \c_one }`

`\int_gincr:c` 3340 `\cs_new_protected_nopar:Npn \int_decr:N #1`

`\int_decr:N` 3341 `{ \tex_advance:D #1 \c_minus_one }`

`\int_decr:c` 3342 `\cs_new_protected_nopar:Npn \int_gincr:N`

`\int_gdecr:N` 3343 `{ \tex_global:D \int_incr:N }`

`\int_gdecr:c` 3344 `\cs_new_protected_nopar:Npn \int_gdecr:N`

3345 `{ \tex_global:D \int_decr:N }`

3346 `\cs_generate_variant:Nn \int_incr:N { c }`

3347 `\cs_generate_variant:Nn \int_decr:N { c }`

3348 `\cs_generate_variant:Nn \int_gincr:N { c }`

3349 `\cs_generate_variant:Nn \int_gdecr:N { c }`

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

`\int_set:Nn` As integers are register-based TeX will issue an error if they are not defined. Thus there
`\int_set:cn` is no need for the checking code seen with token list variables.
`\int_gset:Nn`
`\int_gset:cn`

```

3350 \cs_new_protected:Npn \int_set:Nn #1#2
3351 { #1 ~ \int_eval:w #2\int_eval_end: }
3352 \cs_new_protected_nopar:Npn \int_gset:Nn { \tex_global:D \int_set:Nn }
3353 \cs_generate_variant:Nn \int_set:Nn { c }
3354 \cs_generate_variant:Nn \int_gset:Nn { c }

```

(End definition for `\int_set:Nn` and `\int_set:cn`. These functions are documented on page ??.)

186.4 Using integers

`\int_use:N` Here is how counters are accessed:
`\int_use:c`

```

3355 \cs_new_eq:NN \int_use:N \tex_the:D
3356 \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 ??.)

186.5 Integer expression conditionals

`\int_compare:n` Comparison tests using a simple syntax where only one set of braces is required and
`\int_compare_aux:nw` additional operators such as `!=` and `>=` are supported. First some notes on the idea
`\int_compare_aux:Nw` behind this. We wish to support writing code like

```

\int_compare_p:n { 5 + \l_tmpa_int != 4 - \l_tmpb_int }

```

`int_compare_=:w`
`int_compare_=:w`
`int_compare_!=:w`
`int_compare_<:w`
`int_compare_>:w`
`int_compare_<=:w`
`int_compare_>=:w`

In other words, we want to somehow add the missing `\int_eval:w` where required. We can start evaluating from the left using `\int_eval:w`, and we know that since the relation symbols `<`, `>`, `=` and `!` are not allowed in such expressions, they will terminate the expression. Therefore, we first let TeX evaluate this left hand side of the (in)equality.

```

3357 \prg_new_conditional:Npnn \int_compare:n #1 { p , T , F , TF }
3358 { \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.

```

3359 \cs_new:Npn \int_compare_aux:nw #1#2 \q_stop
3360 {
3361   \exp_after:wN \int_compare_aux:Nw
3362   \int_to_roman:w
3363   \if:w #1 -
3364   \else:
3365     -
3366   \fi:
3367   #1#2 \q_mark #1#2 \q_stop
3368 }

```

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.

```

3369 \cs_new:Npn \int_compare_aux:Nw #1#2#3 \q_mark
3370 { \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:

```

3371 \cs_new:cpn { int_compare_=:w } #1 = #2 \q_stop
3372 {
3373   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3374   \prg_return_true:
3375   \else:
3376   \prg_return_false:
3377   \fi:
3378 }

```

So is the one using == we just have to use == in the parameter text.

```

3379 \cs_new:cpn { int_compare_==:w } #1 == #2 \q_stop
3380 {
3381   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3382   \prg_return_true:
3383   \else:
3384   \prg_return_false:
3385   \fi:
3386 }

```

Not equal is just about reversing the truth value.

```

3387 \cs_new:cpn { int_compare_!=:w } #1 != #2 \q_stop
3388 {
3389   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3390   \prg_return_false:
3391   \else:
3392   \prg_return_true:
3393   \fi:
3394 }

```

Less than and greater than are also straight forward.

```

3395 \cs_new:cpn { int_compare_<:w } #1 < #2 \q_stop
3396 {
3397   \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3398   \prg_return_true:
3399   \else:
3400   \prg_return_false:
3401   \fi:
3402 }
3403 \cs_new:cpn { int_compare_>:w } #1 > #2 \q_stop
3404 {
3405   \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3406   \prg_return_true:

```

```

3407     \else:
3408         \prg_return_false:
3409     \fi:
3410 }

```

The less than or equal operation is just the opposite of the greater than operation. *Vice versa* for less than or equal.

```

3411 \cs_new:cpn { int_compare_<=:w } #1 <= #2 \q_stop
3412 {
3413     \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3414         \prg_return_false:
3415     \else:
3416         \prg_return_true:
3417     \fi:
3418 }
3419 \cs_new:cpn { int_compare_>=:w } #1 >= #2 \q_stop
3420 {
3421     \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3422         \prg_return_false:
3423     \else:
3424         \prg_return_true:
3425     \fi:
3426 }

```

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

`\int_compare:nNn` More efficient but less natural in typing.

```

3427 \prg_new_conditional:Npnn \int_compare:nNn #1#2#3 { p , T , F , TF}
3428 {
3429     \if_int_compare:w \int_eval:w #1 #2 \int_eval:w #3 \int_eval_end:
3430         \prg_return_true:
3431     \else:
3432         \prg_return_false:
3433     \fi:
3434 }

```

(End definition for \int_compare:nNn. This function is documented on page 62.)

`\int_if_odd:n` A predicate function.

`\int_if_even:n`

```

3435 \prg_new_conditional:Npnn \int_if_odd:n #1 { p , T , F , TF}
3436 {
3437     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3438         \prg_return_true:
3439     \else:
3440         \prg_return_false:
3441     \fi:
3442 }
3443 \prg_new_conditional:Npnn \int_if_even:n #1 { p , T , F , TF}
3444 {
3445     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3446         \prg_return_false:

```



```

3447     \else:
3448         \prg_return_true:
3449     \fi:
3450 }

```

(End definition for \int_if_odd:n. This function is documented on page 62.)

186.6 Integer expression loops

\int_while_do:nn These are quite easy given the above functions. The **while** versions test first and then execute the body. The **do_while** does it the other way round.

```

\int_while_do:nn 3451 \cs_new:Npn \int_while_do:nn #1#2
\int_until_do:nn 3452 {
\int_do_while:nn 3453     \int_compare:nT {#1}
\int_do_until:nn 3454     {
3455         #2
3456         \int_while_do:nn {#1} {#2}
3457     }
3458 }
3459 \cs_new:Npn \int_until_do:nn #1#2
3460 {
3461     \int_compare:nF {#1}
3462     {
3463         #2
3464         \int_until_do:nn {#1} {#2}
3465     }
3466 }
3467 \cs_new:Npn \int_do_while:nn #1#2
3468 {
3469     #2
3470     \int_compare:nT {#1}
3471     { \int_do_while:nn {#1} {#2} }
3472 }
3473 \cs_new:Npn \int_do_until:nn #1#2
3474 {
3475     #2
3476     \int_compare:nF {#1}
3477     { \int_do_until:nn {#1} {#2} }
3478 }

```

(End definition for \int_while_do:nn. This function is documented on page 63.)

\int_while_do:nNnn As above but not using the more natural syntax.

```

\int_while_do:nNnn 3479 \cs_new:Npn \int_while_do:nNnn #1#2#3#4
\int_until_do:nNnn 3480 {
\int_do_while:nNnn 3481     \int_compare:nNnT {#1} #2 {#3}
\int_do_until:nNnn 3482     {
3483         #4
3484         \int_while_do:nNnn {#1} #2 {#3} {#4}
3485     }
3486 }

```

```

3487 \cs_new:Npn \int_until_do:nNnn #1#2#3#4
3488 {
3489   \int_compare:nNnF {#1} #2 {#3}
3490   {
3491     #4
3492     \int_until_do:nNnn {#1} #2 {#3} {#4}
3493   }
3494 }
3495 \cs_new:Npn \int_do_while:nNnn #1#2#3#4
3496 {
3497   #4
3498   \int_compare:nNnT {#1} #2 {#3}
3499   { \int_do_while:nNnn {#1} #2 {#3} {#4} }
3500 }
3501 \cs_new:Npn \int_do_until:nNnn #1#2#3#4
3502 {
3503   #4
3504   \int_compare:nNnF {#1} #2 {#3}
3505   { \int_do_until:nNnn {#1} #2 {#3} {#4} }
3506 }

```

(End definition for `\int_while_do:nNnn`. This function is documented on page 63.)

186.7 Formatting integers

`\int_to_arabic:n` Nothing exciting here.

```

3508 \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 64.)

`\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.

```

3508 \cs_new:Npn \int_to_symbols:nnn #1#2#3
3509 {
3510   \int_compare:nNnTF {#1} > {#2}
3511   {
3512     \exp_args:NNo \exp_args:No \int_to_symbols_aux:nnnn
3513     {
3514       \prg_case_int:nnn
3515       { 1 + \int_mod:nn { #1 - 1 } {#2} }
3516       {#3} { }
3517     }
3518     {#1} {#2} {#3}
3519   }
3520   { \prg_case_int:nnn {#1} {#3} { } }

```

```

3521 }
3522 \cs_new:Npn \int_to_symbols_aux:nnnn #1#2#3#4
3523 {
3524   \exp_args:Nf \int_to_symbols:nnn
3525   { \int_div_truncate:nn { #2 - 1 } {#3} } {#3} {#4}
3526   #1
3527 }

```

(End definition for \int_to_symbols:nnn. This function is documented on page 65.)

\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.

```

3528 \cs_new:Npn \int_to_alph:n #1
3529 {
3530   \int_to_symbols:nnn {#1} { 26 }
3531   {
3532     { 1 } { a }
3533     { 2 } { b }
3534     { 3 } { c }
3535     { 4 } { d }
3536     { 5 } { e }
3537     { 6 } { f }
3538     { 7 } { g }
3539     { 8 } { h }
3540     { 9 } { i }
3541     { 10 } { j }
3542     { 11 } { k }
3543     { 12 } { l }
3544     { 13 } { m }
3545     { 14 } { n }
3546     { 15 } { o }
3547     { 16 } { p }
3548     { 17 } { q }
3549     { 18 } { r }
3550     { 19 } { s }
3551     { 20 } { t }
3552     { 21 } { u }
3553     { 22 } { v }
3554     { 23 } { w }
3555     { 24 } { x }
3556     { 25 } { y }
3557     { 26 } { z }
3558   }
3559 }
3560 \cs_new:Npn \int_to_Alph:n #1
3561 {
3562   \int_to_symbols:nnn {#1} { 26 }
3563   {
3564     { 1 } { A }
3565     { 2 } { B }

```

```

3566      { 3 } { C }
3567      { 4 } { D }
3568      { 5 } { E }
3569      { 6 } { F }
3570      { 7 } { G }
3571      { 8 } { H }
3572      { 9 } { I }
3573      { 10 } { J }
3574      { 11 } { K }
3575      { 12 } { L }
3576      { 13 } { M }
3577      { 14 } { N }
3578      { 15 } { O }
3579      { 16 } { P }
3580      { 17 } { Q }
3581      { 18 } { R }
3582      { 19 } { S }
3583      { 20 } { T }
3584      { 21 } { U }
3585      { 22 } { V }
3586      { 23 } { W }
3587      { 24 } { X }
3588      { 25 } { Y }
3589      { 26 } { Z }
3590    }
3591  }

```

(End definition for `\int_to_alph:n` and `\int_to_Alph:n`. These functions are documented on page 64.)

`\int_to_base:nn` 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.

`\int_to_base_aux_i:nn`

`\int_to_base_aux_ii:nnN`

`\int_to_base_aux_iii:nnnN`

`\int_to_letter:n`

```

3592 \cs_new:Npn \int_to_base:nn #1
3593 { \exp_args:Nf \int_to_base_aux_i:nn { \int_eval:n {#1} } }
3594 \cs_new:Npn \int_to_base_aux_i:nn #1#2
3595 {
3596   \int_compare:nNnTF {#1} < \c_zero
3597   { \exp_args:No \int_to_base_aux_ii:nnN { \use_none:n #1 } {#2} - }
3598   { \int_to_base_aux_ii:nnN {#1} {#2} \c_empty_tl }
3599 }

```

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.

```

3600 \cs_new:Npn \int_to_base_aux_ii:nnN #1#2#3
3601 {

```

```

3602 \int_compare:nNnTF {#1} < {#2}
3603 { \exp_last_unbraced:Nf #3 { \int_to_letter:n {#1} } }
3604 {
3605   \exp_args:Nf \int_to_base_aux_iii:nnnN
3606   { \int_to_letter:n { \int_mod:nn {#1} {#2} } }
3607   {#1}
3608   {#2}
3609   #3
3610 }
3611 }
3612 \cs_new:Npn \int_to_base_aux_iii:nnnN #1#2#3#4
3613 {
3614   \exp_args:Nf \int_to_base_aux_ii:nnN
3615   { \int_div_truncate:nn {#2} {#3} }
3616   {#3}
3617   #4
3618   #1
3619 }

```

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:`.

```

3620 \cs_new:Npn \int_to_letter:n #1
3621 {
3622   \exp_after:wN \exp_after:wN
3623   \if_case:w \int_eval:w #1 - \c_ten \int_eval_end:
3624   A
3625   \or: B
3626   \or: C
3627   \or: D
3628   \or: E
3629   \or: F
3630   \or: G
3631   \or: H
3632   \or: I
3633   \or: J
3634   \or: K
3635   \or: L
3636   \or: M
3637   \or: N
3638   \or: O
3639   \or: P
3640   \or: Q
3641   \or: R
3642   \or: S
3643   \or: T
3644   \or: U

```

```

3645     \or: V
3646     \or: W
3647     \or: X
3648     \or: Y
3649     \or: Z
3650     \else: \int_value:w \int_eval:w #1 \exp_after:wN \int_eval_end:
3651     \fi:
3652 }

```

(End definition for \int_to_base:nn. This function is documented on page 68.)

```

\int_to_binary:n Wrappers around the generic function.
\int_to_hexadecimal:n
\int_to_octal:n

```

```

3653 \cs_new:Npn \int_to_binary:n #1
3654 { \int_to_base:nn {#1} { 2 } }
3655 \cs_new:Npn \int_to_hexadecimal:n #1
3656 { \int_to_base:nn {#1} { 16 } }
3657 \cs_new:Npn \int_to_octal:n #1
3658 { \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 65.)

```

\int_to_roman:n The \int_to_roman:w primitive creates tokens of category code 12 (other). Usually,
\int_to_Roman:n what is actually wanted is letters. The approach here is to convert the output of the
\int_to_roman_aux:N primitive into letters using appropriate control sequence names. That keeps everything
\int_to_roman_aux:N expandable. The loop will be terminated by the conversion of the Q.

```

```

\int_to_roman_i:w 3659 \cs_new:Npn \int_to_roman:n #1
\int_to_roman_v:w 3660 {
\int_to_roman_x:w 3661     \exp_after:wN \int_to_roman_aux:N
\int_to_roman_l:w 3662     \int_to_roman:w \int_eval:n {#1} Q
\int_to_roman_c:w 3663 }
\int_to_roman_d:w 3664 \cs_new_nopar:Npn \int_to_roman_aux:N #1
\int_to_roman_m:w 3665 {
\int_to_roman_Q:w 3666     \use:c { int_to_roman_ #1 :w }
\int_to_Roman_i:w 3667     \int_to_roman_aux:N
\int_to_Roman_v:w 3668 }
\int_to_Roman_x:w 3669 \cs_new:Npn \int_to_Roman:n #1
\int_to_Roman_l:w 3670 {
\int_to_Roman_c:w 3671     \exp_after:wN \int_to_Roman_aux:N
\int_to_Roman_d:w 3672     \int_to_roman:w \int_eval:n {#1} Q
\int_to_Roman_m:w 3673 }
\int_to_Roman_Q:w 3674 \cs_new_nopar:Npn \int_to_Roman_aux:N #1
\int_to_Roman_Q:w 3675 {
3676     \use:c { int_to_Roman_ #1 :w }
3677     \int_to_Roman_aux:N
3678 }
3679 \cs_new_nopar:Npn \int_to_roman_i:w { i }
3680 \cs_new_nopar:Npn \int_to_roman_v:w { v }
3681 \cs_new_nopar:Npn \int_to_roman_x:w { x }
3682 \cs_new_nopar:Npn \int_to_roman_l:w { l }
3683 \cs_new_nopar:Npn \int_to_roman_c:w { c }

```

```

3684 \cs_new_nopar:Npn \int_to_roman_d:w { d }
3685 \cs_new_nopar:Npn \int_to_roman_m:w { m }
3686 \cs_new_nopar:Npn \int_to_roman_Q:w #1 { }
3687 \cs_new_nopar:Npn \int_to_Roman_i:w { I }
3688 \cs_new_nopar:Npn \int_to_Roman_v:w { V }
3689 \cs_new_nopar:Npn \int_to_Roman_x:w { X }
3690 \cs_new_nopar:Npn \int_to_Roman_l:w { L }
3691 \cs_new_nopar:Npn \int_to_Roman_c:w { C }
3692 \cs_new_nopar:Npn \int_to_Roman_d:w { D }
3693 \cs_new_nopar:Npn \int_to_Roman_m:w { M }
3694 \cs_new_nopar: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 ??.)

186.8 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.
`\int_get_digits:n` This is done by working through token by token until there is something else at the start
`\int_get_sign_and_digits_aux:nNNN` of the input. The sign of the input is tracked by the first Boolean used by the auxiliary
`\int_get_sign_and_digits_aux:oNNN` function.

```

3695 \cs_new:Npn \int_get_sign:n #1
3696 {
3697   \int_get_sign_and_digits_aux:nNNN {#1}
3698   \c_true_bool \c_true_bool \c_false_bool
3699 }
3700 \cs_new:Npn \int_get_digits:n #1
3701 {
3702   \int_get_sign_and_digits_aux:nNNN {#1}
3703   \c_true_bool \c_false_bool \c_true_bool
3704 }

```

The auxiliary loops through, finding sign tokens and removing them. The sign itself is carried through as a flag.

```

3705 \cs_new:Npn \int_get_sign_and_digits_aux:nNNN #1#2#3#4
3706 {
3707   \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} -
3708   {
3709     \bool_if:NTF #2
3710     {
3711       \int_get_sign_and_digits_aux:oNNN
3712       { \use_none:n #1 } \c_false_bool #3#4
3713     }
3714     {
3715       \int_get_sign_and_digits_aux:oNNN
3716       { \use_none:n #1 } \c_true_bool #3#4
3717     }
3718   }
3719   {
3720     \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} +

```

```

3721         { \int_get_sign_and_digits_aux:oNNN { \use_none:n #1 } #2#3#4 }
3722         {
3723             \bool_if:NT #3 { \bool_if:NF #2 - }
3724             \bool_if:NT #4 {#1}
3725         }
3726     }
3727 }
3728 \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`

```

3729 \cs_new:Npn \int_from_alph:n #1
3730 {
3731     \int_eval:n
3732     {
3733         \int_get_sign:n {#1}
3734         \exp_args:Nf \int_from_alph_aux:n { \int_get_digits:n {#1} }
3735     }
3736 }
3737 \cs_new:Npn \int_from_alph_aux:n #1
3738 { \int_from_alph_aux:nN { 0 } #1 \q_nil }
3739 \cs_new:Npn \int_from_alph_aux:nN #1#2
3740 {
3741     \quark_if_nil:NTF #2
3742     {#1}
3743     {
3744         \exp_args:Nf \int_from_alph_aux:nN
3745         { \int_eval:n { #1 * 26 + \int_from_alph_aux:N #2 } }
3746     }
3747 }
3748 \cs_new:Npn \int_from_alph_aux:N #1
3749 { \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`

```

3750 \cs_new:Npn \int_from_base:nn #1#2
3751 {
3752     \int_eval:n
3753     {
3754         \int_get_sign:n {#1}
3755         \exp_args:Nf \int_from_base_aux:nn
3756         { \int_get_digits:n {#1} } {#2}
3757     }
3758 }
3759 \cs_new:Npn \int_from_base_aux:nn #1#2
3760 { \int_from_base_aux:nnN { 0 } { #2 } #1 \q_nil }

```



```

3761 \cs_new:Npn \int_from_base_aux:nnN #1#2#3
3762 {
3763   \quark_if_nil:NTF #3
3764   {#1}
3765   {
3766     \exp_args:Nf \int_from_base_aux:nnN
3767     { \int_eval:n { #1 * #2 + \int_from_base_aux:N #3 } }
3768     {#2}
3769   }
3770 }

```

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.

```

3771 \cs_new:Npn \int_from_base_aux:N #1
3772 {
3773   \int_compare:nNnTF { '#1 } < { 58 }
3774   {#1}
3775   {
3776     \int_eval:n
3777     { '#1 - \int_compare:nNnTF { '#1 } < { 91 } { 55 } { 87 } }
3778   }
3779 }

```

(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.

```

3780 \cs_new:Npn \int_from_binary:n #1
3781 { \int_from_base:nn {#1} \c_two }
3782 \cs_new:Npn \int_from_hexadecimal:n #1
3783 { \int_from_base:nn {#1} \c_sixteen }
3784 \cs_new:Npn \int_from_octal:n #1
3785 { \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 66.)

```

\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.

```

3786 \int_const:cn { c_int_from_roman_i_int } { 1 }
3787 \int_const:cn { c_int_from_roman_v_int } { 5 }
3788 \int_const:cn { c_int_from_roman_x_int } { 10 }
3789 \int_const:cn { c_int_from_roman_l_int } { 50 }
3790 \int_const:cn { c_int_from_roman_c_int } { 100 }
3791 \int_const:cn { c_int_from_roman_d_int } { 500 }
3792 \int_const:cn { c_int_from_roman_m_int } { 1000 }
3793 \int_const:cn { c_int_from_roman_I_int } { 1 }
3794 \int_const:cn { c_int_from_roman_V_int } { 5 }
3795 \int_const:cn { c_int_from_roman_X_int } { 10 }
3796 \int_const:cn { c_int_from_roman_L_int } { 50 }
3797 \int_const:cn { c_int_from_roman_C_int } { 100 }
3798 \int_const:cn { c_int_from_roman_D_int } { 500 }
3799 \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` 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 .

`\int_from_roman_aux:NN`

`\int_from_roman_end:w`

`\int_from_roman_clean_up:w`

```

3800 \cs_new_nopar:Npn \int_from_roman:n #1
3801 {
3802   \tl_if_blank:nF {#1}
3803   {
3804     \exp_after:wN \int_from_roman_end:w
3805     \int_value:w \int_eval:w
3806     \int_from_roman_aux:NN #1 Q \q_stop
3807   }
3808 }
3809 \cs_new_nopar:Npn \int_from_roman_aux:NN #1#2
3810 {
3811   \str_if_eq:nnTF {#1} { Q }
3812   {#1#2}
3813   {
3814     \str_if_eq:nnTF {#2} { Q }
3815     {
3816       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3817       { \int_from_roman_clean_up:w }
3818       +
3819       \use:c { c_int_from_roman_ #1 _int }
3820       #2
3821     }
3822     {
3823       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3824       { \int_from_roman_clean_up:w }
3825       \cs_if_exist:cF { c_int_from_roman_ #2 _int }
3826       { \int_from_roman_clean_up:w }
3827       \int_compare:nNnTF
3828       { \use:c { c_int_from_roman_ #1 _int } }
3829       <
3830       { \use:c { c_int_from_roman_ #2 _int } }
3831       {
3832         + \use:c { c_int_from_roman_ #2 _int }
3833         - \use:c { c_int_from_roman_ #1 _int }
3834         \int_from_roman_aux:NN
3835       }
3836       {
3837         + \use:c { c_int_from_roman_ #1 _int }
3838         \int_from_roman_aux:NN #2
3839       }
3840     }
3841   }
3842 }
3843 \cs_new_nopar:Npn \int_from_roman_end:w #1 Q #2 \q_stop
3844 { \tl_if_empty:nTF {#2} {#1} {#2} }

```

```
3845 \cs_new_nopar: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 ??.)
```

186.9 Viewing integer

```
\int_show:N
\int_show:c 3846 \cs_new_eq:NN \int_show:N \kernel_register_show:N
3847 \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 ??.)
```

186.10 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 67.)

`\c_zero` Again, one in `l3basics` for obvious reasons.
(End definition for \c_zero. This function is documented on page 67.)

`\c_six` Once again, in `l3basics`.
`\c_seven` *(End definition for \c_six and \c_seven. These functions are documented on page 67.)*

`\c_twelve`
`\c_one` Low-number values not previously defined.
`\c_sixteen`
`\c_two`

```
3848 \int_const:Nn \c_one      { 1 }
3849 \int_const:Nn \c_two      { 2 }
3850 \int_const:Nn \c_three     { 3 }
3851 \int_const:Nn \c_four      { 4 }
3852 \int_const:Nn \c_five      { 5 }
3853 \int_const:Nn \c_eight     { 8 }
3854 \int_const:Nn \c_nine      { 9 }
3855 \int_const:Nn \c_ten       { 10 }
3856 \int_const:Nn \c_eleven    { 11 }
3857 \int_const:Nn \c_thirteen { 13 }
3858 \int_const:Nn \c_fourteen  { 14 }
3859 \int_const:Nn \c_fifteen   { 15 }
      (End definition for \c_one and others. These functions are documented on page 67.)
```

`\c_thirty_two` One middling value.

```
3860 \int_const:Nn \c_thirty_two { 32 }
      (End definition for \c_thirty_two. This function is documented on page 67.)
```

`\c_two_hundred_fifty_five` Two classic mid-range integer constants.

```
\c_two_hundred_fifty_six 3861 \int_const:Nn \c_two_hundred_fifty_five { 255 }
3862 \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 67.)
```

`\c_one_hundred` Simple runs of powers of ten.
`\c_one_thousand` 3863 `\int_const:Nn \c_one_hundred { 100 }`
`\c_ten_thousand` 3864 `\int_const:Nn \c_one_thousand { 1000 }`
3865 `\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 67.)

`\c_max_int` The largest number allowed is $2^{31} - 1$
3866 `\int_const:Nn \c_max_int { 2 147 483 647 }`
(End definition for `\c_max_int`. This function is documented on page 67.)

186.11 Scratch integers

`\l_tmpa_int` We provide four local and two global scratch counters, maybe we need more or less.
`\l_tmpb_int` 3867 `\int_new:N \l_tmpa_int`
`\l_tmpc_int` 3868 `\int_new:N \l_tmpb_int`
`\g_tmpa_int` 3869 `\int_new:N \l_tmpc_int`
`\g_tmpb_int` 3870 `\int_new:N \g_tmpa_int`
3871 `\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 67.)

186.12 Registers for earlier modules

Needed from other modules:

3872 `\int_new:N \g_seq_nesting_depth_int`
3873 `\int_new:N \g_tl_inline_level_int`

186.13 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` 3874 `\cs_new_eq:NN \int_convert_from_base_ten:nn \int_to_base:nn`
`\int_convert_to_base_ten:nn` 3875 `\cs_new_eq:NN \int_convert_to_symbols:nnn \int_to_symbols:nnn`
3876 `\cs_new_eq:NN \int_convert_to_base_ten:nn \int_from_base:nn`
(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` 3877 `\cs_new_nopar:Npn \int_to_symbol:n`
`\int_to_symbol_text:n` 3878 `{`
3879 `\scan_align_safe_stop:`
3880 `\mode_if_math:TF`
3881 `{ \int_to_symbol_math:n }`
3882 `{ \int_to_symbol_text:n }`
3883 `}`
3884 `\cs_new:Npn \int_to_symbol_math:n #1`
3885 `{`

```

3886 \int_to_symbols:nnn {#1} { 9 }
3887 {
3888   { 1 } {          * }
3889   { 2 } {        \dagger }
3890   { 3 } {       \ddagger }
3891   { 4 } {     \mathsection }
3892   { 5 } {   \mathparagraph }
3893   { 6 } {         \l }
3894   { 7 } {         ** }
3895   { 8 } {   \dagger \dagger }
3896   { 9 } { \ddagger \ddagger }
3897 }
3898 }
3899 \cs_new:Npn \int_to_symbol_text:n #1
3900 {
3901   \int_to_symbols:nnn {#1} { 9 }
3902   {
3903     { 1 } {          \textasteriskcentered }
3904     { 2 } {        \textdagger }
3905     { 3 } {       \textdaggerdbl }
3906     { 4 } {     \textsection }
3907     { 5 } {   \textparagraph }
3908     { 6 } {         \textbardbl }
3909     { 7 } { \textasteriskcentered \textasteriskcentered }
3910     { 8 } {        \textdagger \textdagger }
3911     { 9 } {       \textdaggerdbl \textdaggerdbl }
3912   }
3913 }
3914 (End definition for \int_to_symbol:n. This function is documented on page ??.)
3914 \</initex | package>

```

187 l3skip implementation

```

3915 \<*initex | package>
3916 \<*package>
3917 \ProvidesExplPackage
3918   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3919   \package_check_loaded_expl:
3920 \</package>

```

187.1 Length primitives renamed

```

\if_dim:w Primitives renamed.
\dim_eval:w 3921 \cs_new_eq:NN \if_dim:w      \tex_ifdim:D
\dim_eval_end: 3922 \cs_new_eq:NN \dim_eval:w    \etex_dimexpr:D
3923 \cs_new_eq:NN \dim_eval_end: \tex_relax:D
(End definition for \if_dim:w. This function is documented on page ??.)

```

187.2 Creating and initialising dim variables

`\dim_new:N` Allocating $\langle dim \rangle$ registers ...

```
\dim_new:c 3924 \*package>
3925 \cs_new_protected:Npn \dim_new:N #1
3926 {
3927     \chk_if_free_cs:N #1
3928     \newdimen #1
3929 }
3930 \</package>
3931 \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 3932 \cs_new_protected:Npn \dim_zero:N #1 { #1 \c_zero_dim }
\dim_gzero:N 3933 \cs_new_protected:Npn \dim_gzero:N { \tex_global:D \dim_zero:N }
\dim_gzero:c 3934 \cs_generate_variant:Nn \dim_zero:N { c }
3935 \cs_generate_variant:Nn \dim_gzero:N { c }
(End definition for \dim_zero:N and \dim_zero:c. These functions are documented on page ??.)
```

187.3 Setting dim variables

`\dim_set:Nn` Setting dimensions is easy enough.

```
\dim_set:cn 3936 \cs_new_protected:Npn \dim_set:Nn #1#2
\dim_gset:Nn 3937 { #1 ~ \dim_eval:w #2 \dim_eval_end: }
\dim_gset:cn 3938 \cs_new_protected:Npn \dim_gset:Nn { \tex_global:D \dim_set:Nn }
3939 \cs_generate_variant:Nn \dim_set:Nn { c }
3940 \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 3941 \cs_new_protected:Npn \dim_set_eq:NN #1#2 { #1 = #2 }
\dim_set_eq:Nc 3942 \cs_generate_variant:Nn \dim_set_eq:NN { c }
\dim_set_eq:cc 3943 \cs_generate_variant:Nn \dim_set_eq:NN { Nc , cc }
\dim_gset_eq:NN 3944 \cs_new_protected:Npn \dim_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\dim_gset_eq:cN 3945 \cs_generate_variant:Nn \dim_gset_eq:NN { c }
\dim_gset_eq:Nc 3946 \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_max:cn 3947 \cs_new_protected:Npn \dim_set_max:Nn #1#2
\dim_set_min:cn 3948 { \dim_compare:nNnT {#1} < {#2} { \dim_set:Nn #1 {#2} } }
\dim_gset_max:Nn 3949 \cs_new_protected:Npn \dim_gset_max:Nn
\dim_gset_max:cn 3950 { \tex_global:D \dim_set_max:Nn }
\dim_gset_min:Nn 3951 \cs_new_protected:Npn \dim_set_min:Nn #1#2
\dim_gset_min:cn 3952 { \dim_compare:nNnT {#1} > {#2} { \dim_set:Nn #1 {#2} } }
3953 \cs_new_protected:Npn \dim_gset_min:Nn
```

```

3954 { \tex_global:D \dim_set_min:Nn }
3955 \cs_generate_variant:Nn \dim_set_max:Nn { c }
3956 \cs_generate_variant:Nn \dim_gset_max:Nn { c }
3957 \cs_generate_variant:Nn \dim_set_min:Nn { c }
3958 \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 3959 \cs_new_protected:Npn \dim_add:Nn #1#2
\dim_gadd:Nn 3960 { \tex_advance:D #1 by \dim_eval:w #2 \dim_eval_end: }
\dim_gadd:cn 3961 \cs_new_protected:Npn \dim_gadd:Nn { \tex_global:D \dim_add:Nn }
\dim_sub:Nn 3962 \cs_generate_variant:Nn \dim_add:Nn { c }
\dim_sub:cn 3963 \cs_generate_variant:Nn \dim_gadd:Nn { c }
\dim_gsub:Nn 3964 \cs_new_protected:Npn \dim_sub:Nn #1#2
\dim_gsub:cn 3965 { \tex_advance:D #1 by - \dim_eval:w #2 \dim_eval_end: }
3966 \cs_new_protected:Npn \dim_gsub:Nn { \tex_global:D \dim_sub:Nn }
3967 \cs_generate_variant:Nn \dim_sub:Nn { c }
3968 \cs_generate_variant:Nn \dim_gsub:Nn { c }

```

(End definition for `\dim_add:Nn` and `\dim_add:cn`. These functions are documented on page ??.)

187.4 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.

```

3969 \cs_new:Npn \dim_abs:n #1
3970 {
3971   \dim_use:N
3972     \dim_eval:w
3973       \if_dim:w \dim_eval:w #1 < \c_zero_dim
3974       -
3975       \fi:
3976     \dim_eval:w #1 \dim_eval_end:
3977   \dim_eval_end:
3978 }

```

(End definition for `\dim_abs:n`. This function is documented on page 72.)

`\dim_ratio:nn` With dimension expressions, something like `10 pt * (5 pt / 10 pt)` will not work.
`\dim_ratio_aux:n` Instead, the ratio part needs to be converted to an integer expression. Using `\int_value:w` forces everything into `sp`, avoiding any decimal parts.

```

3979 \cs_new:Npn \dim_ratio:nn #1#2
3980 { \dim_ratio_aux:n {#1} / \dim_ratio_aux:n {#2} }
3981 \cs_new:Npn \dim_ratio_aux:n #1
3982 { \int_value:w \dim_eval:w #1 \dim_eval_end: }

```

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

187.5 Dimension expression conditionals

```

\dim_compare_p:nNn
\dim_compare:nNn
3983 \prg_new_conditional:Npnn \dim_compare:nNn #1#2#3 { p , T , F , TF }
3984 {
3985   \if_dim:w \dim_eval:w #1 #2 \dim_eval:w #3 \dim_eval_end:
3986   \prg_return_true: \else: \prg_return_false: \fi:
3987 }

```

(End definition for `\dim_compare_p:nNn`. This function is documented on page 72.)

```

\dim_compare:n [This code plus comments are adapted from the \int_compare:nTF function.] Compar-
\dim_compare_aux:wNN ison tests using a simple syntax where only one set of braces is required and additional
\dim_compare_<:nw operators such as != and >= are supported. First some notes on the idea behind this.
\dim_compare_=:nw We wish to support writing code like
\dim_compare_>:nw
\dim_compare_==:nw
\dim_compare_<=:nw
\dim_compare_!=:nw
\dim_compare_>=:nw

```

```

\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.

```

3988 \prg_new_conditional:Npnn \dim_compare:n #1 { p , T , F , TF }
3989 {
3990   \exp_after:wN \dim_compare_aux:wNN \dim_use:N \dim_eval:w #1
3991   \dim_eval_end:
3992   \prg_return_true:
3993   \else:
3994   \prg_return_false:
3995   \fi:
3996 }

```

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.

```

3997 \exp_args:Nno \use:nn
3998 { \cs_new:Npn \dim_compare_aux:wNN #1 }
3999 { \tl_to_str:n { pt } }
4000 #2 #3
4001 {
4002   \use:c

```



```

4003     {
4004         dim_compare_ #2
4005         \if_meaning:w = #3 = \fi:
4006         :nw
4007     }
4008     { #1 pt } #3
4009 }

```

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.

```

4010 \cs_new:cpn { dim_compare_<:nw } #1 { \if_dim:w #1 < \dim_eval:w }
4011 \cs_new:cpn { dim_compare_=:nw } #1 { \if_dim:w #1 = \dim_eval:w }
4012 \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 `=`.

```

4013 \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.

```

4014 \cs_new:cpn {dim_compare_<=:nw} #1#2 {\reverse_if:N \if_dim:w #1 > \dim_eval:w}
4015 \cs_new:cpn {dim_compare_!=:nw} #1#2 {\reverse_if:N \if_dim:w #1 = \dim_eval:w}
4016 \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 ??.)

187.6 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 4017 \cs_set:Npn \dim_while_do:nn #1#2
\dim_do_until:nn 4018 {
4019     \dim_compare:nT {#1}
4020     {
4021         #2
4022         \dim_while_do:nn {#1} {#2}
4023     }
4024 }
4025 \cs_set:Npn \dim_until_do:nn #1#2
4026 {
4027     \dim_compare:nF {#1}
4028     {
4029         #2

```

```

4030         \dim_until_do:nn {#1} {#2}
4031     }
4032 }
4033 \cs_set:Npn \dim_do_while:nn #1#2
4034 {
4035     #2
4036     \dim_compare:nT {#1}
4037     { \dim_do_while:nNnn {#1} {#2} }
4038 }
4039 \cs_set:Npn \dim_do_until:nn #1#2
4040 {
4041     #2
4042     \dim_compare:nF {#1}
4043     { \dim_do_until:nn {#1} {#2} }
4044 }

```

(End definition for `\dim_while_do:nn`. This function is documented on page 74.)

`\dim_while_do:nNnn` while_do and do_while functions for dimensions. Same as for the `int` type only the names have changed.

```

\dim_until_do:nNnn
\dim_do_while:nNnn
\dim_do_until:nNnn
4045 \cs_set:Npn \dim_while_do:nNnn #1#2#3#4
4046 {
4047     \dim_compare:nNnT {#1} #2 {#3}
4048     {
4049         #4
4050         \dim_while_do:nNnn {#1} #2 {#3} {#4}
4051     }
4052 }
4053 \cs_set:Npn \dim_until_do:nNnn #1#2#3#4
4054 {
4055     \dim_compare:nNnF {#1} #2 {#3}
4056     {
4057         #4
4058         \dim_until_do:nNnn {#1} #2 {#3} {#4}
4059     }
4060 }
4061 \cs_set:Npn \dim_do_while:nNnn #1#2#3#4
4062 {
4063     #4
4064     \dim_compare:nNnT {#1} #2 {#3}
4065     { \dim_do_while:nNnn {#1} #2 {#3} {#4} }
4066 }
4067 \cs_set:Npn \dim_do_until:nNnn #1#2#3#4
4068 {
4069     #4
4070     \dim_compare:nNnF {#1} #2 {#3}
4071     { \dim_do_until:nNnn {#1} #2 {#3} {#4} }
4072 }

```

(End definition for `\dim_while_do:nNnn`. This function is documented on page 73.)

187.7 Using dim expressions and variables

`\dim_eval:n` Evaluating a dimension expression expandably.

```
4073 \cs_new:Npn \dim_eval:n #1
4074 { \dim_use:N \dim_eval:w #1 \dim_eval_end: }
(End definition for \dim_eval:n. This function is documented on page 74.)
```

`\dim_strip_bp:n`

```
4075 \cs_new:Npn \dim_strip_bp:n #1
4076 { \dim_strip_pt:n { 0.996 26 \dim_eval:w #1 \dim_eval_end: } }
(End definition for \dim_strip_bp:n. This function is documented on page 81.)
```

`\dim_strip_pt:n` A function which comes up often enough to deserve a place in the kernel. The idea here is that the input is assumed to be in pt, but can be given in other units, while the output is the value of the dimension in pt but with no units given. This is used a lot by low-level manipulations.

`\dim_strip_pt:w`

```
4077 \cs_new:Npn \dim_strip_pt:n #1
4078 {
4079   \exp_after:wN
4080   \dim_strip_pt:w \dim_use:N \dim_eval:w #1 \dim_eval_end: \q_stop
4081 }
4082 \use:x
4083 {
4084   \cs_new:Npn \exp_not:N \dim_strip_pt:w
4085   ##1 . ##2 \tl_to_str:n { pt } ##3 \exp_not:N \q_stop
4086   {
4087     ##1
4088     \exp_not:N \int_compare:nNnT {##2} > \c_zero
4089     { . ##2 }
4090   }
4091 }
(End definition for \dim_strip_pt:n. This function is documented on page ??.)
```

`\dim_use:N` Accessing a $\langle dim \rangle$.

`\dim_use:c`

```
4092 \cs_new_eq:NN \dim_use:N \tex_the:D
4093 \cs_generate_variant:Nn \dim_use:N { c }
(End definition for \dim_use:N and \dim_use:c. These functions are documented on page ??.)
```

187.8 Viewing dim variables

`\dim_show:N` Diagnostics.

`\dim_show:c`

```
4094 \cs_new_eq:NN \dim_show:N \kernel_register_show:N
4095 \cs_generate_variant:Nn \dim_show:N { c }
(End definition for \dim_show:N and \dim_show:c. These functions are documented on page ??.)
```

187.9 Constant dimensions

`\c_zero_dim` The source for these depends on whether we are in package mode.

```
\c_max_dim
4096 \*initex>
4097 \dim_new:N \c_zero_dim
4098 \dim_new:N \c_max_dim
4099 \dim_set:Nn \c_max_dim { 16383.99999 pt }
4100 \*initex>
4101 \*package>
4102 \cs_new_eq:NN \c_zero_dim \z@
4103 \cs_new_eq:NN \c_max_dim \maxdimen
4104 \*package>
(End definition for \c_zero_dim. This function is documented on page 75.)
```

187.10 Scratch dimensions

`\l_tmpa_dim` We provide three local and two global scratch registers, maybe we need more or less.

```
\l_tmpb_dim 4105 \dim_new:N \l_tmpa_dim
\l_tmpc_dim 4106 \dim_new:N \l_tmpb_dim
\g_tmpa_dim 4107 \dim_new:N \l_tmpc_dim
\g_tmpb_dim 4108 \dim_new:N \g_tmpa_dim
4109 \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 75.)
```

187.11 Creating and initialising skip variables

`\skip_new:N` Allocation of a new internal registers.

```
\skip_new:c 4110 \*package>
4111 \cs_new_protected:Npn \skip_new:N #1
4112 {
4113   \chk_if_free_cs:N #1
4114   \newskip #1
4115 }
4116 \*package>
4117 \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 4118 \cs_new_protected:Npn \skip_zero:N #1 { #1 \c_zero_skip }
\skip_gzero:N 4119 \cs_new_protected:Npn \skip_gzero:N { \tex_global:D \skip_zero:N }
\skip_gzero:c 4120 \cs_generate_variant:Nn \skip_zero:N { c }
4121 \cs_generate_variant:Nn \skip_gzero:N { c }
(End definition for \skip_zero:N and \skip_zero:c. These functions are documented on page ??.)
```

187.12 Setting skip variables

`\skip_set:Nn` Much the same as for dimensions.

```

\skip_set:cn 4122 \cs_new_protected:Npn \skip_set:Nn #1#2
\skip_gset:Nn 4123 { #1 ~ \etex_glueexpr:D #2 \scan_stop: }
\skip_gset:cn 4124 \cs_new_protected:Npn \skip_gset:Nn { \tex_global:D \skip_set:Nn }
4125 \cs_generate_variant:Nn \skip_set:Nn { c }
4126 \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 4127 \cs_new_protected:Npn \skip_set_eq:NN #1#2 { #1 = #2 }
\skip_set_eq:Nc 4128 \cs_generate_variant:Nn \skip_set_eq:NN { c }
\skip_set_eq:cc 4129 \cs_generate_variant:Nn \skip_set_eq:NN { Nc , cc }
\skip_gset_eq:cn 4130 \cs_new_protected:Npn \skip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\skip_gset_eq:Nc 4131 \cs_generate_variant:Nn \skip_gset_eq:NN { c }
\skip_gset_eq:Nc 4132 \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 4133 \cs_new_protected:Npn \skip_add:Nn #1#2
\skip_gadd:Nn 4134 { \tex_advance:D #1 by \etex_glueexpr:D #2 \scan_stop: }
\skip_gadd:cn 4135 \cs_new_protected:Npn \skip_gadd:Nn { \tex_global:D \skip_add:Nn }
\skip_sub:Nn 4136 \cs_generate_variant:Nn \skip_add:Nn { c }
\skip_sub:cn 4137 \cs_generate_variant:Nn \skip_gadd:Nn { c }
\skip_gsub:Nn 4138 \cs_new_protected:Npn \skip_sub:Nn #1#2
\skip_gsub:cn 4139 { \tex_advance:D #1 by - \etex_glueexpr:D #2 \scan_stop: }
4140 \cs_new_protected:Npn \skip_gsub:Nn { \tex_global:D \skip_sub:Nn }
4141 \cs_generate_variant:Nn \skip_sub:Nn { c }
4142 \cs_generate_variant:Nn \skip_gsub:Nn { c }

```

(End definition for `\skip_add:Nn` and `\skip_add:cn`. These functions are documented on page ??.)

187.13 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.

```

4143 \prg_new_conditional:Npnn \skip_if_eq:nn #1#2 { p , T , F , TF }
4144 {
4145   \if_int_compare:w
4146     \pdfTeX_strcmp:D { \skip_eval:n { #1 } } { \skip_eval:n { #2 } }
4147     = \c_zero
4148     \prg_return_true:
4149   \else:
4150     \prg_return_false:
4151   \fi:
4152 }

```

(End definition for `\skip_if_eq:nn`. This function is documented on page 77.)

`\skip_if_infinite_glue:n` With ε -TeX we all of a sudden get access to a lot information we should otherwise consider ourselves lucky to get. One is the stretch and shrink components of a skip register and the order or those components. `\skip_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.

```

4153 \prg_new_conditional:Npnn \skip_if_infinite_glue:n #1 { p , T , F , TF }
4154 {
4155   \bool_if:nTF
4156   {
4157     \int_compare_p:nNn { \etex_gluestretchorder:D #1 } > \c_zero ||
4158     \int_compare_p:nNn { \etex_glueshrinkorder:D #1 } > \c_zero
4159   }
4160   { \prg_return_true: }
4161   { \prg_return_false: }
4162 }

```

(End definition for `\skip_if_infinite_glue:n`. This function is documented on page 77.)

187.14 Using skip expressions and variables

`\skip_eval:n` Evaluating a skip expression expandably.

```

4163 \cs_new:Npn \skip_eval:n #1
4164 { \skip_use:N \etex_glueexpr:D #1 \scan_stop: }

```

(End definition for `\skip_eval:n`. This function is documented on page 77.)

`\skip_use:N` Accessing a $\langle skip \rangle$.

```

\skip_use:c 4165 \cs_new_eq:NN \skip_use:N \tex_the:D
4166 \cs_generate_variant:Nn \skip_use:N { c }

```

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

187.15 Inserting skips into the output

`\skip_horizontal:N` Inserting skips.

```

\skip_horizontal:c 4167 \cs_new_eq:NN \skip_horizontal:N \tex_hskip:D
\skip_horizontal:n 4168 \cs_new:Npn \skip_horizontal:n #1
\skip_vertical:N 4169 { \skip_horizontal:N \etex_glueexpr:D #1 \scan_stop: }
\skip_vertical:c 4170 \cs_new_eq:NN \skip_vertical:N \tex_vskip:D
\skip_vertical:n 4171 \cs_new:Npn \skip_vertical:n #1
4172 { \skip_vertical:N \etex_glueexpr:D #1 \scan_stop: }
4173 \cs_generate_variant:Nn \skip_horizontal:N { c }
4174 \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 ??.)

187.16 Viewing skip variables

`\skip_show:N` Diagnostics.
`\skip_show:c` 4175 `\cs_new_eq:NN \skip_show:N \kernel_register_show:N`
 4176 `\cs_generate_variant:Nn \skip_show:N { c }`
(End definition for `\skip_show:N` and `\skip_show:c`. These functions are documented on page ??.)

187.17 Constant skips

`\c_zero_skip` Skips with no rubber component are just dimensions
`\c_max_skip` 4177 `\cs_new_eq:NN \c_zero_skip \c_zero_dim`
 4178 `\cs_new_eq:NN \c_max_skip \c_max_dim`
(End definition for `\c_zero_skip`. This function is documented on page 78.)

187.18 Scratch skips

`\l_tmpa_skip` We provide three local and two global scratch registers, maybe we need more or less.
`\l_tmpb_skip` 4179 `\skip_new:N \l_tmpa_skip`
`\l_tmpc_skip` 4180 `\skip_new:N \l_tmpb_skip`
`\g_tmpa_skip` 4181 `\skip_new:N \l_tmpc_skip`
`\g_tmpb_skip` 4182 `\skip_new:N \g_tmpa_skip`
 4183 `\skip_new:N \g_tmpb_skip`
(End definition for `\l_tmpa_skip`, `\l_tmpb_skip`, and `\l_tmpc_skip`. These functions are documented on page 78.)

187.19 Creating and initialising muskip variables

`\muskip_new:N` And then we add muskips.
`\muskip_new:c` 4184 `*package`
 4185 `\cs_new_protected:Npn \muskip_new:N #1`
 4186 `{`
 4187 `\chk_if_free_cs:N #1`
 4188 `\newmuskip #1`
 4189 `}`
 4190 `*package`
 4191 `\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` 4192 `\cs_new_protected:Npn \muskip_zero:N #1`
`\muskip_gzero:N` 4193 `{ #1 \c_zero_muskip }`
`\muskip_gzero:c` 4194 `\cs_new_protected:Npn \muskip_gzero:N { \tex_global:D \muskip_zero:N }`
 4195 `\cs_generate_variant:Nn \muskip_zero:N { c }`
 4196 `\cs_generate_variant:Nn \muskip_gzero:N { c }`
(End definition for `\muskip_zero:N` and `\muskip_zero:c`. These functions are documented on page ??.)

187.20 Setting muskip variables

`\muskip_set:Nn` This should be pretty familiar.

```

\muskip_set:cn 4197 \cs_new_protected:Npn \muskip_set:Nn #1#2
\muskip_gset:Nn 4198 { #1 ~ \etex_muexpr:D #2 \scan_stop: }
\muskip_gset:cn 4199 \cs_new_protected:Npn \muskip_gset:Nn { \tex_global:D \muskip_set:Nn }
4200 \cs_generate_variant:Nn \muskip_set:Nn { c }
4201 \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 4202 \cs_new_protected:Npn \muskip_set_eq:NN #1#2 { #1 = #2 }
\muskip_set_eq:Nc 4203 \cs_generate_variant:Nn \muskip_set_eq:NN { c }
\muskip_set_eq:cc 4204 \cs_generate_variant:Nn \muskip_set_eq:NN { Nc , cc }
\muskip_gset_eq:NN 4205 \cs_new_protected:Npn \muskip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\muskip_gset_eq:cN 4206 \cs_generate_variant:Nn \muskip_gset_eq:NN { c }
\muskip_gset_eq:Nc 4207 \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 4208 \cs_new_protected:Npn \muskip_add:Nn #1#2
\muskip_gadd:Nn 4209 { \tex_advance:D #1 by \etex_muexpr:D #2 \scan_stop: }
\muskip_gadd:cn 4210 \cs_new_protected:Npn \muskip_gadd:Nn { \tex_global:D \muskip_add:Nn }
\muskip_sub:Nn 4211 \cs_generate_variant:Nn \muskip_add:Nn { c }
\muskip_sub:cn 4212 \cs_generate_variant:Nn \muskip_gadd:Nn { c }
\muskip_gsub:Nn 4213 \cs_new_protected:Npn \muskip_sub:Nn #1#2
\muskip_gsub:cn 4214 { \tex_advance:D #1 by - \etex_muexpr:D #2 \scan_stop: }
4215 \cs_new_protected:Npn \muskip_gsub:Nn { \tex_global:D \muskip_sub:Nn }
4216 \cs_generate_variant:Nn \muskip_sub:Nn { c }
4217 \cs_generate_variant:Nn \muskip_gsub:Nn { c }

(End definition for \muskip_add:Nn and \muskip_add:cn. These functions are documented on
page ??.)
```

187.21 Using muskip expressions and variables

`\muskip_eval:n` Evaluating a muskip expression expandably.

```

4218 \cs_new:Npn \muskip_eval:n #1
4219 { \muskip_use:N \etex_muexpr:D #1 \scan_stop: }

(End definition for \muskip_eval:n. This function is documented on page 79.)
```

`\muskip_use:N` Accessing a $\langle muskip \rangle$.

```

\muskip_use:c 4220 \cs_new_eq:NN \muskip_use:N \tex_the:D
4221 \cs_generate_variant:Nn \muskip_use:N { c }

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


187.22 Viewing muskip variables

`\muskip_show:N` Diagnostics.
`\muskip_show:c`

```

4222 \cs_new_eq:NN \muskip_show:N \kernel_register_show:N
4223 \cs_generate_variant:Nn \muskip_show:N { c }
      (End definition for \muskip_show:N and \muskip_show:c. These functions are documented on
      page ??.)

```

187.23 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 global.

```

4224 \cs_new:Npn \skip_split_finite_else_action:nnNN #1#2#3#4
4225 {
4226   \skip_if_infinite_glue:nTF {#1}
4227   {
4228     #3 = \c_zero_skip
4229     #4 = \c_zero_skip
4230     #2
4231   }
4232   {
4233     #3 = \etex_gluestretch:D #1 \scan_stop:
4234     #4 = \etex_glueshrink:D #1 \scan_stop:
4235   }
4236 }
      (End definition for \skip_split_finite_else_action:nnNN. This function is documented on page
      81.)
4237 </initex | package>

```

188 l3tl implementation

```

4238 <*initex | package>
4239 <*package>
4240 \ProvidesExplPackage
4241   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4242 \package_check_loaded_expl:
4243 </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.

188.1 Functions

`\tl_new:N` Creating new token list variables is a case of checking for an existing definition and if
`\tl_new:c` free doing the definition.

```

4244 \cs_new_protected_nopar:Npn \tl_new:N #1
4245 {
4246   \chk_if_free_cs:N #1
4247   \cs_gset_eq:NN #1 \c_empty_tl
4248 }
4249 \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 4250 \cs_new_protected:Npn \tl_const:Nn #1#2
\tl_const:cn 4251 {
\tl_const:cx 4252   \chk_if_free_cs:N #1
4253   \cs_gset_nopar:Npx #1 { \exp_not:n {#2} }
4254 }
4255 \cs_new_protected:Npn \tl_const:Nx #1#2
4256 {
4257   \chk_if_free_cs:N #1
4258   \cs_gset_nopar:Npx #1 {#2}
4259 }
4260 \cs_generate_variant:Nn \tl_const:Nn { c }
4261 \cs_generate_variant:Nn \tl_const:Nx { c }

```

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

\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 4262 \cs_new_protected_nopar:Npn \tl_clear:N #1
\tl_gclear:N 4263 { \tl_set_eq:NN #1 \c_empty_tl }
\tl_gclear:c 4264 \cs_new_protected_nopar:Npn \tl_gclear:N #1
4265 { \tl_gset_eq:NN #1 \c_empty_tl }
4266 \cs_generate_variant:Nn \tl_clear:N { c }
4267 \cs_generate_variant:Nn \tl_gclear:N { c }

```

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

\tl_clear_new:N Clearing a token list variable means setting it to an empty value. Error checking will be sorted out by the parent function.

```

\tl_clear_new:c 4268 \cs_new_protected_nopar:Npn \tl_clear_new:N #1
\tl_gclear_new:N 4269 { \cs_if_exist:NTF #1 { \tl_clear:N #1 } { \tl_new:N #1 } }
\tl_gclear_new:c 4270 \cs_new_protected_nopar:Npn \tl_gclear_new:N #1
4271 { \cs_if_exist:NTF #1 { \tl_gclear:N #1 } { \tl_new:N #1 } }
4272 \cs_generate_variant:Nn \tl_clear_new:N { c }
4273 \cs_generate_variant:Nn \tl_gclear_new:N { c }

```

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

\tl_set_eq:NN For setting token list variables equal to each other.

```

\tl_set_eq:Nc 4274 \cs_new_eq:NN \tl_set_eq:NN \cs_set_eq:NN
\tl_set_eq:cN 4275 \cs_new_eq:NN \tl_set_eq:cN \cs_set_eq:cN
\tl_set_eq:cc 4276 \cs_new_eq:NN \tl_set_eq:Nc \cs_set_eq:Nc
\tl_gset_eq:NN 4277 \cs_new_eq:NN \tl_set_eq:cc \cs_set_eq:cc
\tl_gset_eq:Nc
\tl_gset_eq:cN
\tl_gset_eq:cc

```

```

4278 \cs_new_eq:NN \tl_gset_eq:NN \cs_gset_eq:NN
4279 \cs_new_eq:NN \tl_gset_eq:cN \cs_gset_eq:cN
4280 \cs_new_eq:NN \tl_gset_eq:Nc \cs_gset_eq:Nc
4281 \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 ??.)

```

188.2 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 4282 \cs_new_protected:Npn \tl_set:Nn #1#2
\tl_set:Nv 4283 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} } }
\tl_set:Nf 4284 \cs_new_protected:Npn \tl_set:Nf #1#2
\tl_set:Nx 4285 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} } }
\tl_set:cn 4286 \cs_new_protected:Npn \tl_set:Nx #1#2
\tl_set:cV 4287 { \cs_set_nopar:Npx #1 {#2} }
\tl_set:co 4288 \cs_new_protected:Npn \tl_gset:Nn #1#2
\tl_set:cf 4289 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} } }
\tl_set:cx 4290 \cs_new_protected:Npn \tl_gset:No #1#2
\tl_gset:Nn 4291 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} } }
\tl_gset:Nv 4292 \cs_new_protected:Npn \tl_gset:Nx #1#2
\tl_gset:Nf 4293 { \cs_gset_nopar:Npx #1 {#2} }
\tl_gset:Nx 4294 \cs_generate_variant:Nn \tl_set:Nn { NV , Nv , Nf }
\tl_gset:cn 4295 \cs_generate_variant:Nn \tl_set:Nx { c }
\tl_gset:cV 4296 \cs_generate_variant:Nn \tl_set:Nn { c , co , cV , cv , cf }
\tl_gset:co 4297 \cs_generate_variant:Nn \tl_gset:Nn { NV , Nv , Nf }
\tl_gset:cf 4298 \cs_generate_variant:Nn \tl_gset:Nx { c }
\tl_gset:cx 4299 \cs_generate_variant:Nn \tl_gset:Nn { c , co , cV , cv , cf }
(End definition for \tl_set:Nn and others. These functions are documented on page ??.)

```

Adding to the left is done directly to gain a little performance.

```

\tl_put_left:Nn 4300 \cs_new_protected:Npn \tl_put_left:Nn #1#2
\tl_put_left:Nv 4301 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_put_left:Nf 4302 \cs_new_protected:Npn \tl_put_left:Nv #1#2
\tl_put_left:Nx 4303 { \cs_set_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_put_left:cn 4304 \cs_new_protected:Npn \tl_put_left:No #1#2
\tl_put_left:cV 4305 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_put_left:co 4306 \cs_new_protected:Npn \tl_put_left:Nx #1#2
\tl_put_left:cx 4307 { \cs_set_nopar:Npx #1 { #2 \exp_not:o #1 } }
\tl_gput_left:Nn 4308 \cs_new_protected:Npn \tl_gput_left:Nn #1#2
\tl_gput_left:Nv 4309 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_gput_left:Nf 4310 \cs_new_protected:Npn \tl_gput_left:Nv #1#2
\tl_gput_left:Nx 4311 { \cs_gset_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_gput_left:cn 4312 \cs_new_protected:Npn \tl_gput_left:No #1#2
\tl_gput_left:cV 4313 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_gput_left:co 4314 \cs_new_protected:Npn \tl_gput_left:Nx #1#2
\tl_gput_left:cx 4315 { \cs_gset_nopar:Npx #1 { #2 \exp_not:o {#1} } }
4316 \cs_generate_variant:Nn \tl_put_left:Nn { c }

```

```

4317 \cs_generate_variant:Nn \tl_put_left:NV { c }
4318 \cs_generate_variant:Nn \tl_put_left:No { c }
4319 \cs_generate_variant:Nn \tl_put_left:Nx { c }
4320 \cs_generate_variant:Nn \tl_gput_left:Nn { c }
4321 \cs_generate_variant:Nn \tl_gput_left:NV { c }
4322 \cs_generate_variant:Nn \tl_gput_left:No { c }
4323 \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 4324 \cs_new_protected:Npn \tl_put_right:Nn #1#2
\tl_put_right:No 4325 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_put_right:Nx 4326 \cs_new_protected:Npn \tl_put_right:NV #1#2
\tl_put_right:cn 4327 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_put_right:cV 4328 \cs_new_protected:Npn \tl_put_right:No #1#2
\tl_put_right:co 4329 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_put_right:cx 4330 \cs_new_protected:Npn \tl_put_right:Nx #1#2
\tl_gput_right:Nn 4331 { \cs_set_nopar:Npx #1 { \exp_not:o #1 #2 } }
\tl_gput_right:NV 4332 \cs_new_protected:Npn \tl_gput_right:Nn #1#2
\tl_gput_right:No 4333 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_gput_right:Nx 4334 \cs_new_protected:Npn \tl_gput_right:NV #1#2
\tl_gput_right:cn 4335 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_gput_right:cV 4336 \cs_new_protected:Npn \tl_gput_right:No #1#2
\tl_gput_right:co 4337 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_gput_right:cx 4338 \cs_new_protected:Npn \tl_gput_right:Nx #1#2
4339 { \cs_gset_nopar:Npx #1 { \exp_not:o {#1} #2 } }
4340 \cs_generate_variant:Nn \tl_put_right:Nn { c }
4341 \cs_generate_variant:Nn \tl_put_right:NV { c }
4342 \cs_generate_variant:Nn \tl_put_right:No { c }
4343 \cs_generate_variant:Nn \tl_put_right:Nx { c }
4344 \cs_generate_variant:Nn \tl_gput_right:Nn { c }
4345 \cs_generate_variant:Nn \tl_gput_right:NV { c }
4346 \cs_generate_variant:Nn \tl_gput_right:No { c }
4347 \cs_generate_variant:Nn \tl_gput_right:Nx { c }

```

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

188.3 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.

```

4348 \group_begin:
4349 \tex_lccode:D '\A = '\@ \scan_stop:
4350 \tex_lccode:D '\B = '\@ \scan_stop:
4351 \tex_catcode:D '\A = 8 \scan_stop:
4352 \tex_catcode:D '\B = 3 \scan_stop:
4353 \tex_lowercase:D
4354 {
4355 \group_end:
4356 \tl_const:Nn \c_tl_rescan_marker_tl { A B }

```

```

4357 }
      (End definition for \c_tl_rescan_marker_tl. This function is documented on page ??.)

\l_tl_rescan_tl A token list variable to actually store the material being processed.
4358 \tl_new:N \l_tl_rescan_tl
      (End definition for \l_tl_rescan_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
\tl_set_rescan:Nno as a file, and without the correct settings a TEX error occurs:
\tl_set_rescan:cnn ! File ended while scanning definition of ...
\tl_set_rescan:cno
\tl_gset_rescan:Nnn When expanding a token list this can be handled using \exp_not:N but this fails if
\tl_gset_rescan:Nno the token list is not being expanded. So instead a delimited argument is used with an
\tl_gset_rescan:cnn end marker which cannot appear within the token list which is scanned: two @ symbols
\tl_gset_rescan:cno with different category codes. The rescanned token list cannot contain the end marker,
\tl_set_rescan_aux:NNnn because all @ present in the token list are read with the same category code. As every
\tl_rescan_aux:w 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”.

4359 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnn
4360 { \tl_set_rescan_aux:NNnn \tl_set:Nn }
4361 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnn
4362 { \tl_set_rescan_aux:NNnn \tl_gset:Nn }
4363 \cs_new_protected:Npn \tl_set_rescan_aux:NNnn #1#2#3#4
4364 {
4365   \group_begin:
4366     \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl }
4367     \tex_endlinechar:D \c_minus_one
4368     \tex_newlinechar:D \c_minus_one
4369     #3
4370     \tl_clear:N \l_tl_rescan_tl
4371     \exp_after:wN \tl_rescan_aux:w \etex_scantokens:D {#4}
4372     \exp_args:NNNo \group_end:
4373     #1 #2 \l_tl_rescan_tl
4374   }
4375   \use:x
4376   {
4377     \cs_new_protected:Npn \exp_not:N \tl_rescan_aux:w ##1
4378       \c_tl_rescan_marker_tl
4379       { \tl_set:Nn \exp_not:N \l_tl_rescan_tl {##1} }
4380   }
4381   \cs_generate_variant:Nn \tl_set_rescan:Nnn { Nno }
4382   \cs_generate_variant:Nn \tl_set_rescan:Nnn { c , cno }
4383   \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nno }
4384   \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 ??.)

```

`\tl_set_rescan:Nnx` With x-type expansion the `\everyeof` method does apply and the code is simple.

```

\tl_set_rescan:cnx
\tl_gset_rescan:Nnx
\tl_gset_rescan:cnx
\tl_set_rescan_aux:NNnx
4385 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnx
4386 { \tl_set_rescan_aux:NNnx \tl_set:Nn }
4387 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnx
4388 { \tl_set_rescan_aux:NNnx \tl_gset:Nn }
4389 \cs_new_protected_nopar:Npn \tl_set_rescan_aux:NNnx #1#2#3#4
4390 {
4391   \group_begin:
4392     \etex_everyeof:D { \exp_not:N }
4393     \tex_endlinechar:D \c_minus_one
4394     \tex_newlinechar:D \c_minus_one
4395     #3
4396     \tl_set:Nx \l_tl_rescan_tl { \etex_scantokens:D {#4} }
4397     \exp_args:NNNo \group_end:
4398     #1 #2 \l_tl_rescan_tl
4399   }
4400   \cs_generate_variant:Nn \tl_set_rescan:Nnx { c }
4401   \cs_generate_variant:Nn \tl_gset_rescan:Nnx { c }

```

(End definition for `\tl_set_rescan:Nnx` and `\tl_set_rescan:cnx`. These functions are documented on page ??.)

`\tl_rescan:nn` The same idea is also applied to in line token lists.

```

4402 \cs_new_protected:Npn \tl_rescan:nn #1#2
4403 {
4404   \group_begin:
4405     \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl }
4406     \tex_endlinechar:D \c_minus_one
4407     \tex_newlinechar:D \c_minus_one
4408     #1
4409     \exp_after:wN \tl_rescan_aux:w \etex_scantokens:D {#2}
4410     \exp_args:No \group_end:
4411     \l_tl_rescan_tl
4412   }

```

(End definition for `\tl_rescan:nn`. This function is documented on page 86.)

188.4 Reassigning token list character codes

`\tl_to_lowercase:n` Just some names for a few primitives.

```

\tl_to_uppercase:n
4413 \cs_new_eq:NN \tl_to_lowercase:n \tex_lowercase:D
4414 \cs_new_eq:NN \tl_to_uppercase:n \tex_uppercase:D

```

(End definition for `\tl_to_lowercase:n`. This function is documented on page 86.)

188.5 Modifying token list variables

`\l_tl_replace_tl` A scratch variable for doing token replacement.

```

4415 \tl_new:N \l_tl_replace_tl

```

(End definition for `\l_tl_replace_tl`. This function is documented on page ??.)

\tl_replace_all:Nnn \tl_replace_all:cnm \tl_greplace_all:Nnn \tl_greplace_all:cnm \tl_replace_once:Nnn \tl_replace_once:cnm \tl_greplace_once:Nnn \tl_greplace_once:cnm \tl_replace_aux:NNNnn \tl_replace_aux_ii:w \tl_replace_all_aux: \tl_replace_once_aux: \tl_replace_once_aux_end:w	All of the replace functions are based on \tl_replace_aux:NNNnn, whose arguments are: $\langle function \rangle$, $\backslash tl_ (g) set:Nx$, $\langle tl\ var \rangle$, $\langle search\ tokens \rangle$, $\langle replacement\ tokens \rangle$.
--	---

```

4416 \cs_new_protected_nopar:Npn \tl_replace_once:Nnn
4417 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_set:Nx }
4418 \cs_new_protected_nopar:Npn \tl_greplace_once:Nnn
4419 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_gset:Nx }
4420 \cs_new_protected_nopar:Npn \tl_replace_all:Nnn
4421 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_set:Nx }
4422 \cs_new_protected_nopar:Npn \tl_greplace_all:Nnn
4423 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_gset:Nx }
4424 \cs_generate_variant:Nn \tl_replace_once:Nnn { c }
4425 \cs_generate_variant:Nn \tl_greplace_once:Nnn { c }
4426 \cs_generate_variant:Nn \tl_replace_all:Nnn { c }
4427 \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.

```

4428 \cs_new_protected:Npn \tl_replace_aux:NNNnn #1#2#3#4#5
4429 {
4430   \tl_if_empty:nTF {#4}
4431   {
4432     \msg_kernel_error:nxx { tl } { empty-search-pattern }
4433     { \tl_to_str:n {#5} }
4434   }
4435   {
4436     \cs_set:Npx \tl_tmp:w ##1##2 #4
4437     {
4438       ##2
4439       \exp_not:N \q_mark
4440       \exp_not:N \use_none_delimit_by_q_stop:w
4441       \exp_not:n { \exp_not:n {#5} }
4442       ##1
4443     }
4444     #2 #3
4445     {
4446       \exp_after:wN #1
4447       #3 \q_mark #4 \q_stop

```

```

4448         }
4449     }
4450 }
4451 \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.

```

4452 \cs_new:Npn \tl_replace_all_aux:
4453 {
4454     \exp_after:wN \tl_replace_aux_ii:w
4455     \tl_tmp:w \tl_replace_all_aux: \prg_do_nothing:
4456 }
4457 \cs_new_nopar:Npn \tl_replace_once_aux:
4458 {
4459     \exp_after:wN \tl_replace_aux_ii:w
4460     \tl_tmp:w { \tl_replace_once_aux_end:w \prg_do_nothing: } \prg_do_nothing:
4461 }
4462 \cs_new:Npn \tl_replace_once_aux_end:w #1 \q_mark #2 \q_stop
4463 { \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
\tl_gremove_once:Nn 4464 \cs_new_protected_nopar:Npn \tl_remove_once:Nn #1#2
\tl_gremove_once:cn 4465 { \tl_replace_once:Nnn #1 {#2} { } }
                     4466 \cs_new_protected_nopar:Npn \tl_gremove_once:Nn #1#2
                     4467 { \tl_greplace_once:Nnn #1 {#2} { } }
                     4468 \cs_generate_variant:Nn \tl_remove_once:Nn { c }
                     4469 \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
\tl_gremove_all:Nn 4470 \cs_new_protected_nopar:Npn \tl_remove_all:Nn #1#2
\tl_gremove_all:cn 4471 { \tl_replace_all:Nnn #1 {#2} { } }
                     4472 \cs_new_protected_nopar:Npn \tl_gremove_all:Nn #1#2
                     4473 { \tl_greplace_all:Nnn #1 {#2} { } }
                     4474 \cs_generate_variant:Nn \tl_remove_all:Nn { c }
                     4475 \cs_generate_variant:Nn \tl_gremove_all:Nn { c }

```


188.6 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.

```

4476 \prg_new_conditional:Npnn \tl_if_blank:n #1 { p , T , F , TF }
4477 { \tl_if_empty_return:o { \use_none:n #1 ? } }
4478 \cs_generate_variant:Nn \tl_if_blank_p:n { V }
4479 \cs_generate_variant:Nn \tl_if_blank:nT { V }
4480 \cs_generate_variant:Nn \tl_if_blank:nF { V }
4481 \cs_generate_variant:Nn \tl_if_blank:nTF { V }
4482 \cs_generate_variant:Nn \tl_if_blank_p:n { o }
4483 \cs_generate_variant:Nn \tl_if_blank:nT { o }
4484 \cs_generate_variant:Nn \tl_if_blank:nF { o }
4485 \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`

```

4486 \prg_set_conditional:Npnn \tl_if_empty:N #1 { p , T , F , TF }
4487 {
4488   \if_meaning:w #1 \c_empty_tl
4489   \prg_return_true:
4490   \else:
4491     \prg_return_false:
4492   \fi:
4493 }
4494 \cs_generate_variant:Nn \tl_if_empty_p:N { c }
4495 \cs_generate_variant:Nn \tl_if_empty:NT { c }
4496 \cs_generate_variant:Nn \tl_if_empty:NF { c }
4497 \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.

```

4498 \prg_new_conditional:Npnn \tl_if_empty:n #1 { p , TF , T , F }
4499 {
4500   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil \tl_to_str:n {#1} \q_nil
4501   \prg_return_true:
4502   \else:
4503   \prg_return_false:
4504   \fi:
4505 }
4506 \cs_generate_variant:Nn \tl_if_empty_p:n { V }
4507 \cs_generate_variant:Nn \tl_if_empty:nTF { V }
4508 \cs_generate_variant:Nn \tl_if_empty:nT { V }
4509 \cs_generate_variant:Nn \tl_if_empty:nF { V }

```

(End definition for `\tl_if_empty:n` and `\tl_if_empty:V`. These functions are documented on page ??.)

`\tl_if_empty:o` The auxiliary function `\tl_if_empty_return:o` is for use in conditionals on token lists, which mostly reduce to testing if a given token list is empty after applying a simple function to it. The test for emptiness is based on `\tl_if_empty:n(TF)`, but the expansion is hard-coded for efficiency, as this auxiliary function is used in many places. Note that this works because `\tl_to_str:n` expands tokens that follow until reading a catcode 1 (begin-group) token.

`\tl_if_empty_return:o`

```

4510 \cs_new:Npn \tl_if_empty_return:o #1
4511 {
4512   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil
4513   \tl_to_str:n \exp_after:wN {#1} \q_nil
4514   \prg_return_true:
4515   \else:
4516   \prg_return_false:
4517   \fi:
4518 }
4519 \prg_new_conditional:Npnn \tl_if_empty:o #1 { p , TF , T , F }
4520 { \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.

```

4521 \prg_new_conditional:Npnn \tl_if_eq:NN #1#2 { p , T , F , TF }
4522 {
4523   \if_meaning:w #1 #2
4524   \prg_return_true:
4525   \else:
4526   \prg_return_false:
4527   \fi:
4528 }
4529 \cs_generate_variant:Nn \tl_if_eq_p:NN { Nc , c , cc }
4530 \cs_generate_variant:Nn \tl_if_eq:NNTF { Nc , c , cc }
4531 \cs_generate_variant:Nn \tl_if_eq:NNT { Nc , c , cc }
4532 \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.

```

\tl_tl_tmpa_tl 4533 \prg_new_protected_conditional:Npnn \tl_if_eq:nn #1#2 { T , F , TF }
\tl_tl_tmpb_tl 4534 {
4535   \group_begin:
4536     \tl_set:Nn \tl_tl_tmpa_tl {#1}
4537     \tl_set:Nn \tl_tl_tmpb_tl {#2}
4538     \if_meaning:w \tl_tl_tmpa_tl \tl_tl_tmpb_tl
4539     \group_end:
4540     \prg_return_true:
4541   \else:
4542     \group_end:
4543     \prg_return_false:
4544   \fi:
4545 }
4546 \tl_new:N \tl_tl_tmpa_tl
4547 \tl_new:N \tl_tl_tmpb_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)`.

```

4548 \cs_new_protected_nopar:Npn \tl_if_in:NnT { \exp_args:No \tl_if_in:nnT }
4549 \cs_new_protected_nopar:Npn \tl_if_in:NnF { \exp_args:No \tl_if_in:nnF }
4550 \cs_new_protected_nopar:Npn \tl_if_in:NnTF { \exp_args:No \tl_if_in:nnTF }
4551 \cs_generate_variant:Nn \tl_if_in:NnT { c }
4552 \cs_generate_variant:Nn \tl_if_in:NnF { c }
4553 \cs_generate_variant:Nn \tl_if_in:NnTF { c }

```

(End definition for \tl_if_in:Nn and \tl_if_in:cn. These functions are documented on page ??.)

`\tl_if_in:nn` Once more, the test relies on `\tl_to_str:n` for robustness. The function `\tl_tmp:w`
`\tl_if_in:Vn` removes tokens until the first occurrence of #2. If this does not appear in #1, then the
`\tl_if_in:on` final #2 is removed, leaving an empty token list. Otherwise some tokens remain, and the
`\tl_if_in:no` test is false. See `\tl_if_empty:n(TF)` for details on the emptiness test.

Special care is needed to treat correctly cases like `\tl_if_in:nnTF {a state}{states}`, where #1#2 contains #2 before the end. To cater for this case, we insert `{ }{ }` between the two token lists. This marker may not appear in #2 because of TeX limitations on what can delimit a parameter, hence we are safe. Using two brace groups makes the test work also for empty arguments.

```

4554 \prg_new_protected_conditional:Npnn \tl_if_in:nn #1#2 { T , F , TF }
4555 {
4556   \cs_set:Npn \tl_tmp:w ##1 #2 { }
4557   \tl_if_empty:oTF { \tl_tmp:w #1 {} {} } #2 {
4558     { \prg_return_false: } { \prg_return_true: }
4559   }
4560 \cs_generate_variant:Nn \tl_if_in:nnT { V , o , no }
4561 \cs_generate_variant:Nn \tl_if_in:nnF { V , o , no }
4562 \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 ??.)

188.7 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:NN
\tl_map_function:cN
\tl_map_function_aux:NN
4563 \cs_new:Npn \tl_map_function:nN #1#2
4564 { \tl_map_function_aux:Nn #2 #1 \q_recursion_tail \q_recursion_stop }
4565 \cs_new_nopar:Npn \tl_map_function:NN #1#2
4566 {
4567   \exp_after:wN \tl_map_function_aux:Nn
4568   \exp_after:wN #2 #1 \q_recursion_tail \q_recursion_stop
4569 }
4570 \cs_new:Npn \tl_map_function_aux:Nn #1#2
4571 {
4572   \quark_if_recursion_tail_stop:n {#2}
4573   #1 {#2} \tl_map_function_aux:Nn #1
4574 }
4575 \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_tl_inline_level_int` to make them nestable. We can also make use of `\tl_map_function:Nn` from before. (`\g_tl_inline_level_int` is defined in `l3int` for order-of-loading reasons.)

```
\tl_map_inline:NN
\tl_map_inline:cn
\tl_map_inline_aux:n
\g_tl_inline_level_int
4576 \cs_new_protected:Npn \tl_map_inline:nn #1#2
4577 {
4578   \int_gincr:N \g_tl_inline_level_int
4579   \cs_gset:cpn { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4580   ##1 {#2}
4581   \exp_args:Nc \tl_map_function_aux:Nn
4582   { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4583   #1 \q_recursion_tail \q_recursion_stop
4584   \int_gdecr:N \g_tl_inline_level_int
4585 }
4586 \cs_new_protected:Npn \tl_map_inline:Nn #1#2
4587 {
4588   \int_gincr:N \g_tl_inline_level_int
4589   \cs_gset:cpn { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4590   ##1 {#2}
4591   \exp_last_unbraced:NcV \tl_map_function_aux:Nn
4592   { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4593   #1 \q_recursion_tail \q_recursion_stop
4594   \int_gdecr:N \g_tl_inline_level_int
4595 }
4596 \cs_generate_variant:Nn \tl_map_inline:Nn { c }
```

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

`\tl_map_variable:nNn` `\tl_map_variable:nNn` $\langle token\ list \rangle$ $\langle temp \rangle$ $\langle action \rangle$ assigns $\langle temp \rangle$ to each element and executes $\langle action \rangle$.

`\tl_map_variable:NNn`

`\tl_map_variable:cNn`

`\tl_map_variable_aux:NnN`

```

4597 \cs_new_protected:Npn \tl_map_variable:nNn #1#2#3
4598 { \tl_map_variable_aux:Nnn #2 {#3} #1 \q_recursion_tail \q_recursion_stop }
4599 \cs_new_protected_nopar:Npn \tl_map_variable:NNn
4600 { \exp_args:No \tl_map_variable:nNn }
4601 \cs_new_protected:Npn \tl_map_variable_aux:Nnn #1#2#3
4602 {
4603   \tl_set:Nn #1 {#3}
4604   \quark_if_recursion_tail_stop:N #1
4605   #2 \tl_map_variable_aux:Nnn #1 {#2}
4606 }
4607 \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 statement.

```

4608 \cs_new_eq:NN \tl_map_break: \use_none_delimit_by_q_recursion_stop:w

```

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

188.8 Using token lists

`\tl_to_str:n` Another name for a primitive.

```

4609 \cs_new_eq:NN \tl_to_str:n \etex_detokenize:D

```

(End definition for \tl_to_str:n. This function is documented on page 89.)

`\tl_to_str:N` These functions return the replacement text of a token list as a string.

`\tl_to_str:c`

```

4610 \cs_new_nopar:Npn \tl_to_str:N #1 { \etex_detokenize:D \exp_after:wN {#1} }
4611 \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 \TeX 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.

```

4612 \cs_new_eq:NN \tl_use:N \prg_do_nothing:
4613 \cs_new_nopar:Npn \tl_use:c #1
4614 {
4615   \if_cs_exist:w #1 \cs_end:
4616   \cs:w #1 \exp_after:wN \cs_end:
4617   \else:
4618     \msg_expandable_error:n { Undefined~variable~name~'~#1~'! }
4619   \fi:
4620 }

```

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

188.9 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 4621 \cs_new:Npn \tl_length:n #1
\tl_length:V 4622 {
\tl_length:o 4623   \int_eval:n
\tl_length:N 4624     { 0 \tl_map_function:nN {#1} \tl_length_aux:n }
\tl_length:c 4625   }
\tl_length_aux:n 4626 \cs_new_nopar:Npn \tl_length:N #1
4627 {
4628   \int_eval:n
4629     { 0 \tl_map_function:NN #1 \tl_length_aux:n }
4630   }
4631 \cs_new:Npn \tl_length_aux:n #1 { + 1 }
4632 \cs_generate_variant:Nn \tl_length:n { V , o }
4633 \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_recursion_stop`.

```

\tl_reverse_items_aux:nN 4634 \cs_new:Npn \tl_reverse_items:n #1
4635 { \tl_reverse_items_aux:nw #1 \q_recursion_tail \q_recursion_stop }
4636 \cs_new:Npn \tl_reverse_items_aux:nw #1 #2 \q_recursion_stop
4637 {
4638   \quark_if_recursion_tail_stop_do:nn {#1} { \use_none:n }
4639   \tl_reverse_items_aux:nw #2 \q_recursion_stop
4640   {#1}
4641 }

```

(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.

```

4642 \cs_set:Npn \tl_tmp:w #1
4643 {
4644   \cs_new:Npn \tl_trim_spaces:n ##1

```

```

4645 {
4646   \etex_unexpanded:D
4647   \tl_trim_spaces_aux_i:w
4648   \q_mark
4649   ##1
4650   \q_nil
4651   \q_mark #1 { }
4652   \q_mark \tl_trim_spaces_aux_ii:w
4653   \tl_trim_spaces_aux_iii:w
4654   #1 \q_nil
4655   \tl_trim_spaces_aux_iv:w
4656   \q_stop
4657 }
4658 \cs_new:Npn \tl_trim_spaces_aux_i:w ##1 \q_mark #1 ##2 \q_mark ##3
4659 {
4660   ##3
4661   \tl_trim_spaces_aux_i:w
4662   \q_mark
4663   ##2
4664   \q_mark #1 {##1}
4665 }
4666 \cs_new:Npn \tl_trim_spaces_aux_ii:w ##1 \q_mark \q_mark ##2
4667 {
4668   \tl_trim_spaces_aux_iii:w
4669   ##2
4670 }
4671 \cs_new:Npn \tl_trim_spaces_aux_iii:w ##1 #1 \q_nil ##2
4672 {
4673   ##2
4674   ##1 \q_nil
4675   \tl_trim_spaces_aux_iii:w
4676 }
4677 \cs_new:Npn \tl_trim_spaces_aux_iv:w ##1 \q_nil ##2 \q_stop
4678 { \exp_after:wN { \use_none:n ##1 } }
4679 }
4680 \tl_tmp:w { ~ }
4681 \cs_new_protected:Npn \tl_trim_spaces:N #1
4682 { \tl_set:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4683 \cs_new_protected:Npn \tl_gtrim_spaces:N #1
4684 { \tl_gset:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4685 \cs_generate_variant:Nn \tl_trim_spaces:N { c }
4686 \cs_generate_variant:Nn \tl_gtrim_spaces:N { c }

```

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

188.10 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 and \tl_tail:n ensure that a blank argument gives an empty result.

```

\tl_head:n
\tl_head:V
\tl_head:v
\tl_head:f
\tl_head:w
\tl_tail:n
\tl_tail:V
\tl_tail:v
\tl_tail:f
\tl_tail:w

```

```

4688 \cs_new:Npn \tl_tail:w #1#2 \q_stop {#2}
4689 \cs_new:Npn \tl_head:n #1
4690   { \tl_head:w #1 { } \q_stop }
4691 \cs_new:Npn \tl_tail:n #1
4692   { \tl_tail_aux:w #1 \q_mark { } \q_mark \q_stop }
4693 \cs_new:Npn \tl_tail_aux:w #1 #2 \q_mark #3 \q_stop { #2 }
4694 \cs_generate_variant:Nn \tl_head:n { V , v , f }
4695 \cs_generate_variant:Nn \tl_tail:n { V , v , f }

```

(End definition for `\tl_head:n` and others. These functions are documented on page 92.)

`\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 `\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.

```

4696 \cs_new:Npn \str_head:n #1
4697   {
4698     \exp_after:wN \str_head_aux:w
4699     \tl_to_str:n {#1}
4700     { { } } ~ \q_stop
4701   }
4702 \cs_new_nopar:Npn \str_head_aux:w #1 ~ %
4703   { \tl_head:w #1 { ~ } }
4704 \cs_new:Npn \str_tail:n #1
4705   {
4706     \exp_after:wN \str_tail_aux:w
4707     \reverse_if:N \if_charcode:w
4708       \scan_stop: \tl_to_str:n {#1} X X \q_stop
4709   }
4710 \cs_new_nopar: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` 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.


```

\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).

```

4711 \prg_new_conditional:Npnn \tl_if_head_eq_charcode:nN #1#2 { p , T , F , TF }
4712 {
4713   \if_charcode:w
4714     \exp_not:N #2
4715     \tl_if_head_N_type:nTF { #1 ? }
4716     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4717     { \str_head:n {#1} }
4718   \prg_return_true:
4719   \else:
4720     \prg_return_false:
4721   \fi:
4722 }
4723 \cs_generate_variant:Nn \tl_if_head_eq_charcode_p:nN { f }
4724 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNTF { f }
4725 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNT { f }
4726 \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`.

```

4727 \prg_new_conditional:Npnn \tl_if_head_eq_catcode:nN #1 #2 { p , T , F , TF }
4728 {
4729   \if_catcode:w
4730     \exp_not:N #2
4731     \tl_if_head_N_type:nTF { #1 ? }
4732     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4733     {
4734       \tl_if_head_group:nTF {#1}
4735       { \c_group_begin_token }
4736       { \c_space_token }
4737     }
4738   \prg_return_true:
4739   \else:
4740     \prg_return_false:
4741   \fi:
4742 }

```

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.

```

4743 \prg_new_conditional:Npnn \tl_if_head_eq_meaning:nN #1#2 { p , T , F , TF }
4744 {
4745   \tl_if_head_N_type:nTF { #1 ? }
4746   { \tl_if_head_eq_meaning_aux_normal:nN }
4747   { \tl_if_head_eq_meaning_aux_special:nN }
4748   {#1} #2
4749 }
4750 \cs_new:Npn \tl_if_head_eq_meaning_aux_normal:nN #1 #2
4751 {
4752   \exp_after:wN \if_meaning:w \tl_head:w #1 \q_nil \q_stop #2
4753   \prg_return_true:
4754   \else:
4755     \prg_return_false:
4756   \fi:
4757 }
4758 \cs_new:Npn \tl_if_head_eq_meaning_aux_special:nN #1 #2
4759 {
4760   \if_charcode:w \str_head:n {#1} \exp_not:N #2
4761   \exp_after:wN \use:n
4762   \else:
4763     \prg_return_false:
4764     \exp_after:wN \use_none:n
4765   \fi:
4766   {
4767     \if_catcode:w \exp_not:N #2
4768     \tl_if_head_group:nTF {#1}
4769     { \c_group_begin_token }
4770     { \c_space_token }
4771     \prg_return_true:
4772     \else:
4773       \prg_return_false:
4774     \fi:
4775   }
4776 }

```

(End definition for `\tl_if_head_eq_meaning:nN`. This function is documented on page 92.)

`\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”.

```

4777 \prg_new_conditional:Npnn \tl_if_head_N_type:n #1 { p , T , F , TF }
4778 { \str_if_eq_return:on { \use:n #1 { } } { #1 { } } }

```

(End definition for `\tl_if_head_N_type:n`. This function is documented on page 93.)

`\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

```

4779 \prg_new_conditional:Npnn \tl_if_head_group:n #1 { p , T , F , TF }
4780 {
4781   \if_catcode:w *
4782     \exp_after:wN \use_none:n
4783     \exp_after:wN {
4784       \exp_after:wN {
4785         \token_to_str:N #1 ?
4786       }
4787     }
4788   *
4789   \prg_return_false:
4790 \else:
4791   \prg_return_true:
4792 \fi:
4793 }

```

(End definition for `\tl_if_head_group:n`. This function is documented on page 93.)

`\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.

```

4794 \prg_new_conditional:Npnn \tl_if_head_space:n #1 { p , T , F , TF }
4795 {
4796   \if_int_compare:w
4797     \pdfTeX_strcmp:D
4798     { }
4799     { \tl_if_head_space_aux:w \prg_do_nothing: #1 ? ~ }
4800     = \c_zero
4801     \prg_return_true:
4802 \else:
4803   \prg_return_false:
4804 \fi:
4805 }
4806 \cs_new:Npn \tl_if_head_space_aux:w #1 ~ %
4807 {
4808   \exp_not:o {#1}
4809   \if_false: { \fi: }
4810   \exp_after:wN \use_none:n \exp_after:wN { \if_false: } \fi:
4811 }

```

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

⁶Bruno: this could be made faster, but we don't: if we hope to ever have an e-type argument, we need all brace "tricks" to happen in one step of expansion, keeping the token list brace balanced at all times.

188.11 Viewing token lists

`\tl_show:N` Showing token list variables is done directly: at the moment do not worry if they are defined.
`\tl_show:c`

```
4812 \cs_new_protected:Npn \tl_show:N #1 { \cs_show:N #1 }
4813 \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.

```
4814 \cs_new_eq:NN \tl_show:n \etex_showtokens:D
(End definition for \tl_show:n. This function is documented on page 94.)
```

188.12 Constant token lists

`\c_job_name_tl` Inherited from the L^AT_EX3 name for the primitive: this needs to actually contain the text of the job name rather than the name of the primitive, of course. Lua_T_EX does not quote file names containing spaces, whereas pdf_T_EX and X_T_EX do. So there may be a correction to make in the Lua_T_EX case.

```
4815 <*initex>
4816 \tex_everyjob:D \exp_after:wN
4817 {
4818   \tex_the:D \tex_everyjob:D
4819   \luatex_if_engine:T
4820   {
4821     \lua_now:x
4822     { dofile ( assert ( kpse.find_file ("lualatexquotejobname.lua" ) ) ) }
4823   }
4824 }
4825 </initex>
4826 \tl_const:Nx \c_job_name_tl { \tex_jobname:D }
(End definition for \c_job_name_tl. This function is documented on page 94.)
```

`\c_empty_tl` Never full.

```
4827 \tl_const:Nn \c_empty_tl { }
(End definition for \c_empty_tl. This function is documented on page 94.)
```

`\c_space_tl` A space as a token list (as opposed to as a character).

```
4828 \tl_const:Nn \c_space_tl { ~ }
(End definition for \c_space_tl. This function is documented on page 94.)
```

188.13 Scratch token lists

`\g_tmpa_tl` `\g_tmpb_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.

```
4829 \tl_new:N \g_tmpa_tl
4830 \tl_new:N \g_tmpb_tl
```

(End definition for `\g_tmpa_tl` and `\g_tmpb_tl`. These functions are documented on page 94.)

`\l_tmpa_tl` `\l_tmpb_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.

```
4831 \tl_new:N \l_tmpa_tl
4832 \tl_new:N \l_tmpb_tl
```

(End definition for `\l_tmpa_tl` and `\l_tmpb_tl`. These functions are documented on page 94.)

188.14 Experimental functions

`\str_if_eq_return:on` 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.

```
4833 \cs_new:Npn \str_if_eq_return:on #1 #2
4834 {
4835   \if_int_compare:w
4836     \pdfTeX_strcmp:D { \exp_not:o {#1} } { \exp_not:n {#2} }
4837     = \c_zero
4838     \prg_return_true:
4839   \else:
4840     \prg_return_false:
4841   \fi:
4842 }
```

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

`\tl_if_single:N` Expand the token list and feed it to `\tl_if_single:n`.

```
4843 \cs_new:Npn \tl_if_single_p:N { \exp_args:No \tl_if_single_p:n }
4844 \cs_new:Npn \tl_if_single:NT { \exp_args:No \tl_if_single:nT }
4845 \cs_new:Npn \tl_if_single:NF { \exp_args:No \tl_if_single:nF }
4846 \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 87.)

`\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.

```
4847 \prg_new_conditional:Npnn \tl_if_single:n #1 { p , T , F , TF }
4848 { \str_if_eq_return:on { \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.

```

4849 \prg_new_conditional:Npnn \tl_if_single_token:n #1 { p , T , F , TF }
4850 {
4851   \tl_if_head_N_type:nTF {#1}
4852   { \str_if_eq_return:on { \use_none:n #1 } { } }
4853   { \str_if_eq_return:on { ~ } { #1 } }
4854 }

```

(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 95.)

`\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`.

```

4855 \cs_new:Npn \tl_act:NNNnn { \tex_romannumeral:D \tl_act_aux:NNNnn }
4856 \cs_new:Npn \tl_act_aux:NNNnn #1 #2 #3 #4 #5
4857 {
4858   \tl_act_loop:w #5 \q_tl_act_mark \q_tl_act_stop
4859   {#4} #1 #2 #3
4860   \tl_act_result:n { }
4861 }

```

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:wwnNNN` gobble the space.

```

4862 \cs_new:Npn \tl_act_loop:w #1 \q_tl_act_stop
4863 {
4864   \tl_if_head_N_type:nTF {#1}
4865   { \tl_act_normal:NwnNNN }
4866   {
4867     \tl_if_head_group:nTF {#1}
4868     { \tl_act_group:nwnNNN }
4869     { \tl_act_space:wwnNNN }
4870   }
4871   #1 \q_tl_act_stop

```

```

4872 }
4873 \cs_new:Npn \tl_act_normal:NwnNNN #1 #2 \q_tl_act_stop #3#4
4874 {
4875   \if_meaning:w \q_tl_act_mark #1
4876   \exp_after:wN \tl_act_end:wn
4877   \fi:
4878   #4 {#3} #1
4879   \tl_act_loop:w #2 \q_tl_act_stop
4880   {#3} #4
4881 }
4882 \cs_new:Npn \tl_act_end:wn #1 \tl_act_result:n #2 { \c_zero #2 }
4883 \cs_new:Npn \tl_act_group:nwnNNN #1 #2 \q_tl_act_stop #3#4#5
4884 {
4885   #5 {#3} {#1}
4886   \tl_act_loop:w #2 \q_tl_act_stop
4887   {#3} #4 #5
4888 }
4889 \exp_last_unbraced:NNo
4890 \cs_new:Npn \tl_act_space:wwnNNN \c_space_tl #1 \q_tl_act_stop #2#3#4#5
4891 {
4892   #5 {#2}
4893   \tl_act_loop:w #1 \q_tl_act_stop
4894   {#2} #3 #4 #5
4895 }

```

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.

```

4896 \cs_new:Npn \tl_act_output:n #1 #2 \tl_act_result:n #3
4897 { #2 \tl_act_result:n { #3 #1 } }
4898 \cs_new:Npn \tl_act_reverse_output:n #1 #2 \tl_act_result:n #3
4899 { #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.

```

4900 \cs_new:Npn \tl_act_group_recurse:Nnn #1#2#3
4901 {
4902   \exp_args:Nf #1
4903   { \exp_after:wN \exp_after:wN \exp_after:wN { #2 {#3} } }
4904 }

```

(End definition for `\tl_act:NNNnn` and `\tl_act_aux:NNNnn`. These functions are documented on page ??.)

```

\tl_reverse_tokens:n
\tl_act_reverse_normal:nN
\tl_act_reverse_group:nn
\tl_act_reverse_space:n

```

The goal is to reverse a token list. This is done by feeding `\tl_act_aux:NNNnn` three functions, an empty fourth argument (we don't use it for `\tl_act_reverse_tokens:n`), and as a fifth argument the token list to be reversed. Spaces and normal tokens are output to the left of the current output. For groups, we must recursively apply `\tl_act_reverse_tokens:n` to the group, and output, still on the left. Note that in all three cases, we throw one argument away: this *⟨parameter⟩* is where for instance the upper/lowercasing action stores the information of whether it is uppercasing or lowercasing.

```

4905 \cs_new:Npn \tl_reverse_tokens:n
4906 {
4907   \tex_romannumeral:D
4908   \tl_act_aux:NNNnn
4909   \tl_act_reverse_normal:nN
4910   \tl_act_reverse_group:nn
4911   \tl_act_reverse_space:n
4912   { }
4913 }
4914 \cs_new:Npn \tl_act_reverse_space:n #1
4915 { \tl_act_reverse_output:n {~} }
4916 \cs_new:Npn \tl_act_reverse_normal:nN #1 #2
4917 { \tl_act_reverse_output:n {#2} }
4918 \cs_new:Npn \tl_act_reverse_group:nn #1
4919 {
4920   \tl_act_group_recurse:Nnn
4921   \tl_act_reverse_output:n
4922   { \tl_reverse_tokens:n }
4923 }

```

(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 4924 \cs_new:Npn \tl_reverse:n
\tl_reverse_group_preserve:nn 4925 {
4926   \tex_romannumeral:D
4927   \tl_act_aux:NNNnn
4928   \tl_act_reverse_normal:nN
4929   \tl_act_reverse_group_preserve:nn
4930   \tl_act_reverse_space:n
4931   { }
4932 }
4933 \cs_new:Npn \tl_act_reverse_group_preserve:nn #1 #2
4934 { \tl_act_reverse_output:n { {#2} } }
4935 \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 `{}` in front, which in turn is removed by the `\unexpanded`
`\tl_reverse:c` primitive.

```

4936 \cs_new_protected_nopar:Npn \tl_reverse:N #1
4937 { \tl_set:No #1 { \etex_unexpanded:D \tl_reverse:o { #1 { } } } }
4938 \cs_generate_variant:Nn \tl_reverse:N { c }

```

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

`\tl_length_tokens:n` The length is computed through an `\int_eval:n` construction. Each `1+` is output to
`\tl_act_length_normal:nN` the *left*, into the integer expression, and the sum is ended by the `\c_zero` inserted by
`\tl_act_length_group:nn` `\tl_act_end:wn`. Somewhat a hack.
`\tl_act_length_space:n`


```

4939 \cs_new:Npn \tl_length_tokens:n #1
4940 {
4941   \int_eval:n
4942   {
4943     \tl_act_aux:NNNnn
4944     \tl_act_length_normal:nN
4945     \tl_act_length_group:nn
4946     \tl_act_length_space:n
4947     { }
4948     {#1}
4949   }
4950 }
4951 \cs_new:Npn \tl_act_length_normal:nN #1 #2 { 1 + }
4952 \cs_new:Npn \tl_act_length_space:n #1 { 1 + }
4953 \cs_new:Npn \tl_act_length_group:nn #1 #2
4954 { 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
`\c_tl_act_lowercase_tl` the form `aAbBcC...` and `AaBbCc...` respectively.

```

4955 \tl_const:Nn \c_tl_act_uppercase_tl
4956 {
4957   aA bB cC dD eE fF gG hH iI jJ kK lL mM
4958   nN oO pP qQ rR sS tT uU vV wW xX yY zZ
4959 }
4960 \tl_const:Nn \c_tl_act_lowercase_tl
4961 {
4962   Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm
4963   Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz
4964 }

```

(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
`\tl_expandable_lowercase:n` 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
`\tl_act_case_group:nn` applying. A space is simply output. A normal token is compared to each letter in
`\tl_act_case_space:n` 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.

```

4965 \cs_new:Npn \tl_expandable_uppercase:n
4966 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_uppercase_tl } }
4967 \cs_new:Npn \tl_expandable_lowercase:n
4968 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_lowercase_tl } }
4969 \cs_new:Npn \tl_act_case_aux:nn
4970 {
4971   \tl_act_aux:NNNnn
4972   \tl_act_case_normal:nN

```

```

4973     \tl_act_case_group:nn
4974     \tl_act_case_space:n
4975   }
4976   \cs_new:Npn \tl_act_case_space:n #1 { \tl_act_output:n {~} }
4977   \cs_new:Npn \tl_act_case_normal:nN #1 #2
4978   {
4979     \exp_args:Nf \tl_act_output:n
4980     {
4981       \exp_args:NNo \prg_case_str:nnn #2 {#1}
4982       { \exp_stop_f: #2 }
4983     }
4984   }
4985   \cs_new:Npn \tl_act_case_group:nn #1 #2
4986   {
4987     \exp_after:wN \tl_act_output:n \exp_after:wN
4988     { \exp_after:wN { \tex_romannumeral:D \tl_act_case_aux:nn {#1} {#2} } }
4989   }

```

(End definition for `\tl_expandable_uppercase:n` and `\tl_expandable_lowercase:n`. These functions are documented on page ??.)

188.15 Deprecated functions

`\tl_new:Nn` Use either `\tl_const:Nn` or `\tl_new:N`.

```

\tl_new:cn 4990 <*deprecated>
\tl_new:Nx 4991 \cs_new_protected:Npn \tl_new:Nn #1#2
4992 {
4993   \tl_new:N #1
4994   \tl_gset:Nn #1 {#2}
4995 }
4996 \cs_generate_variant:Nn \tl_new:Nn { c }
4997 \cs_generate_variant:Nn \tl_new:Nn { Nx }
4998 </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 4999 <*deprecated>
5000 \cs_new_protected_nopar:Npn \tl_gset:Nc
5001 { \tex_global:D \tl_set:Nc }
5002 \cs_new_protected_nopar:Npn \tl_set:Nc #1#2
5003 { \tl_set:No #1 { \cs:w #2 \cs_end: } }
5004 </deprecated>

```

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

`\tl_replace_in:Nnn` These are renamed.

```

\tl_replace_in:cnn 5005 <*deprecated>
\tl_greplace_in:Nnn 5006 \cs_new_eq:NN \tl_replace_in:Nnn \tl_replace_once:Nnn
\tl_greplace_in:cnn 5007 \cs_new_eq:NN \tl_replace_in:cnn \tl_replace_once:cnn
\tl_replace_all_in:Nnn 5008 \cs_new_eq:NN \tl_greplace_in:Nnn \tl_greplace_once:Nnn
\tl_replace_all_in:cnn
\tl_greplace_all_in:Nnn
\tl_greplace_all_in:cnn

```

```

5009 \cs_new_eq:NN \tl_greplace_in:cnn \tl_greplace_once:cnn
5010 \cs_new_eq:NN \tl_replace_all_in:Nnn \tl_replace_all:Nnn
5011 \cs_new_eq:NN \tl_replace_all_in:cnn \tl_replace_all:cnn
5012 \cs_new_eq:NN \tl_greplace_all_in:Nnn \tl_greplace_all:Nnn
5013 \cs_new_eq:NN \tl_greplace_all_in:cnn \tl_greplace_all:cnn
5014 \</deprecat

```

(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 5015 \*deprecat
\tl_gremove_in:Nn 5016 \cs_new_eq:NN \tl_remove_in:Nn \tl_remove_once:Nn
\tl_gremove_in:cn 5017 \cs_new_eq:NN \tl_remove_in:cn \tl_remove_once:cn
\tl_remove_all_in:Nn 5018 \cs_new_eq:NN \tl_gremove_in:Nn \tl_gremove_once:Nn
\tl_remove_all_in:cn 5019 \cs_new_eq:NN \tl_gremove_in:cn \tl_gremove_once:cn
\tl_gremove_all_in:Nn 5020 \cs_new_eq:NN \tl_remove_all_in:Nn \tl_remove_all:Nn
\tl_gremove_all_in:cn 5021 \cs_new_eq:NN \tl_remove_all_in:cn \tl_remove_all:cn
5022 \cs_new_eq:NN \tl_gremove_all_in:Nn \tl_gremove_all:Nn
5023 \cs_new_eq:NN \tl_gremove_all_in:cn \tl_gremove_all:cn
5024 \</deprecat

```

(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 5025 \*deprecat
\tl_elt_count:o 5026 \cs_new_eq:NN \tl_elt_count:n \tl_length:n
\tl_elt_count:N 5027 \cs_new_eq:NN \tl_elt_count:V \tl_length:V
\tl_elt_count:c 5028 \cs_new_eq:NN \tl_elt_count:o \tl_length:o
5029 \cs_new_eq:NN \tl_elt_count:N \tl_length:N
5030 \cs_new_eq:NN \tl_elt_count:c \tl_length:c
5031 \</deprecat

```

(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 5032 \*deprecat
\tl_head_iii:n 5033 \cs_new_eq:NN \tl_head_i:n \tl_head:n
\tl_head_iii:f 5034 \cs_new_eq:NN \tl_head_i:w \tl_head:w
\tl_head_iii:w 5035 \cs_new:Npn \tl_head_iii:n #1 { \tl_head_iii:w #1 \q_stop }
5036 \cs_generate_variant:Nn \tl_head_iii:n { f }
5037 \cs_new:Npn \tl_head_iii:w #1#2#3#4 \q_stop {#1#2#3}
5038 \</deprecat

```

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

```

5039 \</initex | package)

```

189 l3seq implementation

The following test files are used for this code: *m3seq002,m3seq003*.

```

5040 <*initex | package>
5041 <*package>
5042 \ProvidesExplPackage
5043   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5044 \package_check_loaded_expl:
5045 </package>

```

A sequence is a control sequence whose top-level expansion is of the form “\seq_item:n {<item₀>} ... \seq_item:n {<item_{n-1}>}”. An earlier implementation used the structure “\seq_elt:w <item₁> \seq_elt_end: ... \seq_elt:w <item_n> \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.

```

5046 \cs_new:Npn \seq_item:n
5047 {
5048   \msg_expandable_error:n { A-sequence-was-used-incorrectly. }
5049   \use_none:n
5050 }

```

(End definition for \seq_item:n. This function is documented on page 104.)

\l_seq_tmpa_tl Scratch space for various internal uses.

```

5051 \tl_new:N \l_seq_tmpa_tl
5052 \tl_new:N \l_seq_tmpb_tl

```

(End definition for \l_seq_tmpa_tl and \l_seq_tmpb_tl. These functions are documented on page ??.)

189.1 Allocation and initialisation

\seq_new:N Internally, sequences are just token lists.

```

5053 \cs_new_eq:NN \seq_new:N \tl_new:N
5054 \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.

```

5055 \cs_new_eq:NN \seq_clear:N \tl_clear:N
5056 \cs_new_eq:NN \seq_clear:c \tl_clear:c
5057 \cs_new_eq:NN \seq_gclear:N \tl_gclear:N
5058 \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 5059 \cs_new_eq:NN \seq_clear_new:N \tl_clear_new:N
\seq_gclear_new:N 5060 \cs_new_eq:NN \seq_clear_new:c \tl_clear_new:c
\seq_gclear_new:c 5061 \cs_new_eq:NN \seq_gclear_new:N \tl_gclear_new:N
5062 \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 5063 \cs_new_eq:NN \seq_set_eq:NN \tl_set_eq:NN
\seq_set_eq:Nc 5064 \cs_new_eq:NN \seq_set_eq:Nc \tl_set_eq:Nc
\seq_set_eq:cc 5065 \cs_new_eq:NN \seq_set_eq:cN \tl_set_eq:cN
\seq_gset_eq:NN 5066 \cs_new_eq:NN \seq_set_eq:cc \tl_set_eq:cc
\seq_gset_eq:cN 5067 \cs_new_eq:NN \seq_gset_eq:NN \tl_gset_eq:NN
\seq_gset_eq:Nc 5068 \cs_new_eq:NN \seq_gset_eq:Nc \tl_gset_eq:Nc
\seq_gset_eq:cN 5069 \cs_new_eq:NN \seq_gset_eq:cN \tl_gset_eq:cN
5070 \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_concat:NNN` Concatenating sequences is easy.

```

\seq_concat:ccc 5071 \cs_new_protected_nopar:Npn \seq_concat:NNN #1#2#3
\seq_gconcat:NNN 5072 { \tl_set:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
\seq_gconcat:ccc 5073 \cs_new_protected_nopar:Npn \seq_gconcat:NNN #1#2#3
5074 { \tl_gset:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
5075 \cs_generate_variant:Nn \seq_concat:NNN { ccc }
5076 \cs_generate_variant:Nn \seq_gconcat:NNN { ccc }

```

(End definition for `\seq_concat:NNN` and `\seq_concat:ccc`. These functions are documented on page ??.)

189.2 Appending data to either end

`\seq_put_left:Nn` The code here is just a wrapper for adding to token lists.

```

\seq_put_left:NV 5077 \cs_new_protected:Npn \seq_put_left:Nn #1#2
\seq_put_left:Nv 5078 { \tl_put_left:Nn #1 { \seq_item:n {#2} } }
\seq_put_left:No 5079 \cs_new_protected:Npn \seq_put_right:Nn #1#2
\seq_put_left:Nx 5080 { \tl_put_right:Nn #1 { \seq_item:n {#2} } }
\seq_put_left:cn 5081 \cs_generate_variant:Nn \seq_put_left:Nn { NV , Nv , No , Nx }
\seq_put_left:cV 5082 \cs_generate_variant:Nn \seq_put_left:Nn { c , cV , cv , co , cx }
\seq_put_left:cV 5083 \cs_generate_variant:Nn \seq_put_right:Nn { NV , Nv , No , Nx }
\seq_put_left:co 5084 \cs_generate_variant:Nn \seq_put_right:Nn { c , cV , cv , co , cx }
\seq_put_left:cX

```

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

`\seq_put_right:Nn` The same for global addition.

```

\seq_gput_left:Nn 5085 \cs_new_protected:Npn \seq_gput_left:Nn #1#2
\seq_gput_left:NV 5086 { \tl_gput_left:Nn #1 { \seq_item:n {#2} } }
\seq_gput_left:Nv 5087 \cs_new_protected:Npn \seq_gput_right:Nn #1#2
\seq_gput_left:No 5088 { \tl_gput_right:Nn #1 { \seq_item:n {#2} } }
\seq_gput_left:Nx 5089 \cs_generate_variant:Nn \seq_gput_left:Nn { NV , Nv , No , Nx }
\seq_gput_left:cn
\seq_gput_left:cV
\seq_gput_left:cV
\seq_gput_left:cV
\seq_gput_left:co
\seq_gput_left:cX
\seq_gput_right:Nn
\seq_gput_right:NV
\seq_gput_right:Nv
\seq_gput_right:No
\seq_gput_right:Nx
\seq_gput_right:cn

```

```

5090 \cs_generate_variant:Nn \seq_gput_left:Nn { c , cV , cv , co , cx }
5091 \cs_generate_variant:Nn \seq_gput_right:Nn {      NV , Nv , No , Nx }
5092 \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 ??.)

```

189.3 Modifying sequences

`\l_seq_remove_seq` An internal sequence for the removal routines.

```

5093 \seq_new:N \l_seq_remove_seq
      (End definition for \l_seq_remove_seq. This function is documented on page ??.)

```

`\seq_remove_duplicates:N` Removing duplicates means making a new list then copying it.

```

\seq_remove_duplicates:c
\seq_gremove_duplicates:N
\seq_gremove_duplicates:c
      \seq_remove_duplicates_aux:NN
5094 \cs_new_protected:Npn \seq_remove_duplicates:N
5095   { \seq_remove_duplicates_aux:NN \seq_set_eq:NN }
5096 \cs_new_protected:Npn \seq_gremove_duplicates:N
5097   { \seq_remove_duplicates_aux:NN \seq_gset_eq:NN }
5098 \cs_new_protected:Npn \seq_remove_duplicates_aux:NN #1#2
5099   {
5100     \seq_clear:N \l_seq_remove_seq
5101     \seq_map_inline:Nn #2
5102     {
5103       \seq_if_in:NnF \l_seq_remove_seq {##1}
5104       { \seq_put_right:Nn \l_seq_remove_seq {##1} }
5105     }
5106     #1 #2 \l_seq_remove_seq
5107   }
5108 \cs_generate_variant:Nn \seq_remove_duplicates:N { c }
5109 \cs_generate_variant:Nn \seq_gremove_duplicates:N { c }
      (End definition for \seq_remove_duplicates:N and \seq_remove_duplicates:c. These functions
are documented on page ??.)

```

`\seq_remove_all:Nn` The idea of the code here is to avoid a relatively expensive addition of items one at a time
`\seq_remove_all:cn` to an intermediate sequence. The approach taken is therefore similar to that in `\seq_`
`\seq_gremove_all:Nn` `pop_right_aux_ii:NNN`, using a “flexible” x-type expansion to do most of the work.
`\seq_gremove_all:cn` As `\tl_if_eq:nnT` is not expandable, a two-part strategy is needed. First, the x-type
`\seq_remove_all_aux:NNn` expansion uses `\str_if_eq:nnT` to find potential matches. If one is found, the expansion
is halted and the necessary set up takes place to use the `\tl_if_eq:NNT` test. The x-type
is started again, including all of the items copied already. This will happen repeatedly
until the entire sequence has been scanned. The code is set up to avoid needing and
intermediate scratch list: the lead-off x-type expansion (`#1 #2 {#2}`) will ensure that
nothing is lost.

```

5110 \cs_new_protected:Npn \seq_remove_all:Nn
5111   { \seq_remove_all_aux:NNn \tl_set:Nx }
5112 \cs_new_protected:Npn \seq_gremove_all:Nn
5113   { \seq_remove_all_aux:NNn \tl_gset:Nx }
5114 \cs_new_protected:Npn \seq_remove_all_aux:NNn #1#2#3
5115   {
5116     \seq_push_item_def:n

```

```

5117 {
5118   \str_if_eq:nnT {##1} {#3}
5119   {
5120     \if_false: { \fi: }
5121     \tl_set:Nn \l_seq_tmpb_tl {##1}
5122     #1 #2
5123     { \if_false: } \fi:
5124     \exp_not:o {#2}
5125     \tl_if_eq:NNT \l_seq_tmpa_tl \l_seq_tmpb_tl
5126     { \use_none:nn }
5127   }
5128   \exp_not:n { \seq_item:n {##1} }
5129 }
5130 \tl_set:Nn \l_seq_tmpa_tl {#3}
5131 #1 #2 {#2}
5132 \seq_pop_item_def:
5133 }
5134 \cs_generate_variant:Nn \seq_remove_all:Nn { c }
5135 \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 ??.)

189.4 Sequence conditionals

`\seq_if_empty:N` Simple copies from the token list variable material.

```

\seq_if_empty:c 5136 \prg_new_eq_conditional:Nn \seq_if_empty:N \tl_if_empty:N
5137 { p , T , F , TF }
5138 \prg_new_eq_conditional:Nn \seq_if_empty:c \tl_if_empty:c
5139 { 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 sequence. If the two items are equal, the mapping is terminated and `\prg_return_true:` is inserted. On the other hand, if there is no match then the loop will break returning `\prg_return_false:`. In either case, `\seq_break_point:n` ensures that the group ends before the logical value is returned. Everything is inside a group so that `\seq_item:n` is preserved in nested situations.

```

\seq_if_in:cV 5140 \prg_new_protected_conditional:Npnn \seq_if_in:Nn #1#2
\seq_if_in:cv 5141 { T , F , TF }
\seq_if_in:co 5142 {
\seq_if_in:cx 5143   \group_begin:
\seq_if_in_aux: 5144   \tl_set:Nn \l_seq_tmpa_tl {#2}
5145   \cs_set_protected:Npn \seq_item:n ##1
5146   {
5147     \tl_set:Nn \l_seq_tmpb_tl {##1}
5148     \if_meaning:w \l_seq_tmpa_tl \l_seq_tmpb_tl
5149     \exp_after:wN \seq_if_in_aux:
5150     \fi:

```

```

5151     }
5152     #1
5153     \seq_break:n { \prg_return_false: }
5154     \seq_break_point:n { \group_end: }
5155 }
5156 \cs_new_nopar:Npn \seq_if_in_aux: { \seq_break:n { \prg_return_true: } }
5157 \cs_generate_variant:Nn \seq_if_in:NnT { c , cV , cv , co , cx }
5158 \cs_generate_variant:Nn \seq_if_in:NnT { NV , Nv , No , Nx }
5159 \cs_generate_variant:Nn \seq_if_in:NnF { c , cV , cv , co , cx }
5160 \cs_generate_variant:Nn \seq_if_in:NnF { NV , Nv , No , Nx }
5161 \cs_generate_variant:Nn \seq_if_in:NnTF { c , cV , cv , co , cx }
5162 \cs_generate_variant:Nn \seq_if_in:NnTF { NV , Nv , No , Nx }

```

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

189.5 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
5163 \cs_new_protected_nopar:Npn \seq_get_left:NN #1#2
5164 {
5165   \seq_if_empty_err_break:N #1
5166   \exp_after:wN \seq_get_left_aux:NnwN #1 \q_stop #2
5167   \seq_break_point:n { }
5168 }
5169 \cs_new_protected:Npn \seq_get_left_aux:NnwN \seq_item:n #1#2 \q_stop #3
5170 { \tl_set:Nn #3 {#1} }
5171 \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
5172 \cs_new_protected_nopar:Npn \seq_pop_left:NN
5173 { \seq_pop_left_aux:NNN \tl_set:Nn }
5174 \cs_new_protected_nopar:Npn \seq_gpop_left:NN
5175 { \seq_pop_left_aux:NNN \tl_gset:Nn }
5176 \cs_new_protected_nopar:Npn \seq_pop_left_aux:NNN #1#2#3
5177 {
5178   \seq_if_empty_err_break:N #2
5179   \exp_after:wN \seq_pop_left_aux:NnwNNN #2 \q_stop #1#2#3
5180   \seq_break_point:n { }
5181 }
5182 \cs_new_protected:Npn \seq_pop_left_aux:NnwNNN \seq_item:n #1#2 \q_stop #3#4#5
5183 {
5184   #3 #4 {#2}
5185   \tl_set:Nn #5 {#1}
5186 }
5187 \cs_generate_variant:Nn \seq_pop_left:NN { c }

```



```
5188 \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.

```
5189 \cs_new_protected_nopar:Npn \seq_get_right:NN #1#2
5190 {
5191   \seq_if_empty_err_break:N #1
5192   \seq_get_right_aux:NN #1#2
5193   \seq_break_point:n { }
5194 }
5195 \cs_new_protected_nopar:Npn \seq_get_right_aux:NN #1#2
5196 {
5197   \seq_push_item_def:n { }
5198   \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5199   \exp_after:wN \use_none:n #1
5200   { \tl_set:Nn #2 }
5201   { }
5202   {
5203     \seq_pop_item_def:
5204     \seq_break:
5205   }
5206 }
5207 \cs_new:Npn \seq_get_right_loop:nn #1#2
5208 {
5209   #2 {#1}
5210   \seq_get_right_loop:nn
5211 }
5212 \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.

```
5213 \cs_new_protected_nopar:Npn \seq_pop_right:NN
5214 { \seq_pop_right_aux:NNN \tl_set:Nx }
```

```

5215 \cs_new_protected_nopar:Npn \seq_gpop_right:NN
5216 { \seq_pop_right_aux:NNN \tl_gset:Nx }
5217 \cs_new_protected_nopar:Npn \seq_pop_right_aux:NNN #1#2#3
5218 {
5219   \seq_if_empty_err_break:N #2
5220   \seq_pop_right_aux_ii:NNN #1 #2 #3
5221   \seq_break_point:n { }
5222 }
5223 \cs_new_protected_nopar:Npn \seq_pop_right_aux_ii:NNN #1#2#3
5224 {
5225   \seq_push_item_def:n { \exp_not:n { \seq_item:n {##1} } }
5226   #1 #2 { \if_false: } \fi:
5227   \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5228   \exp_after:wN \use_none:n #2
5229   {
5230     \if_false: { \fi: }
5231     \tl_set:Nn #3
5232   }
5233   { }
5234   {
5235     \seq_pop_item_def:
5236     \seq_break:
5237   }
5238 }
5239 \cs_generate_variant:Nn \seq_pop_right:NN { c }
5240 \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 ??.)

189.6 Mapping to sequences

`\seq_break:` To break a function, the special token `\seq_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.

```

5241 \cs_new:Npn \seq_break: #1 \seq_break_point:n #2 {#2}
5242 \cs_new:Npn \seq_break:n #1#2 \seq_break_point:n #3 { #3 #1 }

```

(End definition for `\seq_break:`. This function is documented on page 105.)

`\seq_map_break:` Semantically-logical copies of the break functions for use inside mappings.

```

5243 \cs_new_eq:NN \seq_map_break: \seq_break:
5244 \cs_new_eq:NN \seq_map_break:n \seq_break:n

```

(End definition for `\seq_map_break:`. This function is documented on page 101.)

`\seq_break_point:n` Normally, the marker token will not be executed, but if it is then the end code is simply inserted.

```

5245 \cs_new_eq:NN \seq_break_point:n \use:n

```

(End definition for `\seq_break_point:n`. This function is documented on page 105.)

`\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.

```

5246 \cs_new_protected_nopar:Npn \seq_if_empty_err_break:N #1
5247 {
5248   \if_meaning:w #1 \c_empty_tl
5249     \msg_kernel_error:nxx { seq } { empty-sequence } { \token_to_str:N #1 }
5250     \exp_after:wN \seq_break:
5251   \fi:
5252 }

```

(End definition for \seq_if_empty_err_break:N. This function is documented on page 104.)

`\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.

`\seq_map_function:cN`

`\seq_map_function_aux:NNn`

```

5253 \cs_new:Npn \seq_map_function:NN #1#2
5254 {
5255   \exp_after:wN \seq_map_function_aux:NNn \exp_after:wN #2 #1
5256   { ? \seq_map_break: } { }
5257   \seq_break_point:n { }
5258 }
5259 \cs_new:Npn \seq_map_function_aux:NNn #1#2#3
5260 {
5261   \use_none:n #2
5262   #1 {#3}
5263   \seq_map_function_aux:NNn #1
5264 }
5265 \cs_generate_variant:Nn \seq_map_function:NN { c }

```

(End definition for \seq_map_function:NN and \seq_map_function:cN. These functions are documented on page ??.)

`\g_seq_nesting_depth_int` A counter to keep track of nested functions: defined in `l3int`.

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

`\seq_push_item_def:n` The definition of `\seq_item:n` needs to be saved and restored at various points within the mapping and manipulation code. That is handled here: as always, this approach uses global assignments.

`\seq_push_item_def:x`

`\seq_push_item_def_aux:`

`\seq_pop_item_def:`

```

5266 \cs_new_protected:Npn \seq_push_item_def:n
5267 {
5268   \seq_push_item_def_aux:
5269   \cs_gset:Npn \seq_item:n ##1
5270 }
5271 \cs_new_protected:Npn \seq_push_item_def:x
5272 {
5273   \seq_push_item_def_aux:
5274   \cs_gset:Npx \seq_item:n ##1
5275 }
5276 \cs_new_protected:Npn \seq_push_item_def_aux:

```

```

5277 {
5278   \cs_gset_eq:cN { seq_item_ \int_use:N \g_seq_nesting_depth_int :n }
5279   \seq_item:n
5280   \int_gincr:N \g_seq_nesting_depth_int
5281 }
5282 \cs_new_protected_nopar:Npn \seq_pop_item_def:
5283 {
5284   \int_gdecr:N \g_seq_nesting_depth_int
5285   \cs_gset_eq:Nc \seq_item:n
5286   { seq_item_ \int_use:N \g_seq_nesting_depth_int :n }
5287 }

```

(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`.

```

5288 \cs_new_protected:Npn \seq_map_inline:Nn #1#2
5289 {
5290   \seq_push_item_def:n {#2}
5291   #1
5292   \seq_break_point:n { \seq_pop_item_def: }
5293 }
5294 \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`

```

5295 \cs_new_protected:Npn \seq_map_variable:NNn #1#2#3
5296 {
5297   \seq_push_item_def:x
5298   {
5299     \tl_set:Nn \exp_not:N #2 {##1}
5300     \exp_not:n {#3}
5301   }
5302   #1
5303   \seq_break_point:n { \seq_pop_item_def: }
5304 }
5305 \cs_generate_variant:Nn \seq_map_variable:NNn { Nc }
5306 \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 ??.)

189.7 Sequence stacks

The same functions as for sequences, but with the correct naming.

`\seq_push:Nn` Pushing to a sequence is the same as adding on the left.

<code>\seq_push:Nv</code>	5307	<code>\cs_new_eq:NN \seq_push:Nn \seq_put_left:Nn</code>
<code>\seq_push:No</code>	5308	<code>\cs_new_eq:NN \seq_push:Nv \seq_put_left:Nv</code>
<code>\seq_push:Nx</code>	5309	<code>\cs_new_eq:NN \seq_push:No \seq_put_left:No</code>
<code>\seq_push:cn</code>	5310	<code>\cs_new_eq:NN \seq_push:Nx \seq_put_left:Nx</code>
<code>\seq_push:cV</code>	5311	<code>\cs_new_eq:NN \seq_push:cn \seq_put_left:cn</code>
<code>\seq_push:cV</code>	5312	<code>\cs_new_eq:NN \seq_push:cV \seq_put_left:cV</code>
<code>\seq_push:co</code>	5313	<code>\cs_new_eq:NN \seq_push:cV \seq_put_left:cv</code>
<code>\seq_push:co</code>	5314	<code>\cs_new_eq:NN \seq_push:co \seq_put_left:co</code>
<code>\seq_push:cx</code>	5315	<code>\cs_new_eq:NN \seq_push:co \seq_put_left:co</code>
<code>\seq_gpush:Nn</code>	5316	<code>\cs_new_eq:NN \seq_gpush:Nn \seq_gput_left:Nn</code>
<code>\seq_gpush:Nv</code>	5317	<code>\cs_new_eq:NN \seq_gpush:Nv \seq_gput_left:Nv</code>
<code>\seq_gpush:No</code>	5318	<code>\cs_new_eq:NN \seq_gpush:No \seq_gput_left:No</code>
<code>\seq_gpush:Nx</code>	5319	<code>\cs_new_eq:NN \seq_gpush:Nx \seq_gput_left:Nx</code>
<code>\seq_gpush:cn</code>	5320	<code>\cs_new_eq:NN \seq_gpush:cn \seq_gput_left:cn</code>
<code>\seq_gpush:cV</code>	5321	<code>\cs_new_eq:NN \seq_gpush:cV \seq_gput_left:cV</code>
<code>\seq_gpush:cv</code>	5322	<code>\cs_new_eq:NN \seq_gpush:cv \seq_gput_left:cv</code>
<code>\seq_gpush:co</code>	5323	<code>\cs_new_eq:NN \seq_gpush:co \seq_gput_left:co</code>
<code>\seq_gpush:co</code>	5324	<code>\cs_new_eq:NN \seq_gpush:co \seq_gput_left:co</code>
<code>\seq_gpush:cx</code>	5325	<code>\cs_new_eq:NN \seq_gpush:cx \seq_gput_left:cx</code>
<code>\seq_gpush:cx</code>	5326	<code>\cs_new_eq:NN \seq_gpush:cx \seq_gput_left:cx</code>

(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
`\seq_get:cN` left. So alias are provided.

<code>\seq_pop:NN</code>	5327	<code>\cs_new_eq:NN \seq_get:NN \seq_get_left:NN</code>
<code>\seq_pop:cN</code>	5328	<code>\cs_new_eq:NN \seq_get:cN \seq_get_left:cN</code>
<code>\seq_gpop:NN</code>	5329	<code>\cs_new_eq:NN \seq_pop:NN \seq_pop_left:NN</code>
<code>\seq_gpop:cN</code>	5330	<code>\cs_new_eq:NN \seq_pop:cN \seq_pop_left:cN</code>
	5331	<code>\cs_new_eq:NN \seq_gpop:NN \seq_gpop_left:NN</code>
	5332	<code>\cs_new_eq:NN \seq_gpop:cN \seq_gpop_left:cN</code>

(End definition for \seq_get:NN and \seq_get:cN. These functions are documented on page ??.)

189.8 Viewing sequences

`\l_seq_show_tl` Used to store the material for display.

5333	<code>\tl_new:N \l_seq_show_tl</code>
------	---------------------------------------

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

`\seq_show:N` The aim of the mapping here is to create a token list containing the formatted sequence.
`\seq_show:c` The very first item needs the new line and `>\` removing, which is achieved using a `w`-type
`\seq_show_aux:n` auxiliary. To avoid a low-level `TeX` error if there is an empty sequence, a simple test is
`\seq_show_aux:w` used to keep the output “clean”.

5334	<code>\cs_new_protected_nopar:Npn \seq_show:N #1</code>
5335	<code>{</code>
5336	<code>\seq_if_empty:NTF #1</code>
5337	<code>{</code>
5338	<code>\iow_term:x { Sequence~\token_to_str:N #1~is~empty }</code>

```

5339     \tl_show:n { }
5340   }
5341   {
5342     \iow_term:x
5343     {
5344       Sequence~\token_to_str:N #1~
5345       contains~the~items~(without~outer~braces):
5346     }
5347     \tl_set:Nx \l_seq_show_tl
5348     { \seq_map_function:NN #1 \seq_show_aux:n }
5349     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
5350     { \exp_after:wN \seq_show_aux:w \l_seq_show_tl }
5351   }
5352 }
5353 \cs_new:Npn \seq_show_aux:n #1
5354 {
5355   \iow_newline: > \c_space_tl \c_space_tl
5356   \iow_char:N {\ \exp_not:n {#1} \iow_char:N \}
5357 }
5358 \cs_new:Npn \seq_show_aux:w #1 > ~ { }
5359 \cs_generate_variant:Nn \seq_show:N { c }

```

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

189.9 Experimental functions

`\seq_if_empty_break_return_false:N` The name says it all: of the sequence is empty, returns logical false.

```

5360 \cs_new_nopar:Npn \seq_if_empty_break_return_false:N #1
5361 {
5362   \if_meaning:w #1 \c_empty_tl
5363   \prg_return_false:
5364   \exp_after:wN \seq_break:
5365   \fi:
5366 }

```

(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 5367 \prg_new_protected_conditional:Npnn \seq_get_left:NN #1 #2 { T , F , TF }
\seq_get_right:NN 5368 {
\seq_get_right:cN 5369   \seq_if_empty_break_return_false:N #1
5370   \exp_after:wN \seq_get_left_aux:Nw #1 \q_stop #2
5371   \prg_return_true:
5372   \seq_break:
5373   \seq_break_point:n { }
5374 }
5375 \prg_new_protected_conditional:Npnn \seq_get_right:NN #1#2 { T , F , TF }
5376 {
5377   \seq_if_empty_break_return_false:N #1
5378   \seq_get_right_aux:NN #1#2

```

```

5379 \prg_return_true: \seq_break:
5380 \seq_break_point:n { }
5381 }

```

```

5382 \cs_generate_variant:Nn \seq_get_left:NNT { c }
5383 \cs_generate_variant:Nn \seq_get_left:NNF { c }
5384 \cs_generate_variant:Nn \seq_get_left:NNTF { c }
5385 \cs_generate_variant:Nn \seq_get_right:NNT { c }
5386 \cs_generate_variant:Nn \seq_get_right:NNF { c }
5387 \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
\seq_gpop_left:NN 5388 \prg_new_protected_conditional:Npnn \seq_pop_left:NN #1#2 { T , F , TF }
\seq_gpop_left:cN 5389 {
\seq_if_empty_break_return_false:N #1
\seq_pop_right:NN 5390 \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_set:Nn #1#2
\seq_pop_right:cN 5391 \prg_return_true: \seq_break:
\seq_gpop_right:NN 5392 \seq_break_point:n { }
\seq_gpop_right:cN 5393 }
5394 \prg_new_protected_conditional:Npnn \seq_gpop_left:NN #1#2 { T , F , TF }
5395 {
5396 \seq_if_empty_break_return_false:N #1
5397 \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_gset:Nn #1#2
5398 \prg_return_true: \seq_break:
5399 \seq_break_point:n { }
5400 }
5401 \prg_new_protected_conditional:Npnn \seq_pop_right:NN #1#2 { T , F , TF }
5402 {
5403 \seq_if_empty_break_return_false:N #1
5404 \seq_pop_right_aux_ii:NNN \tl_set:Nx #1 #2
5405 \prg_return_true: \seq_break:
5406 \seq_break_point:n { }
5407 }
5408 \prg_new_protected_conditional:Npnn \seq_gpop_right:NN #1#2 { T , F , TF }
5409 {
5410 \seq_if_empty_break_return_false:N #1
5411 \seq_pop_right_aux_ii:NNN \tl_gset:Nx #1 #2
5412 \prg_return_true: \seq_break:
5413 \seq_break_point:n { }
5414 }
5415 \cs_generate_variant:Nn \seq_pop_left:NNT { c }
5416 \cs_generate_variant:Nn \seq_pop_left:NNF { c }
5417 \cs_generate_variant:Nn \seq_pop_left:NNTF { c }
5418 \cs_generate_variant:Nn \seq_gpop_left:NNT { c }
5419 \cs_generate_variant:Nn \seq_gpop_left:NNF { c }
5420 \cs_generate_variant:Nn \seq_gpop_left:NNTF { c }
5421 \cs_generate_variant:Nn \seq_pop_right:NNT { c }
5422 \cs_generate_variant:Nn \seq_pop_right:NNF { c }
5423 \cs_generate_variant:Nn \seq_pop_right:NNTF { c }
5424 \cs_generate_variant:Nn \seq_gpop_right:NNTF { c }

```

```

5425 \cs_generate_variant:Nn \seq_gpop_right:NNT { c }
5426 \cs_generate_variant:Nn \seq_gpop_right:NNF { c }
5427 \cs_generate_variant:Nn \seq_gpop_right:NNTF { c }

```

(End definition for `\seq_pop_left:NN` and `\seq_pop_left:cN`. These functions are documented on page ??.)

`\seq_length:N` Counting the items in a sequence is done using the same approach as for other length functions: turn each entry into a +1 then use integer evaluation to actually do the mathematics.
`\seq_length:c`
`\seq_length_aux:n`

```

5428 \cs_new:Npn \seq_length:N #1
5429 {
5430   \int_eval:n
5431   {
5432     0
5433     \seq_map_function:NN #1 \seq_length_aux:n
5434   }
5435 }
5436 \cs_new:Npn \seq_length_aux:n #1 { +1 }
5437 \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 } { }` will be used by the auxiliary, terminating the loop and returning nothing at all.
`\seq_item:cn`
`\seq_item_aux:nnn`

```

5438 \cs_new_nopar:Npn \seq_item:Nn #1#2
5439 {
5440   \exp_last_unbraced:Nfo \seq_item_aux:nnn
5441   {
5442     \int_eval:n
5443     {
5444       \int_compare:nNnT {#2} < \c_zero
5445       { \seq_length:N #1 + }
5446       #2
5447     }
5448   }
5449   #1
5450   { ? \seq_break: }
5451   { }
5452   \seq_break_point:n { }
5453 }
5454 \cs_new_nopar:Npn \seq_item_aux:nnn #1#2#3
5455 {
5456   \use_none:n #2
5457   \int_compare:nNnTF {#1} = \c_zero
5458   { \seq_break:n {#3} }
5459   { \exp_args:Nf \seq_item_aux:nnn { #1 - 1 } }
5460 }
5461 \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` 5462 `\cs_new_nopar:Npn \seq_use:N #1 { \seq_map_function:NN #1 \use:n }`
 5463 `\cs_generate_variant:Nn \seq_use:N { c }`

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

`\seq_mapthread_function:NNN` The idea here is to first expand both of the sequences, adding the usual `{ ? \seq_break: } { }`
`\seq_mapthread_function:NcN` to the end of each on. This is most conveniently done in two steps using an auxiliary
`\seq_mapthread_function:cNN` function. The mapping then throws away the first token of #2 and #5, which for items
`\seq_mapthread_function:ccN` in the sequences will both be `\seq_item:n`. The function to be mapped will then be
`\seq_mapthread_function_aux:NN` applied to the two entries. When the code hits the end of one of the sequences, the break
`\seq_mapthread_function_aux:Nnnwnn` material will stop the entire loop and tidy up. This avoids needing to find the length of
 the two sequences, or worrying about which is longer.

5464 `\cs_new_nopar:Npn \seq_mapthread_function:NNN #1#2#3`
 5465 `{`
 5466 `\exp_after:wN \seq_mapthread_function_aux:NN`
 5467 `\exp_after:wN #3`
 5468 `\exp_after:wN #1`
 5469 `#2`
 5470 `{ ? \seq_break: } { }`
 5471 `\seq_break_point:n { }`
 5472 `}`
 5473 `\cs_new_nopar:Npn \seq_mapthread_function_aux:NN #1#2`
 5474 `{`
 5475 `\exp_after:wN \seq_mapthread_function_aux:Nnnwnn`
 5476 `\exp_after:wN #1`
 5477 `#2`
 5478 `{ ? \seq_break: } { }`
 5479 `\q_stop`
 5480 `}`
 5481 `\cs_new:Npn \seq_mapthread_function_aux:Nnnwnn #1#2#3#4 \q_stop #5#6`
 5482 `{`
 5483 `\use_none:n #2`
 5484 `\use_none:n #5`
 5485 `#1 {#3} {#6}`
 5486 `\seq_mapthread_function_aux:Nnnwnn #1 #4 \q_stop`
 5487 `}`
 5488 `\cs_generate_variant:Nn \seq_mapthread_function:NNN { Nc }`
 5489 `\cs_generate_variant:Nn \seq_mapthread_function:NNN { c , cc }`

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

`\seq_set_from_clist:NN` Setting a sequence from a comma-separated list is done using a simple mapping.

`\seq_set_from_clist:cN` 5490 `\cs_new_protected:Npn \seq_set_from_clist:NN #1#2`
`\seq_set_from_clist:Nc` 5491 `{`
`\seq_set_from_clist:cc` 5492 `\tl_set:Nx #1`
`\seq_set_from_clist:Nn` 5493 `{ \clist_map_function:NN #2 \seq_wrap_item:n }`
`\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`
`\seq_wrap_item:n`

```

5494 }
5495 \cs_new_protected:Npn \seq_set_from_clist:Nn #1#2
5496 {
5497   \tl_set:Nx #1
5498     { \clist_map_function:nN {#2} \seq_wrap_item:n }
5499 }
5500 \cs_new_protected:Npn \seq_gset_from_clist:NN #1#2
5501 {
5502   \tl_gset:Nx #1
5503     { \clist_map_function:NN #2 \seq_wrap_item:n }
5504 }
5505 \cs_new_protected:Npn \seq_gset_from_clist:Nn #1#2
5506 {
5507   \tl_gset:Nx #1
5508     { \clist_map_function:nN {#2} \seq_wrap_item:n }
5509 }
5510 \cs_new:Npn \seq_wrap_item:n #1 { \exp_not:n { \seq_item:n {#1} } }
5511 \cs_generate_variant:Nn \seq_set_from_clist:NN { Nc }
5512 \cs_generate_variant:Nn \seq_set_from_clist:NN { c , cc }
5513 \cs_generate_variant:Nn \seq_set_from_clist:Nn { c }
5514 \cs_generate_variant:Nn \seq_gset_from_clist:NN { Nc }
5515 \cs_generate_variant:Nn \seq_gset_from_clist:NN { c , cc }
5516 \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_set_reverse:N` Define `\seq_item:n` to place its argument after a marker, `\prg_do_nothing:.` Then
`\seq_gset_reverse:N` x-expand the sequence.

```

5517 \cs_new_protected_nopar:Npn \seq_tmp:w
5518 { \msg_expandable_error:n { There-is-a-bug-in-LaTeX3! } }
5519 \cs_new_protected_nopar:Npn \seq_set_reverse:N
5520 { \seq_reverse_aux:NN \tl_set:Nx }
5521 \cs_new_protected_nopar:Npn \seq_gset_reverse:N
5522 { \seq_reverse_aux:NN \tl_gset:Nx }
5523 \cs_new_protected_nopar:Npn \seq_reverse_aux:NN #1 #2
5524 {
5525   \cs_set_eq:NN \seq_tmp:w \seq_item:n
5526   \cs_set_eq:NN \seq_item:n \seq_reverse_aux_item:w
5527   #1 #2 { #2 \prg_do_nothing: }
5528   \cs_set_eq:NN \seq_item:n \seq_tmp:w
5529 }
5530 \cs_new:Npn \seq_reverse_aux_item:w #1 #2 \prg_do_nothing:
5531 {
5532   #2
5533   \prg_do_nothing:
5534   \exp_not:n { \seq_item:n {#1} }
5535 }

```

(End definition for `\seq_set_reverse:N` and `\seq_gset_reverse:N`. These functions are documented on page 104.)

`\seq_set_split:Nnn` The goal is to split a given token list at a marker, strip spaces from each item, and
`\seq_gset_split:Nnn` remove one set of outer braces if after removing leading and trailing spaces the item
`\seq_set_split_aux:NNnn` is enclosed within braces. After `\tl_replace_all:Nnn`, the token list `\l_seq_tmpa_tl`
`\seq_set_split_aux_i:w` is a repetition of the pattern `\seq_set_split_aux_i:w \prg_do_nothing: <item with`
`\seq_set_split_aux_ii:w` `spaces> \seq_set_split_aux_end:.` Then, x-expansion causes `\seq_set_split_aux_-`
`\seq_set_split_aux_end:` `i:w` to trim spaces, and leaves its result as `\seq_set_split_aux_ii:w <trimmed item>`
`\seq_set_split_aux_end:.` This is then converted to the `l3seq` internal structure by
another x-expansion. In the first step, we insert `\prg_do_nothing:` to avoid losing
braces too early: that would cause space trimming to act within those lost braces. The
second step is solely there to strip braces which are outermost after space trimming.

```

5536 \cs_new_protected_nopar:Npn \seq_set_split:Nnn
5537 { \seq_set_split_aux:NNnn \tl_set:Nx }
5538 \cs_new_protected_nopar:Npn \seq_gset_split:Nnn
5539 { \seq_set_split_aux:NNnn \tl_gset:Nx }
5540 \cs_new_protected_nopar:Npn \seq_set_split_aux:NNnn #1 #2 #3 #4
5541 {
5542   \tl_if_empty:nTF {#4}
5543   { #1 #2 { } }
5544   {
5545     \tl_set:Nn \l_seq_tmpa_tl
5546     {
5547       \seq_set_split_aux_i:w \prg_do_nothing:
5548       #4
5549       \seq_set_split_aux_end:
5550     }
5551     \tl_replace_all:Nnn \l_seq_tmpa_tl { #3 }
5552     {
5553       \seq_set_split_aux_end:
5554       \seq_set_split_aux_i:w \prg_do_nothing:
5555     }
5556     \tl_set:Nx \l_seq_tmpa_tl { \l_seq_tmpa_tl }
5557     #1 #2 { \l_seq_tmpa_tl }
5558   }
5559 }
5560 \cs_new:Npn \seq_set_split_aux_i:w #1 \seq_set_split_aux_end:
5561 {
5562   \exp_not:N \seq_set_split_aux_ii:w
5563   \exp_args:No \tl_trim_spaces:n {#1}
5564   \exp_not:N \seq_set_split_aux_end:
5565 }
5566 \cs_new:Npn \seq_set_split_aux_ii:w #1 \seq_set_split_aux_end:
5567 { \exp_not:n { \seq_item:n {#1} } }

```

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

189.10 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 5568 <*deprecated>
5569 \cs_new_eq:NN \seq_top:NN \seq_get_left:NN
5570 \cs_new_eq:NN \seq_top:cN \seq_get_left:cN
5571 </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 5572 <*deprecated>
5573 \cs_new_eq:NN \seq_display:N \seq_show:N
5574 \cs_new_eq:NN \seq_display:c \seq_show:c
5575 </deprecated>
(End definition for \seq_display:N and \seq_display:c. These functions are documented on
page ??.)
5576 </initex | package>
```

190 l3clist implementation

The following test files are used for this code: `m3clist002`.

```
5577 <*initex | package>
5578 <*package>
5579 \ProvidesExplPackage
5580 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
5581 \package_check_loaded_expl:
5582 </package>
```

`\l_clist_tmpa_tl` Scratch space for various internal uses.

```
5583 \tl_new:N \l_clist_tmpa_tl
(End definition for \l_clist_tmpa_tl. This function is documented on page ??.)
```

`\clist_tmp:w` A temporary function for various purposes.

```
5584 \cs_new_protected:Npn \clist_tmp:w { }
(End definition for \clist_tmp:w. This function is documented on page ??.)
```

190.1 Allocation and initialisation

`\clist_new:N` Internally, comma lists are just token lists.

`\clist_new:c` 5585 `\cs_new_eq:NN \clist_new:N \tl_new:N`
 5586 `\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` 5587 `\cs_new_eq:NN \clist_clear:N \tl_clear:N`
`\clist_gclear:N` 5588 `\cs_new_eq:NN \clist_clear:c \tl_clear:c`
`\clist_gclear:c` 5589 `\cs_new_eq:NN \clist_gclear:N \tl_gclear:N`
 5590 `\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` 5591 `\cs_new_eq:NN \clist_clear_new:N \tl_clear_new:N`
`\clist_gclear_new:N` 5592 `\cs_new_eq:NN \clist_clear_new:c \tl_clear_new:c`
`\clist_gclear_new:c` 5593 `\cs_new_eq:NN \clist_gclear_new:N \tl_gclear_new:N`
 5594 `\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` 5595 `\cs_new_eq:NN \clist_set_eq:NN \tl_set_eq:NN`
`\clist_set_eq:Nc` 5596 `\cs_new_eq:NN \clist_set_eq:Nc \tl_set_eq:Nc`
`\clist_set_eq:cc` 5597 `\cs_new_eq:NN \clist_set_eq:cN \tl_set_eq:cN`
`\clist_gset_eq:NN` 5598 `\cs_new_eq:NN \clist_set_eq:cc \tl_set_eq:cc`
`\clist_gset_eq:cN` 5599 `\cs_new_eq:NN \clist_gset_eq:NN \tl_gset_eq:NN`
`\clist_gset_eq:Nc` 5600 `\cs_new_eq:NN \clist_gset_eq:Nc \tl_gset_eq:Nc`
`\clist_gset_eq:cN` 5601 `\cs_new_eq:NN \clist_gset_eq:cN \tl_gset_eq:cN`
`\clist_gset_eq:cc` 5602 `\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` 5603 `\cs_new_protected_nopar:Npn \clist_concat:NNN`
`\clist_gconcat:ccc` 5604 `{ \clist_concat_aux:NNNN \tl_set:Nx }`
`\clist_concat_aux:NNNN` 5605 `\cs_new_protected_nopar:Npn \clist_gconcat:NNN`
 5606 `{ \clist_concat_aux:NNNN \tl_gset:Nx }`
 5607 `\cs_new_protected_nopar:Npn \clist_concat_aux:NNNN #1#2#3#4`
 5608 `{`
 5609 `#1 #2`
 5610 `{`
 5611 `\exp_not:o #3`
 5612 `\clist_if_empty:NF #3 { \clist_if_empty:NF #4 { , } }`
 5613 `\exp_not:o #4`
 5614 `}`

```

5615 }
5616 \cs_generate_variant:Nn \clist_concat:NNN { ccc }
5617 \cs_generate_variant:Nn \clist_gconcat:NNN { ccc }

```

(End definition for `\clist_concat:NNN` and `\clist_concat:ccc`. These functions are documented on page ??.)

190.2 Removing spaces around items

`\clist_trim_spaces_generic:nw` Used as ‘`\clist_trim_spaces_generic:nw {<code>} \q_mark <item> ,`’ (including the comma). This expands to the `<code>`, followed by a brace group containing the `<item>`, with leading and trailing spaces removed. The calling function is responsible for inserting `\q_mark` in front of the `<item>`, as well as testing for the end of the list. See `\tl_trim_spaces:n` for a partial explanation of what is happening here. We changed `\tl_trim_spaces_aux_iv:w` into `\clist_trim_spaces_generic_aux:w` compared to `\tl_trim_spaces:n`, and dropped a `\q_mark`, which is already included in the argument `##2`.

```

5618 \cs_set:Npn \clist_tmp:w #1
5619 {
5620   \cs_new:Npn \clist_trim_spaces_generic:nw ##1 ##2 ,
5621   {
5622     \tl_trim_spaces_aux_i:w
5623     ##2
5624     \q_nil
5625     \q_mark #1 { }
5626     \q_mark \tl_trim_spaces_aux_ii:w
5627     \tl_trim_spaces_aux_iii:w
5628     #1 \q_nil
5629     \clist_trim_spaces_generic_aux:w
5630     \q_stop
5631     {##1}
5632   }
5633 }
5634 \clist_tmp:w {~}
5635 \cs_new:Npn \clist_trim_spaces_generic_aux:w #1 \q_nil #2 \q_stop
5636 { \exp_args:No \clist_trim_spaces_generic_aux_ii:nn { \use_none:n #1 } }
5637 \cs_new:Npn \clist_trim_spaces_generic_aux_ii:nn #1 #2 { #2 {#1} }

```

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

`\clist_trim_spaces:n` The argument of `\clist_trim_spaces_aux:n` is initially empty, and later a comma, namely, as soon as we have added an item to the resulting list. The second auxiliary tests for the end of the list, and also prevents empty arguments from finding their way into the output.

```

5638 \cs_new:Npn \clist_trim_spaces:n #1
5639 {
5640   \clist_trim_spaces_aux:n { } \q_mark #1 ,
5641   \q_recursion_tail, \q_recursion_stop
5642 }
5643 \cs_new:Npn \clist_trim_spaces_aux:n #1
5644 {

```

```

5645 \clist_trim_spaces_generic:nw
5646 { \clist_trim_spaces_aux_ii:nn {#1} }
5647 }
5648 \cs_new:Npn \clist_trim_spaces_aux_ii:nn #1 #2
5649 {
5650 \quark_if_recursion_tail_stop:n {#2}
5651 \tl_if_empty:nTF {#2}
5652 {
5653 \clist_trim_spaces_aux:n {#1} \q_mark
5654 }
5655 {
5656 #1 \exp_not:n {#2}
5657 \clist_trim_spaces_aux:n { , } \q_mark
5658 }
5659 }

```

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

190.3 Adding data to comma lists

```

\clist_set:Nn
\clist_set:NV 5660 \cs_new_protected:Npn \clist_set:Nn #1#2
\clist_set:No 5661 { \tl_set:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:Nx 5662 \cs_new_protected:Npn \clist_gset:Nn #1#2
\clist_set:cn 5663 { \tl_gset:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:cV 5664 \cs_generate_variant:Nn \clist_set:Nn { NV , No , Nx , c , cV , co , cx }
\clist_set:co 5665 \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
5666 \cs_new_protected_nopar:Npn \clist_put_left:Nn
5667 { \clist_put_aux:NnnNn \tl_set_eq:NN \tl_put_left:Nx { } , }
5668 \cs_new_protected_nopar:Npn \clist_gput_left:Nn
5669 { \clist_put_aux:NnnNn \tl_gset_eq:NN \tl_gput_left:Nx { } , }
5670 \cs_new_protected:Npn \clist_put_aux:NnnNn #1#2#3#4#5#6
5671 {
5672 \tl_set:Nx \l_clist_tmpa_tl { \clist_trim_spaces:n {#6} }
5673 \clist_if_empty:NTF #5
5674 { #1 #5 \l_clist_tmpa_tl }
5675 {
5676 \tl_if_empty:NF \l_clist_tmpa_tl
5677 { #2 #5 { #3 \exp_not:o \l_clist_tmpa_tl #4 } }
5678 }
5679 }
5680 \cs_generate_variant:Nn \clist_put_left:Nn { NV , No , Nx }
5681 \cs_generate_variant:Nn \clist_put_left:Nn { c , cV , co , cx }
5682 \cs_generate_variant:Nn \clist_gput_left:Nn { NV , No , Nx }
5683 \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 ??.)

<code>\clist_push:Nn</code>	Pushing to a sequence is the same as adding on the left.
<code>\clist_push:Nv</code>	5715 <code>\cs_new_eq:NN \clist_push:Nn \clist_put_left:Nn</code>
<code>\clist_push:No</code>	5716 <code>\cs_new_eq:NN \clist_push:Nv \clist_put_left:Nv</code>
<code>\clist_push:Nx</code>	5717 <code>\cs_new_eq:NN \clist_push:No \clist_put_left:No</code>
<code>\clist_push:cn</code>	5718 <code>\cs_new_eq:NN \clist_push:Nx \clist_put_left:Nx</code>
<code>\clist_push:cV</code>	5719 <code>\cs_new_eq:NN \clist_push:cn \clist_put_left:cn</code>
<code>\clist_push:co</code>	5720 <code>\cs_new_eq:NN \clist_push:cV \clist_put_left:cV</code>
<code>\clist_push:cx</code>	5721 <code>\cs_new_eq:NN \clist_push:co \clist_put_left:co</code>
<code>\clist_gpush:Nn</code>	5722 <code>\cs_new_eq:NN \clist_gpush:cx \clist_put_left:cx</code>
<code>\clist_gpush:Nv</code>	5723 <code>\cs_new_eq:NN \clist_gpush:Nn \clist_gput_left:Nn</code>
<code>\clist_gpush:No</code>	5724 <code>\cs_new_eq:NN \clist_gpush:Nv \clist_gput_left:Nv</code>
<code>\clist_gpush:No</code>	5725 <code>\cs_new_eq:NN \clist_gpush:No \clist_gput_left:No</code>
<code>\clist_gpush:Nx</code>	5726 <code>\cs_new_eq:NN \clist_gpush:Nx \clist_gput_left:Nx</code>
<code>\clist_gpush:cn</code>	5727 <code>\cs_new_eq:NN \clist_gpush:cn \clist_gput_left:cn</code>
<code>\clist_gpush:cV</code>	5728 <code>\cs_new_eq:NN \clist_gpush:cV \clist_gput_left:cV</code>
<code>\clist_gpush:co</code>	5729 <code>\cs_new_eq:NN \clist_gpush:co \clist_gput_left:co</code>
<code>\clist_gpush:cx</code>	5730 <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 ??.)

190.5 Using comma lists

<code>\clist_use:N</code>	The approach is the same as for <code>\tl_use:N</code> .
<code>\clist_use:c</code>	5731 <code>\cs_new_eq:NN \clist_use:N \tl_use:N</code>
	5732 <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 ??.)

190.6 Modifying comma lists

<code>\l_clist_remove_clist</code>	An internal comma list for the removal routines.
	5733 <code>\clist_new:N \l_clist_remove_clist</code>

(End definition for `\l_clist_remove_clist`. This function is documented on page ??.)

<code>\clist_remove_duplicates:N</code>	Removing duplicates means making a new list then copying it.
<code>\clist_remove_duplicates:c</code>	5734 <code>\cs_new_protected:Npn \clist_remove_duplicates:N</code>
<code>\clist_gremove_duplicates:N</code>	5735 <code>{ \clist_remove_duplicates_aux:NN \clist_set_eq:NN }</code>
<code>\clist_gremove_duplicates:c</code>	5736 <code>\cs_new_protected:Npn \clist_gremove_duplicates:N</code>
<code>\clist_remove_duplicates_aux:NN</code>	5737 <code>{ \clist_remove_duplicates_aux:NN \clist_gset_eq:NN }</code>
	5738 <code>\cs_new_protected:Npn \clist_remove_duplicates_aux:NN #1#2</code>
	5739 <code>{</code>
	5740 <code>\clist_clear:N \l_clist_remove_clist</code>
	5741 <code>\clist_map_inline:Nn #2</code>
	5742 <code>{</code>
	5743 <code>\clist_if_in:NnF \l_clist_remove_clist {##1}</code>
	5744 <code>{ \clist_put_right:Nn \l_clist_remove_clist {##1} }</code>
	5745 <code>}</code>
	5746 <code>#1 #2 \l_clist_remove_clist</code>
	5747 <code>}</code>

```

5748 \cs_generate_variant:Nn \clist_remove_duplicates:N { c }
5749 \cs_generate_variant:Nn \clist_gremove_duplicates:N { c }
      (End definition for \clist_remove_duplicates:N and \clist_remove_duplicates:c. These func-
tions 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.

```

5750 \cs_new_protected:Npn \clist_remove_all:Nn
5751   { \clist_remove_all_aux:NNn \tl_set:Nx }
5752 \cs_new_protected:Npn \clist_gremove_all:Nn
5753   { \clist_remove_all_aux:NNn \tl_gset:Nx }
5754 \cs_new_protected:Npn \clist_remove_all_aux:NNn #1#2#3
5755   {
5756     \cs_set:Npn \clist_tmp:w ##1 , #3 ,
5757     {
5758       ##1
5759       , \q_mark , \use_none_delimit_by_q_stop:w ,
5760       \clist_remove_all_aux:
5761     }
5762     #1 #2
5763     {
5764       \exp_after:wN \clist_remove_all_aux:
5765       #2 , \q_mark , #3 , \q_stop
5766     }
5767     \clist_if_empty:NF #2
5768     {
5769       #1 #2
5770       {
5771         \exp_args:No \exp_not:o
5772         { \exp_after:wN \use_none:n #2 }
5773       }
5774     }
5775   }
5776 \cs_new:Npn \clist_remove_all_aux:
5777   { \exp_after:wN \clist_remove_all_aux:w \clist_tmp:w , }
5778 \cs_new:Npn \clist_remove_all_aux:w #1 , \q_mark , #2 , { \exp_not:n {#1} }

```

```

5779 \cs_generate_variant:Nn \clist_remove_all:Nn { c }
5780 \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 ??.)

190.7 Comma list conditionals

\l_clist_if_in_clist An internal comma list for \clist_if_in:nn conditionals.

```

5781 \clist_new:N \l_clist_if_in_clist

```

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

\clist_if_empty:N Simple copies from the token list variable material.

```

\clist_if_empty:c
5782 \prg_new_eq_conditional:NNn \clist_if_empty:N \tl_if_empty:N { p , T , F , TF }
5783 \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
5784 \prg_new_eq_conditional:NNn \clist_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\clist_if_eq:cN
5785 \prg_new_eq_conditional:NNn \clist_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
\clist_if_eq:cc
5786 \prg_new_eq_conditional:NNn \clist_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
5787 \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
\clist_if_in:No
5788 \prg_new_protected_conditional:Npnn \clist_if_in:Nn #1#2 { T , F , TF }
\clist_if_in:cn
5789 {
\clist_if_in:cV
5790 \exp_args:No \clist_if_in_return:nn #1 {#2}
\clist_if_in:co
5791 }
\clist_if_in:nn
5792 \prg_new_protected_conditional:Npnn \clist_if_in:nn #1#2 { T , F , TF }
\clist_if_in:nV
5793 {
\clist_if_in:no
5794 \clist_set:Nn \l_clist_if_in_clist {#1}
5795 \exp_args:No \clist_if_in_return:nn \l_clist_if_in_clist {#2}
\clist_if_in_return:nn
5796 }
5797 \cs_new_protected:Npn \clist_if_in_return:nn #1#2
5798 {
5799 \cs_set:Npn \clist_tmp:w ##1 ,#2, { }
5800 \tl_if_empty:oTF
5801 { \clist_tmp:w ,#1, {} {} ,#2, }
5802 { \prg_return_false: } { \prg_return_true: }
5803 }
5804 \cs_generate_variant:Nn \clist_if_in:NnT { NV , No }
5805 \cs_generate_variant:Nn \clist_if_in:NnT { c , cV , co }
5806 \cs_generate_variant:Nn \clist_if_in:NnF { NV , No }
5807 \cs_generate_variant:Nn \clist_if_in:NnF { c , cV , co }
5808 \cs_generate_variant:Nn \clist_if_in:NnTF { NV , No }
5809 \cs_generate_variant:Nn \clist_if_in:NnTF { c , cV , co }

```

```

5810 \cs_generate_variant:Nn \clist_if_in:nnT { nV , no }
5811 \cs_generate_variant:Nn \clist_if_in:nnF { nV , no }
5812 \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 ??.)

190.8 Mapping to comma lists

\clist_map_function:NN Mapping to comma lists is pretty simple, if not massively efficient. The auxiliary function
\clist_map_function:cN \clist_map_function_aux:Nw is used directly in \clist_length:n. Change with care.
\clist_map_function:nN
\clist_map_function_aux:Nw

```

5813 \cs_new_nopar:Npn \clist_map_function:NN #1#2
5814 {
5815   \clist_if_empty:NF #1
5816   {
5817     \exp_last_unbraced:NNo \clist_map_function_aux:Nw #2 #1
5818     , \q_recursion_tail , \q_recursion_stop
5819   }
5820 }
5821 \cs_new:Npn \clist_map_function_aux:Nw #1#2 ,
5822 {
5823   \quark_if_recursion_tail_stop:n {#2}
5824   #1 {#2}
5825   \clist_map_function_aux:Nw #1
5826 }
5827 \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 ??.)

\g_clist_map_inline_int For the nesting of mappings.

```

5828 \int_new:N \g_clist_map_inline_int

```

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

\clist_map_inline:Nn Inline mapping is done by creating a suitable function “on the fly”: this is done globally
\clist_map_inline:cn to avoid any issues with T_EX’s groups.

```

\clist_map_inline:nn
5829 \cs_new_protected:Npn \clist_map_inline:Nn #1#2
5830 {
5831   \int_gincr:N \g_clist_map_inline_int
5832   \cs_gset:cpn { clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5833   ##1
5834   {#2}
5835   \exp_args:NNc \clist_map_function:NN #1
5836   { clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5837   \int_gdecr:N \g_clist_map_inline_int
5838 }
5839 \cs_new_protected:Npn \clist_map_inline:nn #1#2
5840 {
5841   \int_gincr:N \g_clist_map_inline_int
5842   \cs_gset:cpn { clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5843   ##1

```

```

5844     {#2}
5845     \exp_args:Nnc \clist_map_function:nN {#1}
5846     { \clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5847     \int_gdecr:N \g_clist_map_inline_int
5848   }
5849   \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 This is similar to the \clist_map_function:NN code. The n version is done by first trimming all spaces, then using the N version.

```

\clist_map_variable:cNn
\clist_map_variable:nNn
\clist_map_variable_aux:Nnw
5850 \cs_new_protected:Npn \clist_map_variable:NNn #1#2#3
5851 {
5852   \clist_if_empty:NF #1
5853   {
5854     \exp_args:Nno \use:nn
5855     { \clist_map_variable_aux:Nnw #2 {#3} }
5856     #1
5857     , \q_recursion_tail , \q_recursion_stop
5858   }
5859 }
5860 \cs_new_protected:Npn \clist_map_variable_aux:Nnw #1#2#3 ,
5861 {
5862   \quark_if_recursion_tail_stop:n {#3}
5863   \tl_set:Nn #1 {#3}
5864   #2
5865   \clist_map_variable_aux:Nnw #1 {#2}
5866 }
5867 \cs_new_protected:Npn \clist_map_variable:nNn #1
5868 {
5869   \tl_set:Nx \l_clist_tmpa_tl { \clist_trim_spaces:n {#1} }
5870   \clist_map_variable:NNn \l_clist_tmpa_tl
5871 }
5872 \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_aux_unbrace:Nw Within mappings with explicit n-type comma lists, braces must be stripped after space trimming. Here, #1 is the function to map, and #2 the item to brace-strip.

```

5873 \cs_new:Npn \clist_map_aux_unbrace:Nw #1 #2, { #1 {#2} }

```

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

\clist_map_function:nN 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 before brace stripping by \clist_map_aux_unbrace:Nw.

```

5874 \cs_new:Npn \clist_map_function:nN #1#2
5875 {
5876   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #2 }

```

```

5877 \q_mark #1, \q_recursion_tail, \q_recursion_stop
5878 }
5879 \cs_new:Npn \clist_map_function:n_aux:Nn #1 #2
5880 {
5881   \quark_if_recursion_tail_stop:n {#2}
5882   \tl_if_empty:nF {#2} { \clist_map_aux_unbrace:Nw #1 #2, }
5883   \clist_trim_spaces_generic:nw { \clist_map_function:n_aux:Nn #1 }
5884   \q_mark
5885 }

```

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

\clist_map_break: Both are simple renaming.

```

\clist_map_break:n 5886 \cs_new_eq:NN \clist_map_break: \use_none_delimit_by_q_recursion_stop:w
5887 \cs_new_eq:NN \clist_map_break:n \use_i_delimit_by_q_recursion_stop:nw
(End definition for \clist_map_break:. This function is documented on page 112.)

```

191 Viewing comma lists

\clist_show:N The aim of the mapping here is to create a token list containing the formatted comma list. The very first item needs the new line and >_ removing, which is achieved using a w-type auxiliary. To avoid a low-level T_EX error if there is an empty comma list, a simple test is used to keep the output “clean”.

```

5888 \cs_new_protected_nopar:Npn \clist_show:N #1
5889 {
5890   \clist_if_empty:NTF #1
5891   {
5892     \iow_term:x { Comma-list-\token_to_str:N #1-is-empty }
5893     \tl_show:n { }
5894   }
5895   {
5896     \iow_term:x
5897     {
5898       Comma-list-\token_to_str:N #1~
5899       contains~the~items~(without~outer~braces):
5900     }
5901     \tl_set:Nx \l_clist_show_tl
5902     { \clist_map_function:NN #1 \clist_show_aux:n }
5903     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
5904     { \exp_after:wN \clist_show_aux:w \l_clist_show_tl }
5905   }
5906 }
5907 \cs_new:Npn \clist_show_aux:n #1
5908 {
5909   \iow_newline: > \c_space_tl \c_space_tl
5910   \iow_char:N \{ \exp_not:n {#1} \iow_char:N \}
5911 }
5912 \cs_new:Npn \clist_show_aux:w #1 > ~ { }
5913 \cs_generate_variant:Nn \clist_show:N { c }

```

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

191.1 Scratch comma lists

`\l_tmpa_clist` Temporary comma list variables.

`\l_tmpb_clist` 5914 `\clist_new:N \l_tmpa_clist`

`\g_tmpa_tl` 5915 `\clist_new:N \l_tmpb_clist`

`\g_tmpb_tl` 5916 `\clist_new:N \g_tmpa_clist`

5917 `\clist_new:N \g_tmpb_clist`

(End definition for `\l_tmpa_clist` and `\l_tmpb_clist`. These functions are documented on page 94.)

191.2 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 call `\clist_map_function_aux:Nw` directly, and test for each item whether it is blank (hence should be ignored).

```
5918 \cs_new:Npn \clist_length:N #1
5919 {
5920   \int_eval:n
5921   {
5922     0
5923     \clist_map_function:NN #1 \clist_length_aux:n
5924   }
5925 }
5926 \cs_new:Npn \clist_length_aux:n #1 { +1 }
5927 \cs_new:Npn \clist_length:n #1
5928 {
5929   \int_eval:n
5930   {
5931     0
5932     \clist_map_function_aux:Nw \clist_length_n_aux:n #1
5933     , \q_recursion_tail , \q_recursion_stop
5934   }
5935 }
5936 \cs_new:Npn \clist_length_n_aux:n #1 { \tl_if_blank:nF {#1} {+1} }
5937 \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

`\clist_item:cn`

`\clist_item_aux:nnNn`

`\clist_item_N_loop:nw`

before performing the loop. The loop itself is very simple, return the item if the counter reached zero, otherwise, decrease the counter and repeat.

```

5938 \cs_new:Npn \clist_item:Nn #1#2
5939 {
5940   \exp_args:Nfo \clist_item_aux:nnNn
5941   { \clist_length:N #1 }
5942   #1
5943   \clist_item_N_loop:nw
5944   {#2}
5945 }
5946 \cs_new:Npn \clist_item_aux:nnNn #1#2#3#4
5947 {
5948   \int_compare:nNnTF {#4} < \c_zero
5949   {
5950     \int_compare:nNnTF {#4} < { - #1 }
5951     { \use_none_delimit_by_q_stop:w }
5952     { \exp_args:Nf #3 { \int_eval:n { #4 + #1 } } }
5953   }
5954   {
5955     \int_compare:nNnTF {#4} < {#1}
5956     { #3 {#4} }
5957     { \use_none_delimit_by_q_stop:w }
5958   }
5959   #2, \q_stop
5960 }
5961 \cs_new:Npn \clist_item_N_loop:nw #1 #2,
5962 {
5963   \int_compare:nNnTF {#1} = \c_zero
5964   { \use_i_delimit_by_q_stop:nw {#2} }
5965   { \exp_args:Nf \clist_item_N_loop:nw { \int_eval:n { #1 - 1 } } }
5966 }
5967 \cs_generate_variant:Nn \clist_item:Nn { c }

```

(End definition for \clist_item:Nn and \clist_item:cn. These functions are documented on page ??.)

<code>\clist_item:nn</code> <code>\clist_item_n_aux:nw</code> <code>\clist_item_n_loop:nw</code> <code>\clist_item_n_end:n</code> <code>\clist_item_n_strip:w</code>	<p>This starts in the same way as <code>\clist_item:Nn</code> by checking the length of the comma list. The final item should be space-trimmed before being brace-stripped, hence we insert a couple of odd-looking <code>\prg_do_nothing:</code> to avoid losing braces. Blank items are ignored.</p>
--	--

```

5968 \cs_new:Npn \clist_item:nn #1#2
5969 {
5970   \exp_args:Nf \clist_item_aux:nnNn
5971   { \clist_length:n {#1} }
5972   {#1}
5973   \clist_item_n_aux:nw
5974   {#2}
5975 }
5976 \cs_new:Npn \clist_item_n_aux:nw #1
5977 { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
5978 \cs_new:Npn \clist_item_n_loop:nw #1 #2,

```



```

5979 {
5980   \exp_args:No \tl_if_blank:nTF {#2}
5981   { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
5982   {
5983     \int_compare:nNnTF {#1} = \c_zero
5984     { \exp_args:No \clist_item_n_end:n {#2} }
5985     {
5986       \exp_args:Nf \clist_item_n_loop:nw
5987       { \int_eval:n { #1 - 1 } }
5988       \prg_do_nothing:
5989     }
5990   }
5991 }
5992 \cs_new:Npn \clist_item_n_end:n #1 #2 \q_stop
5993 {
5994   \exp_after:wN \exp_after:wN \exp_after:wN \clist_item_n_strip:w
5995   \tl_trim_spaces:n {#1} ,
5996 }
5997 \cs_new:Npn \clist_item_n_strip:w #1 , {#1}

```

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

\clist_set_from_seq:NN Setting a comma list from a comma-separated list is done using a simple mapping. We
\clist_set_from_seq:cN wrap most items with \exp_not:n, and a comma. Items which contain a comma or a
\clist_set_from_seq:Nc space are surrounded by an extra set of braces. The first comma must be removed, except
\clist_set_from_seq:cc in the case of an empty comma-list.

```

\clist_gset_from_seq:NN 5998 \cs_new_protected:Npn \clist_set_from_seq:NN
\clist_gset_from_seq:cN 5999 { \clist_set_from_seq_aux:NNNN \clist_clear:N \tl_set:Nx }
\clist_gset_from_seq:Nc 6000 \cs_new_protected:Npn \clist_gset_from_seq:NN
\clist_gset_from_seq:cc 6001 { \clist_set_from_seq_aux:NNNN \clist_gclear:N \tl_gset:Nx }
6002 \cs_new_protected:Npn \clist_set_from_seq_aux:NNNN #1#2#3#4
6003 {
6004   \seq_if_empty:NTF #4
6005   { #1 #3 }
6006   {
6007     \seq_push_item_def:n
6008     {
6009       ,
6010       \tl_if_empty:oTF { \clist_set_from_seq_aux:w #1 ~ , #1 ~ }
6011       { \exp_not:n {##1} }
6012       { \exp_not:n { {##1} } }
6013     }
6014     #2 #3 { \exp_last_unbraced:Nf \use_none:n #4 }
6015     \seq_pop_item_def:
6016   }
6017 }
6018 \cs_new:Npn \clist_set_from_seq_aux:w #1 , #2 ~ { }
6019 \cs_generate_variant:Nn \clist_set_from_seq:NN { Nc }
6020 \cs_generate_variant:Nn \clist_set_from_seq:NN { c , cc }
6021 \cs_generate_variant:Nn \clist_gset_from_seq:NN { Nc }

```

6022 `\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 ??.)

191.3 Deprecated interfaces

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\clist_top:NN` These are old stack functions.
`\clist_top:cN` 6023 `*deprecated`
6024 `\cs_new_eq:NN \clist_top:NN \clist_get:NN`
6025 `\cs_new_eq:NN \clist_top:cN \clist_get:cN`
6026 `*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` 6027 `*deprecated`
6028 `\cs_new_eq:NN \clist_remove_element:Nn \clist_remove_all:Nn`
6029 `\cs_new_eq:NN \clist_gremove_element:Nn \clist_gremove_all:Nn`
6030 `*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` 6031 `*deprecated`
6032 `\cs_new_eq:NN \clist_display:N \clist_show:N`
6033 `\cs_new_eq:NN \clist_display:c \clist_show:c`
6034 `*deprecated`
(End definition for \clist_display:N and \clist_display:c. These functions are documented on page ??.)
 Deprecated on 2011-09-05, for removal by 2011-12-32.

`\clist_trim_spaces:N` Since clist items are now always stripped from their surrounding spaces, it is redundant to provide these functions. The `\clist_trim_spaces:n` function is now internal, deprecated for use outside the kernel.
`\clist_trim_spaces:c`
`\clist_gtrim_spaces:N`
`\clist_gtrim_spaces:c` 6035 `\cs_new_protected:Npn \clist_trim_spaces:N #1 { \clist_set:No #1 {#1} }`
6036 `\cs_new_protected:Npn \clist_gtrim_spaces:N #1 { \clist_gset:No #1 {#1} }`
6037 `\cs_generate_variant:Nn \clist_trim_spaces:N { c }`
6038 `\cs_generate_variant:Nn \clist_gtrim_spaces:N { c }`
(End definition for \clist_trim_spaces:N and others. These functions are documented on page ??.)
6039 `*initex | package`

192 l3prop implementation

The following test files are used for this code: *m3prop001*.

```
6040 <*initex | package>
6041 <*package>
6042 \ProvidesExplPackage
6043   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
6044   \package_check_loaded_expl:
6045 </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.

```
6046 \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`.

```
6047 \tl_const:Nn \c_empty_prop { \q_prop }
(End definition for \c_empty_prop. This function is documented on page 120.)
```

192.1 Allocation and initialisation

`\prop_new:N` Internally, property lists are token lists, but an empty prop is not an empty tl, so we need to do things by hand.

```
6048 \cs_new_protected:Npn \prop_new:N #1 { \cs_new_eq:NN #1 \c_empty_prop }
6049 \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 6050 \cs_new_protected:Npn \prop_clear:N #1 { \cs_set_eq:NN #1 \c_empty_prop }
\prop_gclear:N 6051 \cs_new_protected:Npn \prop_clear:c #1 { \cs_set_eq:cN {#1} \c_empty_prop }
\prop_gclear:c 6052 \cs_new_protected:Npn \prop_gclear:N #1 { \cs_gset_eq:NN #1 \c_empty_prop }
6053 \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 6054 \cs_new_protected:Npn \prop_clear_new:N #1
\prop_gclear_new:N 6055   { \cs_if_exist:NTF #1 { \prop_clear:N #1 } { \prop_new:N #1 } }
\prop_gclear_new:c 6056 \cs_generate_variant:Nn \prop_clear_new:N {c}
6057 \cs_new_protected:Npn \prop_gclear_new:N #1
6058   { \cs_if_exist:NTF #1 { \prop_gclear:N #1 } { \prop_new:N #1 } }
6059 \cs_new_eq:NN \prop_gclear_new:c \prop_gclear: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 6060 \cs_new_eq:NN \prop_set_eq:NN \tl_set_eq:NN
\prop_set_eq:Nc 6061 \cs_new_eq:NN \prop_set_eq:Nc \tl_set_eq:Nc
\prop_set_eq:cc 6062 \cs_new_eq:NN \prop_set_eq:cN \tl_set_eq:cN
\prop_gset_eq:NN 6063 \cs_new_eq:NN \prop_set_eq:cc \tl_set_eq:cc
\prop_gset_eq:cN 6064 \cs_new_eq:NN \prop_gset_eq:NN \tl_gset_eq:NN
\prop_gset_eq:Nc 6065 \cs_new_eq:NN \prop_gset_eq:Nc \tl_gset_eq:Nc
\prop_gset_eq:cN 6066 \cs_new_eq:NN \prop_gset_eq:cN \tl_gset_eq:cN
\prop_gset_eq:cc 6067 \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 ??.)

192.2 Accessing data in property lists

`\prop_split:NnTF`
`\prop_split_aux:NnTF`
`\prop_split_aux:nnnn`
`\prop_split_aux:w`

This function is used by most of the module, and hence must be fast. The aim here is to split a property list at a given key into the part before the key–value pair, the value associated with the key and the part after the key–value pair. To do this, the key is first 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 *<extract1>**<extract2>* to be a well-formed property list, *<extract1>* has a leading and trailing `\q_prop`, retaining exactly the structure of a property list, while *<extract2>* omits the leading `\q_prop`.

If the key is not there, then *<extra argument>* is `? \use_ii:nn { }`, and `\prop_split_aux:nnnn ? \u` removes the three brace groups that just follow. Then `\use_ii:nn` removes the true branch, leaving the false branch, with no trailing material.

```

6068 \cs_set_protected:Npn \prop_split:NnTF #1#2
6069 { \exp_args:NNo \prop_split_aux:NnTF #1 { \tl_to_str:n {#2} } }
6070 \cs_new_protected:Npn \prop_split_aux:NnTF #1#2
6071 {
6072   \cs_set_protected:Npn \prop_split_aux:w
6073     ##1 \q_prop #2 \q_prop ##2 ##3 ##4 \q_mark ##5 \q_stop
6074   { \prop_split_aux:nnnn ##3 { {##1 \q_prop } {##2} {##4} } }
6075   \exp_after:wN \prop_split_aux:w #1 \q_mark
6076     \q_prop #2 \q_prop { } { ? \use_ii:nn { } } \q_mark \q_stop
6077 }
6078 \cs_new:Npn \prop_split_aux:nnnn #1#2#3#4 { #3 #2 }
6079 \cs_new_protected:Npn \prop_split_aux:w { }

```

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

`\prop_split:Nnn`

The goal here is to provide a common interface for both true and false branches of `\prop_split:NnTF`. In both cases, the code given by the user will be placed in front of three brace groups, $\{\langle extract_1 \rangle\} \{\langle value \rangle\} \{\langle extract_2 \rangle\}$. If the key was missing from the property list, then *<extract1>* is the full property list, *<value>* is `\q_no_value`, and

$\langle extract2 \rangle$ is empty. Otherwise, $\langle extract1 \rangle$ 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 $\langle extract2 \rangle$ (the part after the $\langle key \rangle$) is missing the leading $\backslash q_prop$.

```

6080 \cs_set_protected:Npn \prop_split:Nnn #1#2#3
6081 {
6082   \prop_split:NnTF #1 {#2}
6083   {#3}
6084   { \exp_args:Nno \use:n {#3} {#1} { \q_no_value } { } }
6085 }

```

(End definition for $\backslash prop_split:Nnn$. This function is documented on page 121.)

$\backslash prop_del:Nn$	Deleting from a property starts by splitting the list. If the key is present in the property
$\backslash prop_del:NV$	list, the returned value is ignored. If the key is missing, nothing happens.
$\backslash prop_del:cn$	
$\backslash prop_del:cV$	6086 \cs_new_protected:Npn \prop_del:Nn #1#2
$\backslash prop_gdel:Nn$	6087 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_set:Nn #1 } { } }
$\backslash prop_gdel:NV$	6088 \cs_new_protected:Npn \prop_gdel:Nn #1#2
$\backslash prop_gdel:cn$	6089 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_gset:Nn #1 } { } }
$\backslash prop_gdel:cV$	6090 \cs_new_protected:Npn \prop_gdel_aux:NNnnn #1#2#3#4#5
$\backslash prop_del_aux:NNnnn$	6091 { #1 #2 { #3 #5 } }
	6092 \cs_generate_variant:Nn \prop_del:Nn { NV }
	6093 \cs_generate_variant:Nn \prop_del:Nn { c , cV }
	6094 \cs_generate_variant:Nn \prop_gdel:Nn { NV }
	6095 \cs_generate_variant:Nn \prop_gdel:Nn { c , cV }

(End definition for $\backslash prop_del:Nn$ and others. These functions are documented on page ??.)

$\backslash prop_get:NnN$	Getting an item from a list is very easy: after splitting, if the key is in the property list,
$\backslash prop_get:NVN$	just set the token list variable to the return value, otherwise to $\backslash q_no_value$.
$\backslash prop_get:NoN$	
$\backslash prop_get:cnN$	6096 \cs_new_protected:Npn \prop_get:NnN #1#2#3
$\backslash prop_get:cVN$	6097 {
$\backslash prop_get:NoN$	6098 \prop_split:NnTF #1 {#2}
$\backslash prop_get_aux:NNnnn$	6099 { \prop_get_aux:Nnnn #3 }
	6100 { \tl_set:Nn #3 { \q_no_value } }
	6101 }
	6102 \cs_new_protected:Npn \prop_get_aux:NNnnn #1#2#3#4
	6103 { \tl_set:Nn #1 {#3} }
	6104 \cs_generate_variant:Nn \prop_get:NnN { NV , No }
	6105 \cs_generate_variant:Nn \prop_get:NnN { c , cV , co }

(End definition for $\backslash prop_get:NnN$ and others. These functions are documented on page ??.)

$\backslash prop_pop:NnN$	Popping a value also starts by doing the split. If the key is present, save the value in
$\backslash prop_pop:NoN$	the token list and update the property list as when deleting. If the key is missing, save
$\backslash prop_pop:cnN$	$\backslash q_no_value$ in the token list.
$\backslash prop_pop:coN$	
$\backslash prop_gpop:NnN$	6106 \cs_new_protected:Npn \prop_pop:NnN #1#2#3
$\backslash prop_gpop:NoN$	6107 {
$\backslash prop_gpop:cnN$	6108 \prop_split:NnTF #1 {#2}
$\backslash prop_gpop:coN$	6109 { \prop_pop_aux:NNnnnn \tl_set:Nn #1 #3 }
$\backslash prop_pop_aux:NNnnnn$	6110 { \tl_set:Nn #3 { \q_no_value } }
	6111 }

```

6112 \cs_new_protected:Npn \prop_gpop:NnN #1#2#3
6113 {
6114   \prop_split:NnTF #1 {#2}
6115   { \prop_pop_aux:NNNnnn \tl_gset:Nn #1 #3 }
6116   { \tl_set:Nn #3 { \q_no_value } }
6117 }
6118 \cs_new_protected:Npn \prop_pop_aux:NNNnnn #1#2#3#4#5#6
6119 {
6120   \tl_set:Nn #3 {#5}
6121   #1 #2 { #4 #6 }
6122 }
6123 \cs_generate_variant:Nn \prop_pop:NnN { No }
6124 \cs_generate_variant:Nn \prop_pop:NnN { c , co }
6125 \cs_generate_variant:Nn \prop_gpop:NnN { No }
6126 \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 value. The property list is then reconstructed with the two remaining parts #5 and #7 first, followed by the new or updated entry.

```

6127 \cs_new_protected:Npn \prop_put:Nnn { \prop_put_aux:NNnn \tl_set:Nx }
6128 \cs_new_protected:Npn \prop_gput:Nnn { \prop_put_aux:NNnn \tl_gset:Nx }
6129 \cs_new_protected:Npn \prop_put_aux:NNnn #1#2#3#4
6130 {
6131   \prop_split:Nnn #2 {#3} { \prop_put_aux:NNnnnnn #1 #2 {#3} {#4} }
6132 }
6133 \cs_new_protected:Npn \prop_put_aux:NNnnnnn #1#2#3#4#5#6#7
6134 {
6135   #1 #2
6136   {
6137     \exp_not:n { #5 #7 }
6138     \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop }
6139   }
6140 }
6141 \cs_generate_variant:Nn \prop_put:Nnn
6142 { NnV , Nno , Nnx , NV , NVV , No , Noo }
6143 \cs_generate_variant:Nn \prop_put:Nnn
6144 { c , cnV , cno , cnx , cV , cVV , co , coo }
6145 \cs_generate_variant:Nn \prop_gput:Nnn
6146 { NnV , Nno , Nnx , NV , NVV , No , Noo }
6147 \cs_generate_variant:Nn \prop_gput:Nnn
6148 { 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 given by \prop_split:NnTF are removed. If the key is new, then the value is added, being careful to convert the key to a string using \tl_to_str:n.

```

6149 \cs_new_protected_nopar:Npn \prop_put_if_new:Nnn
6150 { \prop_put_if_new_aux:NNnn \tl_put_right:Nx }

```

\prop_put_if_new:Nnn

```

6151 \cs_new_protected_nopar:Npn \prop_gput_if_new:Nnn
6152   { \prop_put_if_new_aux:NNnn \tl_gput_right:Nx }
6153 \cs_new_protected:Npn \prop_put_if_new_aux:NNnn #1#2#3#4
6154   {
6155     \prop_split:NnTF #2 {#3}
6156     { \use_none:nnn }
6157     {
6158       #1 #2
6159       { \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop } }
6160     }
6161   }
6162 \cs_generate_variant:Nn \prop_put_if_new:Nnn { c }
6163 \cs_generate_variant:Nn \prop_gput_if_new:Nnn { c }

```

(End definition for \prop_put_if_new:Nnn and \prop_put_if_new:cnn. These functions are documented on page ??.)

192.3 Property list conditionals

\prop_if_empty:N The test here uses \c_empty_prop as it is not really empty!

```

\prop_if_empty:c
6164 \prg_new_conditional:Npnn \prop_if_empty:N #1 { p , T , F , TF }
6165   {
6166     \if_meaning:w #1 \c_empty_prop
6167     \prg_return_true:
6168     \else:
6169     \prg_return_false:
6170     \fi:
6171   }
6172 \cs_generate_variant:Nn \prop_if_empty_p:N {c}
6173 \cs_generate_variant:Nn \prop_if_empty:N {NTF {c}}
6174 \cs_generate_variant:Nn \prop_if_empty:NT {c}
6175 \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 pairs one by one. This is rather slow, and a faster test would be

```

\prop_if_in:NV
\prop_if_in:No
\prop_if_in:cn
\prop_if_in:cV
\prop_if_in:co
\prop_if_in_aux:nwn
\prop_if_in_aux:Nw

```

```

\prg_new_protected_conditional:Npnn \prop_if_in:Nn #1 #2
{
  \prop_split:NnTF #1 {#2}
  {
    \prg_return_true:
    \use_none:nnn
  }
  { \prg_return_false: }
}

```

but \prop_split:NnTF is non-expandable.

Instead, the key is compared to each key in turn using `\str_if_eq:xx`, which is expandable. To terminate the mapping, we add the key that is search for at the end of the property list. This second `\tl_to_str:n` is not expanded at the start, but only when included in the `\str_if_eq:xx`. It cannot make the breaking mechanism choke, because the arbitrary token list material is enclosed in braces. When ending, we test the next token: it is either `\q_prop` or `\q_recursion_tail` in the case of a missing key. Here, `\prop_map_function:NN` is not sufficient for the mapping, since it can only map a single token, and cannot carry the key that is searched for.

```

6176 \prg_new_conditional:Npnn \prop_if_in:Nn #1#2 { p , T , F , TF }
6177 {
6178   \exp_last_unbraced:Noo \prop_if_in_aux:nwn
6179   { \tl_to_str:n {#2} } #1
6180   \tl_to_str:n {#2} \q_prop { }
6181   \q_recursion_tail \q_recursion_stop
6182 }
6183 \cs_new:Npn \prop_if_in_aux:nwn #1 \q_prop #2 \q_prop #3
6184 {
6185   \str_if_eq:xxTF {#1} {#2}
6186   { \prop_if_in_aux:Nw }
6187   { \prop_if_in_aux:nwn {#1} }
6188 }
6189 \cs_new:Npn \prop_if_in_aux:Nw #1 #2 \q_recursion_stop
6190 {
6191   \if_meaning:w \q_prop #1
6192   \prg_return_true:
6193   \else:
6194     \prg_return_false:
6195   \fi:
6196 }
6197 \cs_generate_variant:Nn \prop_if_in_p:Nn { NV , No }
6198 \cs_generate_variant:Nn \prop_if_in_p:Nn { c , cV , co }
6199 \cs_generate_variant:Nn \prop_if_in:NnT { NV , No }
6200 \cs_generate_variant:Nn \prop_if_in:NnT { c , cV , co }
6201 \cs_generate_variant:Nn \prop_if_in:NnF { NV , No }
6202 \cs_generate_variant:Nn \prop_if_in:NnF { c , cV , co }
6203 \cs_generate_variant:Nn \prop_if_in:NnTF { NV , No }
6204 \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 ??.)

192.4 Recovering values from property lists with branching

<code>\prop_get:NnN</code>	Getting the value corresponding to a key, keeping track of whether the key was present
<code>\prop_get:NVN</code>	or not, is implemented as a conditional (with side effects). If the key was absent, the
<code>\prop_get:NoN</code>	token list is not altered.
<code>\prop_get:cnN</code>	6205 \prg_new_protected_conditional:Npnn \prop_get:NnN #1#2#3 { T , F , TF }
<code>\prop_get:cVN</code>	6206 {
<code>\prop_get:coN</code>	6207 \prop_split:NnTF #1 {#2}
<code>\prop_get_aux_true:Nnnn</code>	6208 { \prop_get_aux_true:Nnnn #3 }


```

6209     { \prg_return_false: }
6210   }
6211   \cs_new_protected:Npn \prop_get_aux_true:Nnnn #1#2#3#4
6212   {
6213     \tl_set:Nn #1 {#3}
6214     \prg_return_true:
6215   }
6216   \cs_generate_variant:Nn \prop_get:NnNT { NV , No }
6217   \cs_generate_variant:Nn \prop_get:NnNF { NV , No }
6218   \cs_generate_variant:Nn \prop_get:NnNTF { NV , No }
6219   \cs_generate_variant:Nn \prop_get:NnNT { c , cV , co }
6220   \cs_generate_variant:Nn \prop_get:NnNF { c , cV , co }
6221   \cs_generate_variant:Nn \prop_get:NnNTF { c , cV , co }

```

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

192.5 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 strings, and thus cannot match the marker \q_recursion_tail.

\prop_map_function:Nc

\prop_map_function:cN

\prop_map_function:cc

\prop_map_function_aux:Nwn

```

6222   \cs_new_nopar:Npn \prop_map_function:NN #1#2
6223   {
6224     \exp_last_unbraced:Nno \prop_map_function_aux:Nwn #2
6225     #1 \q_recursion_tail \q_prop { } \q_recursion_stop
6226   }
6227   \cs_new:Npn \prop_map_function_aux:Nwn #1 \q_prop #2 \q_prop #3
6228   {
6229     \if_meaning:w \q_recursion_tail #2
6230     \exp_after:wN \prop_map_break:
6231     \fi:
6232     #1 {#2} {#3}
6233     \prop_map_function_aux:Nwn #1
6234   }
6235   \cs_generate_variant:Nn \prop_map_function:NN { Nc }
6236   \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 ??.)

\g_prop_map_inline_int A nesting counter for mapping.

```

6237   \int_new:N \g_prop_map_inline_int

```

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

\prop_map_inline:Nn Mapping in line requires a nesting level counter.

\prop_map_inline:cn

```

6238   \cs_new_protected:Npn \prop_map_inline:Nn #1#2
6239   {
6240     \int_gincr:N \g_prop_map_inline_int
6241     \cs_gset:cpn { prop_map_inline_ \int_use:N \g_prop_map_inline_int :nn }
6242     ##1##2 {#2}
6243     \prop_map_function:Nc #1
6244     { prop_map_inline_ \int_use:N \g_prop_map_inline_int :nn }

```

```

6245 \int_gdecr:N \g_prop_map_inline_int
6246 }
6247 \cs_generate_variant:Nn \prop_map_inline:Nn { c }
        (End definition for \prop_map_inline:Nn and \prop_map_inline:cn. These functions are docu-
        mented on page ??.)

```

`\prop_map_break:` Breaking the map function simply means removing everything up to the `\q_stop` marker.

```

6248 \cs_new_eq:NN \prop_map_break: \use_none_delimit_by_q_recursion_stop:w
        (End definition for \prop_map_break:. This function is documented on page ??.)

```

`\prop_map_break:n` The same idea for using one set of tokens.

```

6249 \cs_new_eq:NN \prop_map_break:n \use_i_delimit_by_q_recursion_stop:nw
        (End definition for \prop_map_break:n. This function is documented on page 119.)

```

192.6 Viewing property lists

`\l_prop_show_tl` Used to store the material for display.

```

6250 \tl_new:N \l_prop_show_tl
        (End definition for \l_prop_show_tl. This function is documented on page ??.)

```

`\prop_show:N` The aim of the mapping here is to create a token list containing the formatted property list. The very first item needs the new line and `>\` removing, which is achieved using `\prop_show:c` a `w`-type auxiliary. To avoid a low-level \TeX error if there is an empty property list, a simple test is used to keep the output “clean”.

```

\prop_show_aux:n a w-type auxiliary. To avoid a low-level  $\TeX$  error if there is an empty property list, a
\prop_show_aux:w simple test is used to keep the output “clean”.

6251 \cs_new_protected_nopar:Npn \prop_show:N #1
6252 {
6253   \prop_if_empty:NTF #1
6254   {
6255     \iow_term:x { Property-list~\token_to_str:N #1-is~empty }
6256     \tl_show:n { }
6257   }
6258   {
6259     \iow_term:x
6260     {
6261       Property-list~\token_to_str:N #1~
6262       contains~the~pairs~(without~outer~braces):
6263     }
6264     \tl_set:Nx \l_prop_show_tl
6265     { \prop_map_function:NN #1 \prop_show_aux:nn }
6266     \tl_show:n \exp_after:wN \exp_after:wN \exp_after:wN
6267     { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
6268   }
6269 }
6270 \cs_new:Npn \prop_show_aux:nn #1#2
6271 {
6272   \iow_newline: > \c_space_tl \c_space_tl
6273   \iow_char:N \{ #1 \iow_char:N \}
6274   \c_space_tl \c_space_tl => \c_space_tl \c_space_tl

```

```

6275     \iow_char:N \{ \exp_not:n {#2} \iow_char:N \}
6276   }
6277   \cs_new:Npn \prop_show_aux:w #1 > ~ { }
6278   \cs_generate_variant:Nn \prop_show:N { c }

```

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

192.7 Experimental functions

\prop_pop:NnN 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, \prg_return_true: is used after the assignments.

```

\prop_gpop:cnN
\prop_gpop:cnN
\prop_pop_aux_true:NNNnnn
6279 \prg_new_protected_conditional:Npnn \prop_pop:NnN #1#2#3 { T , F , TF }
6280 {
6281   \prop_split:NnTF #1 {#2}
6282   { \prop_pop_aux_true:NNNnnn \tl_set:Nn #1 #3 }
6283   { \prg_return_false: }
6284 }
6285 \prg_new_protected_conditional:Npnn \prop_gpop:NnN #1#2#3 { T , F , TF }
6286 {
6287   \prop_split:NnTF #1 {#2}
6288   { \prop_pop_aux_true:NNNnnn \tl_gset:Nn #1 #3 }
6289   { \prg_return_false: }
6290 }
6291 \cs_new_protected:Npn \prop_pop_aux_true:NNNnnn #1#2#3#4#5#6
6292 {
6293   \tl_set:Nn #3 {#5}
6294   #1 #2 { #4 #6 }
6295   \prg_return_true:
6296 }
6297 \cs_generate_variant:Nn \prop_pop:NnNT { c }
6298 \cs_generate_variant:Nn \prop_pop:NnNF { c }
6299 \cs_generate_variant:Nn \prop_pop:NnNTF { c }
6300 \cs_generate_variant:Nn \prop_gpop:NnNT { c }
6301 \cs_generate_variant:Nn \prop_gpop:NnNF { c }
6302 \cs_generate_variant:Nn \prop_gpop:NnNTF { c }

```

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

\prop_map_tokens:Nn The mapping grabs one key–value pair at a time, and stops when reaching the marker key \q_recursion_tail, which cannot appear in normal keys since those are strings.

\prop_map_tokens:cn The odd construction \use:n {#1} allows #1 to contain any token.

\prop_map_tokens_aux:nwn

```

6303 \cs_new:Npn \prop_map_tokens:Nn #1#2
6304 {
6305   \exp_last_unbraced:Nno \prop_map_tokens_aux:nwn {#2} #1
6306   \q_recursion_tail \q_prop { } \q_recursion_stop
6307 }
6308 \cs_new:Npn \prop_map_tokens_aux:nwn #1 \q_prop #2 \q_prop #3
6309 {
6310   \if_meaning:w \q_recursion_tail #2

```

```
6311 \exp_after:wN \prop_map_break:
```

```
6312 \fi:
```

```
6313 \use:n {#1} {#2} {#3}
```

```
6314 \prop_map_tokens_aux:nwn {#1}
```

```
6315 }
```

```
6316 \cs_generate_variant:Nn \prop_map_tokens:Nn { c }
```

(End definition for `\prop_map_tokens:Nn` and `\prop_map_tokens:cn`. These functions are documented on page ??.)

`\prop_get:Nn`

`\prop_get:cn`

`\prop_get_Nn_aux:nwn`

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.

```
6317 \cs_new:Npn \prop_get:Nn #1#2
```

```
6318 {
```

```
6319 \exp_last_unbraced:Noo \prop_get_Nn_aux:nwn
```

```
6320 { \tl_to_str:n {#2} } #1
```

```
6321 \tl_to_str:n {#2} \q_prop { }
```

```
6322 \q_recursion_stop
```

```
6323 }
```

```
6324 \cs_new:Npn \prop_get_Nn_aux:nwn #1 \q_prop #2 \q_prop #3
```

```
6325 {
```

```
6326 \str_if_eq:xxTF {#1} {#2}
```

```
6327 { \use_i_delimit_by_q_recursion_stop:nw {#3} }
```

```
6328 { \prop_get_Nn_aux:nwn {#1} }
```

```
6329 }
```

```
6330 \cs_generate_variant:Nn \prop_get:Nn { c }
```

(End definition for `\prop_get:Nn` and `\prop_get:cn`. These functions are documented on page ??.)

192.8 Deprecated interfaces

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\prop_display:N`

`\prop_display:c`

An older name for `\prop_show:N`.

```
6331 \*deprecated
```

```
6332 \cs_new_eq:NN \prop_display:N \prop_show:N
```

```
6333 \cs_new_eq:NN \prop_display:c \prop_show:c
```

```
6334 \*deprecated
```

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

`\prop_gget:NnN`

`\prop_gget:NVN`

`\prop_gget:cnN`

`\prop_gget:cVN`

`\prop_gget_aux:Nnnn`

Getting globally is no longer supported: this is a conceptual change, so the necessary code for the transition is provided directly.

```
6335 \*deprecated
```

```
6336 \cs_new_protected:Npn \prop_gget:NnN #1#2#3
```

```
6337 { \prop_split:Nnn #1 {#2} { \prop_gget_aux:Nnnn #3 } }
```

```
6338 \cs_new_protected:Npn \prop_gget_aux:Nnnn #1#2#3#4
```

```

6339 { \tl_gset:Nn #1 {#3} }
6340 \cs_generate_variant:Nn \prop_gget:NnN { NV }
6341 \cs_generate_variant:Nn \prop_gget:NnN { c , cV }
6342 \deprecated
(End definition for \prop_gget:NnN and others. These functions are documented on page ??.)

```

`\prop_get_gdel:NnN` This name seems very odd.

```

6343 \*deprecated
6344 \cs_new_eq:NN \prop_get_gdel:NnN \prop_gpop:NnN
6345 \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

```

6346 \*deprecated
6347 \cs_generate_variant:Nn \prop_if_in:NnT { cc }
6348 \cs_generate_variant:Nn \prop_if_in:NnF { cc }
6349 \cs_generate_variant:Nn \prop_if_in:NnTF { cc }
6350 \deprecated
(End definition for \prop_if_in:cc. This function is documented on page ??.)

```

`\prop_gput:ccx` Another one.

```

6351 \*deprecated
6352 \cs_generate_variant:Nn \prop_gput:Nnn { ccx }
6353 \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 6354 \*deprecated
\prop_if_eq:cN 6355 \prg_new_eq_conditional:NNn \prop_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\prop_if_eq:cc 6356 \prg_new_eq_conditional:NNn \prop_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
6357 \prg_new_eq_conditional:NNn \prop_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
6358 \prg_new_eq_conditional:NNn \prop_if_eq:cc \tl_if_eq:cc { p , T , F , TF }
6359 \deprecated
(End definition for \prop_if_eq:NN and others. These functions are documented on page ??.)
6360 \initex | package)

```

193 l3box implementation

```

6361 \*initex | package)
6362 \*package)
6363 \ProvidesExplPackage
6364 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
6365 \package_check_loaded_expl:
6366 \package)

```

The code in this module is very straight forward so I'm not going to comment it very extensively.

193.1 Creating and initialising boxes

The following test files are used for this code: m3box001.lvt.

<code>\box_new:N</code> <code>\box_new:c</code>	Defining a new $\langle box \rangle$ register: remember that box 255 is not generally available. <pre> 6367 <*package> 6368 \cs_new_protected:Npn \box_new:N #1 6369 { 6370 \chk_if_free_cs:N #1 6371 \newbox #1 6372 } 6373 </package> 6374 \cs_generate_variant:Nn \box_new:N { c } </pre>
<code>\box_clear:N</code> <code>\box_clear:c</code> <code>\box_gclear:N</code> <code>\box_gclear:c</code>	Clear a $\langle box \rangle$ register. <pre> 6375 \cs_new_protected:Npn \box_clear:N #1 6376 { \box_set_eq:NN #1 \c_empty_box } 6377 \cs_new_protected:Npn \box_gclear:N #1 6378 { \box_gset_eq:NN #1 \c_empty_box } 6379 \cs_generate_variant:Nn \box_clear:N { c } 6380 \cs_generate_variant:Nn \box_gclear:N { c } </pre>
<code>\box_clear_new:N</code> <code>\box_clear_new:c</code> <code>\box_gclear_new:N</code> <code>\box_gclear_new:c</code>	Clear or new. <pre> 6381 \cs_new_protected:Npn \box_clear_new:N #1 6382 { 6383 \cs_if_exist:NTF #1 6384 { \box_set_eq:NN #1 \c_empty_box } 6385 { \box_new:N #1 } 6386 } 6387 \cs_new_protected:Npn \box_gclear_new:N #1 6388 { 6389 \cs_if_exist:NTF #1 6390 { \box_gset_eq:NN #1 \c_empty_box } 6391 { \box_new:N #1 } 6392 } 6393 \cs_generate_variant:Nn \box_clear_new:N { c } 6394 \cs_generate_variant:Nn \box_gclear_new:N { c } </pre>
<code>\box_set_eq:NN</code> <code>\box_set_eq:cN</code> <code>\box_set_eq:Nc</code> <code>\box_set_eq:cc</code> <code>\box_gset_eq:NN</code> <code>\box_gset_eq:cN</code> <code>\box_gset_eq:Nc</code> <code>\box_gset_eq:cc</code>	Assigning the contents of a box to be another box. <pre> 6395 \cs_new_protected:Npn \box_set_eq:NN #1#2 6396 { \tex_setbox:D #1 \tex_copy:D #2 } 6397 \cs_new_protected:Npn \box_gset_eq:NN 6398 { \tex_global:D \box_set_eq:NN } 6399 \cs_generate_variant:Nn \box_set_eq:NN { cN , Nc , cc } 6400 \cs_generate_variant:Nn \box_gset_eq:NN { cN , Nc , cc } </pre>

<code>\box_set_eq_clear:NN</code>	Assigning the contents of a box to be another box. This clears the second box globally
<code>\box_set_eq_clear:cN</code>	(that's how TeX does it).
<code>\box_set_eq_clear:Nc</code>	6401 <code>\cs_new_protected:Npn \box_set_eq_clear:NN #1#2</code>
<code>\box_set_eq_clear:cc</code>	6402 <code>{ \tex_setbox:D #1 \tex_box:D #2 }</code>
<code>\box_gset_eq_clear:NN</code>	6403 <code>\cs_new_protected:Npn \box_gset_eq_clear:NN</code>
<code>\box_gset_eq_clear:cN</code>	6404 <code>{ \tex_global:D \box_set_eq_clear:NN }</code>
<code>\box_gset_eq_clear:Nc</code>	6405 <code>\cs_generate_variant:Nn \box_set_eq_clear:NN { cN , Nc , cc }</code>
<code>\box_gset_eq_clear:cc</code>	6406 <code>\cs_generate_variant:Nn \box_gset_eq_clear:NN { cN , Nc , cc }</code>

193.2 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>	6407 <code>\cs_new_eq:NN \box_ht:N \tex_ht:D</code>
<code>\box_dp:N</code>	6408 <code>\cs_new_eq:NN \box_dp:N \tex_dp:D</code>
<code>\box_dp:c</code>	6409 <code>\cs_new_eq:NN \box_wd:N \tex_wd:D</code>
<code>\box_wd:N</code>	6410 <code>\cs_generate_variant:Nn \box_ht:N { c }</code>
<code>\box_wd:c</code>	6411 <code>\cs_generate_variant:Nn \box_dp:N { c }</code>
	6412 <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>	6413 <code>\cs_new_protected:Npn \box_set_dp:Nn #1#2</code>
<code>\box_set_dp:cn</code>	6414 <code>{ \box_dp:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
<code>\box_set_wd:Nn</code>	6415 <code>\cs_new_protected:Npn \box_set_ht:Nn #1#2</code>
<code>\box_set_wd:cn</code>	6416 <code>{ \box_ht:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6417 <code>\cs_new_protected:Npn \box_set_wd:Nn #1#2</code>
	6418 <code>{ \box_wd:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6419 <code>\cs_generate_variant:Nn \box_set_ht:Nn { c }</code>
	6420 <code>\cs_generate_variant:Nn \box_set_dp:Nn { c }</code>
	6421 <code>\cs_generate_variant:Nn \box_set_wd:Nn { c }</code>

193.3 Using boxes

<code>\box_use_clear:N</code>	Using a $\langle box \rangle$. These are just TeX primitives with meaningful names.
<code>\box_use_clear:c</code>	6422 <code>\cs_new_eq:NN \box_use_clear:N \tex_box:D</code>
<code>\box_use:N</code>	6423 <code>\cs_new_eq:NN \box_use:N \tex_copy:D</code>
<code>\box_use:c</code>	6424 <code>\cs_generate_variant:Nn \box_use_clear:N { c }</code>
	6425 <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>	6426 <code>\cs_new_protected:Npn \box_move_left:nn #1#2</code>
<code>\box_move_up:nn</code>	6427 <code>{ \tex_moveleft:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
<code>\box_move_down:nn</code>	6428 <code>\cs_new_protected:Npn \box_move_right:nn #1#2</code>
	6429 <code>{ \tex_moveright:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6430 <code>\cs_new_protected:Npn \box_move_up:nn #1#2</code>
	6431 <code>{ \tex_raise:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6432 <code>\cs_new_protected:Npn \box_move_down:nn #1#2</code>
	6433 <code>{ \tex_lower:D \dim_eval:w #1 \dim_eval_end: #2 }</code>

193.4 Box conditionals

`\if_hbox:N` The primitives for testing if a $\langle box \rangle$ is empty/void or which type of box it is.

`\if_vbox:N` 6434 `\cs_new_eq:NN \if_hbox:N \tex_ifhbox:D`

`\if_box_empty:N` 6435 `\cs_new_eq:NN \if_vbox:N \tex_ifvbox:D`

6436 `\cs_new_eq:NN \if_box_empty:N \tex_ifvoid:D`

`\box_if_horizontal:N`

`\box_if_horizontal:c` 6437 `\prg_new_conditional:Npnn \box_if_horizontal:N #1 { p , T , F , TF }`

`\box_if_vertical:N` 6438 `{ \if_hbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

`\box_if_vertical:c` 6439 `\prg_new_conditional:Npnn \box_if_vertical:N #1 { p , T , F , TF }`

6440 `{ \if_vbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

6441 `\cs_generate_variant:Nn \box_if_horizontal_p:N { c }`

6442 `\cs_generate_variant:Nn \box_if_horizontal:NT { c }`

6443 `\cs_generate_variant:Nn \box_if_horizontal:NF { c }`

6444 `\cs_generate_variant:Nn \box_if_horizontal:NTF { c }`

6445 `\cs_generate_variant:Nn \box_if_vertical_p:N { c }`

6446 `\cs_generate_variant:Nn \box_if_vertical:NT { c }`

6447 `\cs_generate_variant:Nn \box_if_vertical:NF { c }`

6448 `\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` 6449 `\prg_new_conditional:Npnn \box_if_empty:N #1 { p , T , F , TF }`

6450 `{ \if_box_empty:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

6451 `\cs_generate_variant:Nn \box_if_empty_p:N { c }`

6452 `\cs_generate_variant:Nn \box_if_empty:NT { c }`

6453 `\cs_generate_variant:Nn \box_if_empty:NF { c }`

6454 `\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 ??.)

193.5 The last box inserted

`\box_set_to_last:N` Set a box to the previous box.

`\box_set_to_last:c` 6455 `\cs_new_protected:Npn \box_set_to_last:N #1`

`\box_gset_to_last:N` 6456 `{ \tex_setbox:D #1 \tex_lastbox:D }`

`\box_gset_to_last:c` 6457 `\cs_new_protected:Npn \box_gset_to_last:N`

6458 `{ \tex_global:D \box_set_to_last:N }`

6459 `\cs_generate_variant:Nn \box_set_to_last:N { c }`

6460 `\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 ??.)

193.6 Constant boxes

`\c_empty_box`

6461 $\langle *package \rangle$

6462 `\cs_new_eq:NN \c_empty_box \voidb@x`

6463 $\langle /package \rangle$


```

6464 <*initex>
6465 \box_new:N \c_empty_box
6466 </initex>

```

(End definition for `\c_empty_box`. This function is documented on page 127.)

193.7 Scratch boxes

```

\l_tmpa_box
\l_tmpb_box
6467 <*package>
6468 \cs_new_eq:NN \l_tmpa_box \@tempboxa
6469 </package>
6470 <*initex>
6471 \box_new:N \l_tmpa_box
6472 </initex>
6473 \box_new:N \l_tmpb_box

```

(End definition for `\l_tmpa_box` and `\l_tmpb_box`. These functions are documented on page 127.)

193.8 Viewing box contents

```

\box_show:N Show the contents of a box and write it into the log file.
\box_show:c
6474 \cs_new_eq:NN \box_show:N \tex_showbox:D
6475 \cs_generate_variant:Nn \box_show:N { c }

```

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

193.9 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.

```

6476 \cs_new_protected:Npn \hbox:n { \tex_hbox:D \scan_stop: }

```

(End definition for `\hbox:n`. This function is documented on page 127.)

```

\hbox_set:Nn
\hbox_set:cn
6477 \cs_new_protected:Npn \hbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_hbox:D {#2} }
\hbox_gset:Nn
6478 \cs_new_protected:Npn \hbox_gset:Nn { \tex_global:D \hbox_set:Nn }
\hbox_gset:cn
6479 \cs_generate_variant:Nn \hbox_set:Nn { c }
6480 \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
6481 \cs_new_protected:Npn \hbox_set_to_wd:Nnn #1#2#3
\hbox_gset_to_wd:Nnn
6482 { \tex_setbox:D #1 \tex_hbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }
\hbox_gset_to_wd:cnn
6483 \cs_new_protected:Npn \hbox_gset_to_wd:Nnn
6484 { \tex_global:D \hbox_set_to_wd:Nnn }
6485 \cs_generate_variant:Nn \hbox_set_to_wd:Nnn { c }
6486 \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` 6487 `\cs_new_protected:Npn \hbox_set:Nw #1`

`\hbox_gset:Nw` 6488 `{ \tex_setbox:D #1 \tex_hbox:D \c_group_begin_token }`

`\hbox_gset:cw` 6489 `\cs_new_protected:Npn \hbox_gset:Nw`

`\hbox_set_end:` 6490 `{ \tex_global:D \hbox_set:Nw }`

`\hbox_gset_end:` 6491 `\cs_generate_variant:Nn \hbox_set:Nw { c }`

6492 `\cs_generate_variant:Nn \hbox_gset:Nw { c }`

6493 `\cs_new_eq:NN \hbox_set_end: \c_group_end_token`

6494 `\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` 6495 `\cs_new_eq:NN \hbox_set_inline_begin:N \hbox_set:Nw`

`\hbox_gset_inline_begin:N` 6496 `\cs_new_eq:NN \hbox_set_inline_begin:c \hbox_set:cw`

`\hbox_gset_inline_begin:c` 6497 `\cs_new_eq:NN \hbox_set_inline_end: \hbox_set_end:`

`\hbox_set_inline_end:` 6498 `\cs_new_eq:NN \hbox_gset_inline_begin:N \hbox_gset:Nw`

`\hbox_gset_inline_end:` 6499 `\cs_new_eq:NN \hbox_gset_inline_begin:c \hbox_gset:cw`

6500 `\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` 6501 `\cs_new_protected:Npn \hbox_to_wd:nn #1#2`

6502 `{ \tex_hbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }`

6503 `\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 128.)

`\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.

6504 `\cs_new_protected:Npn \hbox_overlap_left:n #1`

6505 `{ \hbox_to_zero:n { \tex_hss:D #1 } }`

6506 `\cs_new_protected:Npn \hbox_overlap_right:n #1`

6507 `{ \hbox_to_zero:n { #1 \tex_hss:D } }`

(End definition for \hbox_overlap_left:n. This function is documented on page 128.)

`\hbox_unpack:N` Unpacking a box and if requested also clear it.

`\hbox_unpack:c` 6508 `\cs_new_eq:NN \hbox_unpack:N \tex_unhcopy:D`

`\hbox_unpack_clear:N` 6509 `\cs_new_eq:NN \hbox_unpack_clear:N \tex_unhbox:D`

`\hbox_unpack_clear:c` 6510 `\cs_generate_variant:Nn \hbox_unpack:N { c }`

6511 `\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 ??.)

193.10 Vertical mode boxes

`\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.

```
6512 \cs_new_protected:Npn \vbox:n { \tex_vbox:D \scan_stop: }
6513 \cs_new_protected:Npn \vbox_top:n { \tex_vtop:D \scan_stop: }
(End definition for \vbox:n. This function is documented on page 129.)
```

`\vbox_to_ht:nn` Put a vertical box directly into the input stream.

```
\vbox_to_zero:n 6514 \cs_new_protected:Npn \vbox_to_ht:nn #1#2
\vbox_to_ht:nn 6515 { \tex_vbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }
\vbox_to_zero:n 6516 \cs_new_protected:Npn \vbox_to_zero:n #1 { \tex_vbox:D to \c_zero_dim {#1} }
(End definition for \vbox_to_ht:nn and \vbox_to_zero:n. These functions are documented on
page 130.)
```

`\vbox_set:Nn` Storing material in a vertical box with a natural height.

```
\vbox_set:cn 6517 \cs_new_protected:Npn \vbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_vbox:D {#2} }
\vbox_gset:Nn 6518 \cs_new_protected:Npn \vbox_gset:Nn { \tex_global:D \vbox_set:Nn }
\vbox_gset:cn 6519 \cs_generate_variant:Nn \vbox_set:Nn { c }
6520 \cs_generate_variant:Nn \vbox_gset:Nn { c }
(End definition for \vbox_set:Nn and \vbox_set:cn. These functions are documented on page
??.)
```

`\vbox_set_top:Nn` Storing material in a vertical box with a natural height and reference point at the baseline
`\vbox_set_top:cn` of the first object in the box.

```
\vbox_gset_top:Nn 6521 \cs_new_protected:Npn \vbox_set_top:Nn #1#2
\vbox_gset_top:cn 6522 { \tex_setbox:D #1 \tex_vtop:D {#2} }
6523 \cs_new_protected:Npn \vbox_gset_top:Nn
6524 { \tex_global:D \vbox_set_top:Nn }
6525 \cs_generate_variant:Nn \vbox_set_top:Nn { c }
6526 \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 6527 \cs_new_protected:Npn \vbox_set_to_ht:Nnn #1#2#3
\vbox_gset_to_ht:Nnn 6528 { \tex_setbox:D #1 \tex_vbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }
\vbox_gset_to_ht:cnn 6529 \cs_new_protected:Npn \vbox_gset_to_ht:Nnn
6530 { \tex_global:D \vbox_set_to_ht:Nnn }
6531 \cs_generate_variant:Nn \vbox_set_to_ht:Nnn { c }
6532 \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 docu-
mented on page ??.)
```

`\vbox_set:Nw` Storing material in a vertical box. This type is useful in environment definitions.

`\vbox_set:cw` 6533 `\cs_new:Npn \vbox_set:Nw #1`

`\vbox_gset:Nw` 6534 `{ \tex_setbox:D #1 \tex_vbox:D \c_group_begin_token }`

`\vbox_gset:cw` 6535 `\cs_new_protected:Npn \vbox_gset:Nw`

`\vbox_set_end:` 6536 `{ \tex_global:D \vbox_set:Nw }`

`\vbox_gset_end:` 6537 `\cs_generate_variant:Nn \vbox_set:Nw { c }`

6538 `\cs_generate_variant:Nn \vbox_gset:Nw { c }`

6539 `\cs_new_eq:NN \vbox_set_end: \c_group_end_token`

6540 `\cs_new_eq:NN \vbox_gset_end: \c_group_end_token`

(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` 6541 `\cs_new_eq:NN \vbox_set_inline_begin:N \vbox_set:Nw`

`\vbox_gset_inline_begin:N` 6542 `\cs_new_eq:NN \vbox_set_inline_begin:c \vbox_set:cw`

`\vbox_gset_inline_begin:c` 6543 `\cs_new_eq:NN \vbox_set_inline_end: \vbox_set_end:`

`\vbox_set_inline_end:` 6544 `\cs_new_eq:NN \vbox_gset_inline_begin:N \vbox_gset:Nw`

`\vbox_gset_inline_end:` 6545 `\cs_new_eq:NN \vbox_gset_inline_begin:c \vbox_gset:cw`

6546 `\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` 6547 `\cs_new_eq:NN \vbox_unpack:N \tex_unvcopy:D`

`\vbox_unpack_clear:N` 6548 `\cs_new_eq:NN \vbox_unpack_clear:N \tex_unvbox:D`

`\vbox_unpack_clear:c` 6549 `\cs_generate_variant:Nn \vbox_unpack:N { c }`

6550 `\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.

6551 `\cs_new_protected:Npn \vbox_set_split_to_ht:NNn #1#2#3`

6552 `{ \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 131.)

193.11 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.

6553 `\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` 6554 `\fp_new:N \l_box_cos_fp`

6555 `\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 ??.)

<pre> \l_box_top_dim \l_box_bottom_dim \l_box_left_dim \l_box_right_dim </pre>	<p>These are the positions of the four edges of a box before manipulation.</p> <pre> 6556 \dim_new:N \l_box_top_dim 6557 \dim_new:N \l_box_bottom_dim 6558 \dim_new:N \l_box_left_dim 6559 \dim_new:N \l_box_right_dim </pre> <p>(End definition for \l_box_top_dim and others. These functions are documented on page ??.)</p>
<pre> \l_box_top_new_dim \l_box_bottom_new_dim \l_box_left_new_dim \l_box_right_new_dim </pre>	<p>These are the positions of the four edges of a box after manipulation.</p> <pre> 6560 \dim_new:N \l_box_top_new_dim 6561 \dim_new:N \l_box_bottom_new_dim 6562 \dim_new:N \l_box_left_new_dim 6563 \dim_new:N \l_box_right_new_dim </pre> <p>(End definition for \l_box_top_new_dim and others. These functions are documented on page ??.)</p>
<pre> \l_box_tmp_box \l_box_tmp_fp </pre>	<p>Scratch space.</p> <pre> 6564 \box_new:N \l_box_tmp_box 6565 \fp_new:N \l_box_tmp_fp </pre> <p>(End definition for \l_box_tmp_box and \l_box_tmp_fp. These functions are documented on page ??.)</p>
<pre> \l_box_x_fp \l_box_y_fp \l_box_x_new_fp \l_box_y_new_fp </pre>	<p>Used as the input and output values for a point when manipulation the location.</p> <pre> 6566 \fp_new:N \l_box_x_fp 6567 \fp_new:N \l_box_y_fp 6568 \fp_new:N \l_box_x_new_fp 6569 \fp_new:N \l_box_y_new_fp </pre> <p>(End definition for \l_box_x_fp and others. These functions are documented on page ??.)</p>
<pre> \box_rotate:Nn \box_rotate_aux:N \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: </pre>	<p>Rotation of a box starts with working out the relevant sine and cosine. There is then a check to avoid doing any real work for the trivial rotation.</p> <pre> 6570 \cs_new_protected:Npn \box_rotate:Nn #1#2 6571 { 6572 \hbox_set:Nn #1 6573 { 6574 \group_begin: 6575 \fp_set:Nn \l_box_angle_fp {#2} 6576 \box_rotate_set_sin_cos: 6577 \fp_compare:NNNTF \l_box_sin_fp = \c_zero_fp 6578 { 6579 \fp_compare:NNNTF \l_box_cos_fp = \c_one_fp 6580 { \box_use:N #1 } 6581 { \box_rotate_aux:N #1 } 6582 } 6583 { \box_rotate_aux:N #1 } 6584 \group_end: 6585 } 6586 } </pre>

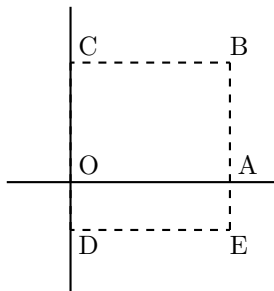


Figure 1: Co-ordinates of a box prior to rotation.

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.

```

6587 \cs_new_protected:Npn \box_rotate_aux:N #1
6588 {
6589   \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6590   \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6591   \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6592   \dim_zero:N \l_box_left_dim

```

The next step is to work out the x and y coordinates of vertices of the rotated box in relation to its original coordinates. The box can be visualized with vertices B , C , D and E is illustrated (Figure 1). The vertex O is the reference point on the baseline, and in this implementation is also the centre of rotation. The formulae are, for a point P and angle α :

$$\begin{aligned}
 P'_x &= P_x - O_x \\
 P'_y &= P_y - O_y \\
 P''_x &= (P'_x \cos(\alpha)) - (P'_y \sin(\alpha)) \\
 P''_y &= (P'_x \sin(\alpha)) + (P'_y \cos(\alpha)) \\
 P'''_x &= P''_x + O_x + L_x \\
 P'''_y &= P''_y + O_y
 \end{aligned}$$

The “extra” horizontal translation L_x at the end is calculated so that the leftmost point of the resulting box has x -coordinate 0. This is desirable as \TeX boxes must have the reference point at the left edge of the box. (As O is always $(0,0)$, this part of the calculation is omitted here.)

```

6593   \fp_compare:NNTF \l_box_sin_fp > \c_zero_fp
6594   {
6595     \fp_compare:NNTF \l_box_cos_fp > \c_zero_fp
6596     { \box_rotate_quadrant_one: }
6597     { \box_rotate_quadrant_two: }
6598   }
6599   {
6600     \fp_compare:NNTF \l_box_cos_fp < \c_zero_fp
6601     { \box_rotate_quadrant_three: }

```

```

6602         { \box_rotate_quadrant_four: }
6603     }

```

The position of the box edges are now known, but the box at this stage be misplaced relative to the current T_EX 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.

```

6604     \hbox_set:Nn \l_box_tmp_box { \box_use:N #1 }
6605     \hbox_set:Nn \l_box_tmp_box
6606     {
6607         \tex_kern:D -\l_box_left_new_dim
6608         \hbox:n
6609         {
6610             \driver_box_rotate_begin:
6611             \box_use:N \l_box_tmp_box
6612             \driver_box_rotate_end:
6613         }
6614     }

```

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.

```

6615     \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6616     \box_set_dp:Nn \l_box_tmp_box { -\l_box_bottom_new_dim }
6617     \box_set_wd:Nn \l_box_tmp_box
6618     { \l_box_right_new_dim - \l_box_left_new_dim }
6619     \box_use:N \l_box_tmp_box
6620 }

```

A simple conversion from degrees to radians followed by calculation of the sine and cosine.

```

6621 \cs_new_protected:Npn \box_rotate_set_sin_cos:
6622 {
6623     \fp_set_eq:NN \l_box_tmp_fp \l_box_angle_fp
6624     \fp_div:Nn \l_box_tmp_fp { 180 }
6625     \fp_mul:Nn \l_box_tmp_fp { \c_pi_fp }
6626     \fp_sin:Nn \l_box_sin_fp { \l_box_tmp_fp }
6627     \fp_cos:Nn \l_box_cos_fp { \l_box_tmp_fp }
6628 }

```

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.

```

6629 \cs_new_protected:Npn \box_rotate_x:nnN #1#2#3
6630 {
6631     \fp_set_from_dim:Nn \l_box_x_fp {#1}
6632     \fp_set_from_dim:Nn \l_box_y_fp {#2}
6633     \fp_set_eq:NN \l_box_x_new_fp \l_box_x_fp
6634     \fp_set_eq:NN \l_box_tmp_fp \l_box_y_fp
6635     \fp_mul:Nn \l_box_x_new_fp { \l_box_cos_fp }
6636     \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }

```

```

6637 \fp_sub:Nn \l_box_x_new_fp { \l_box_tmp_fp }
6638 \dim_set:Nn #3 { \fp_to_dim:N \l_box_x_new_fp }
6639 }
6640 \cs_new_protected:Npn \box_rotate_y:nnN #1#2#3
6641 {
6642 \fp_set_from_dim:Nn \l_box_x_fp {#1}
6643 \fp_set_from_dim:Nn \l_box_y_fp {#2}
6644 \fp_set_eq:NN \l_box_y_new_fp \l_box_y_fp
6645 \fp_set_eq:NN \l_box_tmp_fp \l_box_x_fp
6646 \fp_mul:Nn \l_box_y_new_fp { \l_box_cos_fp }
6647 \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }
6648 \fp_add:Nn \l_box_y_new_fp { \l_box_tmp_fp }
6649 \dim_set:Nn #3 { \fp_to_dim:N \l_box_y_new_fp }
6650 }

```

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.

```

6651 \cs_new_protected:Npn \box_rotate_quadrant_one:
6652 {
6653 \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6654 \l_box_top_new_dim
6655 \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6656 \l_box_bottom_new_dim
6657 \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6658 \l_box_left_new_dim
6659 \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6660 \l_box_right_new_dim
6661 }
6662 \cs_new_protected:Npn \box_rotate_quadrant_two:
6663 {
6664 \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6665 \l_box_top_new_dim
6666 \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6667 \l_box_bottom_new_dim
6668 \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6669 \l_box_left_new_dim
6670 \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6671 \l_box_right_new_dim
6672 }
6673 \cs_new_protected:Npn \box_rotate_quadrant_three:
6674 {
6675 \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6676 \l_box_top_new_dim
6677 \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6678 \l_box_bottom_new_dim
6679 \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6680 \l_box_left_new_dim

```



```

6681 \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6682 \l_box_right_new_dim
6683 }
6684 \cs_new_protected:Npn \box_rotate_quadrant_four:
6685 {
6686 \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6687 \l_box_top_new_dim
6688 \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6689 \l_box_bottom_new_dim
6690 \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6691 \l_box_left_new_dim
6692 \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6693 \l_box_right_new_dim
6694 }

```

(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 6695 \fp_new:N \l_box_scale_x_fp
6696 \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 6697 \cs_new_protected:Npn \box_resize:Nnn #1#2#3
\box_resize_aux:Nnn 6698 {
6699 \hbox_set:Nn #1
6700 {
6701 \group_begin:
6702 \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6703 \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6704 \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6705 \dim_zero:N \l_box_left_dim

```

The 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.

```

6706 \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6707 \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6708 \fp_div:Nn \l_box_scale_x_fp { \l_box_tmp_fp }

```

The y -scaling needs both the height and the depth of the current box.

```

6709 \fp_set_from_dim:Nn \l_box_scale_y_fp {#3}
6710 \fp_set_from_dim:Nn \l_box_tmp_fp
6711 { \l_box_top_dim - \l_box_bottom_dim }
6712 \fp_div:Nn \l_box_scale_y_fp { \l_box_tmp_fp }

```

At this stage, check for trivial scaling. If both scalings are unity, then the code does nothing. Otherwise, pass on to the auxiliary function to find the new dimensions.

```

6713 \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6714 {
6715 \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp

```

```

6716         { \box_use:N #1 }
6717         { \box_resize_aux:Nnn #1 {#2} {#3} }
6718     }
6719     { \box_resize_aux:Nnn #1 {#2} {#3} }
6720 \group_end:
6721 }
6722 }
6723 \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.

```

6724 \cs_new_protected:Npn \box_resize_aux:Nnn #1#2#3
6725 {
6726   \dim_compare:nNnTF {#2} > \c_zero_dim
6727   { \dim_set:Nn \l_box_right_new_dim {#2} }
6728   { \dim_set:Nn \l_box_right_new_dim { \c_zero_dim - ( #2 ) } }
6729   \dim_compare:nNnTF {#3} > \c_zero_dim
6730   {
6731     \dim_set:Nn \l_box_top_new_dim
6732     { \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6733     \dim_set:Nn \l_box_bottom_new_dim
6734     { \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6735   }
6736   {
6737     \dim_set:Nn \l_box_top_new_dim
6738     { - \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6739     \dim_set:Nn \l_box_bottom_new_dim
6740     { - \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6741   }
6742   \box_resize_common:N #1
6743 }

```

(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,
\box_resize_to_ht_plus_dp:cn 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
\box_resize_to_wd:cn internal code to be used as for the general case).

```

6744 \cs_new_protected:Npn \box_resize_to_ht_plus_dp:Nn #1#2
6745 {
6746   \hbox_set:Nn #1
6747   {
6748     \group_begin:
6749     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6750     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6751     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }

```

```

6752         \dim_zero:N \l_box_left_dim
6753         \fp_set_from_dim:Nn \l_box_scale_y_fp {#2}
6754         \fp_set_from_dim:Nn \l_box_tmp_fp
6755             { \l_box_top_dim - \l_box_bottom_dim }
6756         \fp_div:Nn \l_box_scale_y_fp { \l_box_tmp_fp }
6757         \fp_set_eq:NN \l_box_scale_x_fp \l_box_scale_y_fp
6758         \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6759             { \box_use:N #1 }
6760             { \box_resize_aux:Nnn #1 {#2} {#2} }
6761     \group_end:
6762 }
6763 }
6764 \cs_generate_variant:Nn \box_resize_to_ht_plus_dp:Nn { c }
6765 \cs_new_protected:Npn \box_resize_to_wd:Nn #1#2
6766 {
6767     \hbox_set:Nn #1
6768     {
6769         \group_begin:
6770         \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6771         \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6772         \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6773         \dim_zero:N \l_box_left_dim
6774         \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6775         \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6776         \fp_div:Nn \l_box_scale_x_fp { \l_box_tmp_fp }
6777         \fp_set_eq:NN \l_box_scale_y_fp \l_box_scale_x_fp
6778         \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6779             { \box_use:N #1 }
6780             { \box_resize_aux:Nnn #1 {#2} {#2} }
6781         \group_end:
6782     }
6783 }
6784 \cs_generate_variant:Nn \box_resize_to_wd:Nn { c }

```

(End definition for `\box_resize_to_ht_plus_dp:Nn` and `\box_resize_to_ht_plus_dp:cn`. These functions are documented on page ??.)

`\box_scale:Nnn` When scaling a box, setting the scaling itself is easy enough. The new dimensions are
`\box_scale:cn` also relatively easy to find, allowing only for the need to keep them positive in all cases.
`\box_scale_aux:Nnn` Once that is done then after a check for the trivial scaling a hand-off can be made to the common code. The dimension scaling operations are carried out using the \TeX mechanism as it avoids needing to use fp operations.

```

6785 \cs_new_protected:Npn \box_scale:Nnn #1#2#3
6786 {
6787     \hbox_set:Nn #1
6788     {
6789         \group_begin:
6790         \fp_set:Nn \l_box_scale_x_fp {#2}
6791         \fp_set:Nn \l_box_scale_y_fp {#3}
6792         \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }

```

```

6793 \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6794 \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6795 \dim_zero:N \l_box_left_dim
6796 \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6797 {
6798 \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6799 { \box_use:N #1 }
6800 { \box_scale_aux:Nnn #1 {#2} {#3} }
6801 }
6802 { \box_scale_aux:Nnn #1 {#2} {#3} }
6803 \group_end:
6804 }
6805 }
6806 \cs_generate_variant:Nn \box_scale:Nnn { c }
6807 \cs_new_protected:Npn \box_scale_aux:Nnn #1#2#3
6808 {
6809 \fp_compare:NNTF \l_box_scale_y_fp > \c_zero_fp
6810 {
6811 \dim_set:Nn \l_box_top_new_dim { #3 \l_box_top_dim }
6812 \dim_set:Nn \l_box_bottom_new_dim { #3 \l_box_bottom_dim }
6813 }
6814 {
6815 \dim_set:Nn \l_box_top_new_dim { -#3 \l_box_bottom_dim }
6816 \dim_set:Nn \l_box_bottom_new_dim { -#3 \l_box_top_dim }
6817 }
6818 \fp_compare:NNTF \l_box_scale_x_fp > \c_zero_fp
6819 { \l_box_right_new_dim #2 \l_box_right_dim }
6820 { \l_box_right_new_dim -#2 \l_box_right_dim }
6821 \box_resize_common:N #1
6822 }

```

(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.

```

6823 \cs_new_protected:Npn \box_resize_common:N #1
6824 {
6825 \hbox_set:Nn \l_box_tmp_box
6826 {
6827 \driver_box_scale_begin:
6828 \hbox_overlap_right:n { \box_use:N #1 }
6829 \driver_box_scale_end:
6830 }

```

The new height and depth can be applied directly.

```

6831 \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6832 \box_set_dp:Nn \l_box_tmp_box { \l_box_bottom_new_dim }

```

Things are not quite as obvious for the width, as the reference point needs to remain unchanged. For positive scaling factors resizing the box is all that is needed. However,

for case of a negative scaling the material must be shifted such that the reference point ends up in the right place.

```

6833 \fp_compare:NNTF \l_box_scale_x_fp < \c_zero_fp
6834 {
6835   \hbox_to_wd:nn { \l_box_right_new_dim }
6836   {
6837     \tex_kern:D \l_box_right_new_dim
6838     \box_use:N \l_box_tmp_box
6839     \tex_hss:D
6840   }
6841 }
6842 {
6843   \box_set_wd:Nn \l_box_tmp_box { \l_box_right_new_dim }
6844   \box_use:N \l_box_tmp_box
6845 }
6846 }

```

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

193.12 Viewing part of a box

`\box_clip:N` A wrapper around the driver-dependent code.

```

\box_clip:c
6847 \cs_new_protected:Npn \box_clip:N #1
6848 { \hbox_set:Nn #1 { \driver_box_use_clip:N #1 } }
6849 \cs_generate_variant:Nn \box_clip:N { c }

```

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

`\box_trim:Nnnnn` Trimming from the left- and right-hand edges of the box is easy. The total width is set to remove from the right, and a skip will shift the material to remove from the left.

```

\box_trim:cnnnn
6850 \cs_new_protected:Npn \box_trim:Nnnnn #1#2#3#4#5
6851 {
6852   \box_set_wd:Nn #1 { \box_wd:N #1 - \dim_eval:n {#4} - \dim_eval:n {#2} }
6853   \hbox_set:Nn #1
6854   {
6855     \skip_horizontal:n { - \dim_eval:n {#2} }
6856     \box_use:N #1
6857   }

```

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.

```

6858 \dim_compare:nNnTF { \box_dp:N #1 } > {#3}
6859 { \box_set_dp:Nn #1 { \box_dp:N #1 - \dim_eval:n {#3} } }
6860 {
6861   \hbox_set:Nn #1
6862   {
6863     \box_move_down:nn { \dim_eval:n {#3} - \box_dp:N #1 }
6864     { \box_use:N #1 }
6865   }
6866   \box_set_dp:Nn #1 \c_zero_dim

```

```

6867     }
6868     \dim_compare:nNnTF { \box_ht:N #1 } > {#5}
6869     { \box_set_ht:Nn #1 { \box_ht:N #1 - \dim_eval:n {#5} } }
6870     {
6871       \hbox_set:Nn #1
6872       {
6873         \box_move_up:nn { \dim_eval:n {#5} - \box_ht:N #1 }
6874         { \box_use:N #1 }
6875       }
6876       \box_set_ht:Nn #1 \c_zero_dim
6877     }
6878   }
6879   \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.

`\box_viewport:cnnnn`

```

6880   \cs_new_protected:Npn \box_viewport:Nnnnn #1#2#3#4#5
6881   {
6882     \box_set_wd:Nn #1 { \dim_eval:n {#4} - \dim_eval:n {#2} }
6883     \hbox_set:Nn #1
6884     {
6885       \skip_horizontal:n { - \dim_eval:n {#2} }
6886       \box_use:N #1
6887     }
6888     \dim_compare:nNnTF {#3} > \c_zero_dim
6889     {
6890       \hbox_set:Nn #1 { \box_move_down:nn {#3} { \box_use:N #1 } }
6891       \box_set_dp:Nn #1 \c_zero_dim
6892     }
6893     { \box_set_dp:Nn #1 { - \dim_eval:n {#3} } }
6894     \dim_compare:nNnTF {#5} > \c_zero_dim
6895     { \box_set_ht:Nn #1 {#5} }
6896     {
6897       \hbox_set:Nn #1
6898       { \box_move_up:nn { -\dim_eval:n {#5} } { \box_use:N #1 } }
6899       \box_set_ht:Nn #1 \c_zero_dim
6900     }
6901   }
6902   \cs_generate_variant:Nn \box_viewport:Nnnnn { c }

```

(End definition for `\box_viewport:Nnnnn` and `\box_viewport:cnnnn`. These functions are documented on page ??.)

193.13 Deprecated functions

`\l_last_box` Deprecated 2011-11-13, for removal by 2012-02-28.

```

6903   \cs_new_eq:NN \l_last_box \tex_lastbox:D

```

(End definition for `\l_last_box`. This function is documented on page ??.)

6904 `\</initex | package>`

194 l3coffins Implementation

6905 `<*initex | package>`

6906 `<*package>`

6907 `\ProvidesExplPackage`

6908 `{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}`

6909 `\package_check_loaded_expl:`

6910 `\</package>`

194.1 Coffins: data structures and general variables

`\l_coffin_tmp_box` Scratch variables.

`\l_coffin_tmp_dim` 6911 `\box_new:N \l_coffin_tmp_box`

`\l_coffin_tmp_fp` 6912 `\dim_new:N \l_coffin_tmp_dim`

`\l_coffin_tmp_tl` 6913 `\fp_new:N \l_coffin_tmp_fp`

6914 `\tl_new:N \l_coffin_tmp_tl`

(End definition for `\l_coffin_tmp_box`. This function is documented on page ??.)

`\c_coffin_corners_prop` The “corners”; of a coffin define the real content, as opposed to the \TeX bounding box. They all start off in the same place, of course.

6915 `\prop_new:N \c_coffin_corners_prop`

6916 `\prop_put:Nnn \c_coffin_corners_prop { tl } { { 0 pt } { 0 pt } }`

6917 `\prop_put:Nnn \c_coffin_corners_prop { tr } { { 0 pt } { 0 pt } }`

6918 `\prop_put:Nnn \c_coffin_corners_prop { bl } { { 0 pt } { 0 pt } }`

6919 `\prop_put:Nnn \c_coffin_corners_prop { br } { { 0 pt } { 0 pt } }`

(End definition for `\c_coffin_corners_prop`. This function is documented on page ??.)

`\c_coffin_poles_prop` Pole positions are given for horizontal, vertical and reference-point based values.

6920 `\prop_new:N \c_coffin_poles_prop`

6921 `\tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } }`

6922 `\prop_put:Nno \c_coffin_poles_prop { l } { \l_coffin_tmp_tl }`

6923 `\prop_put:Nno \c_coffin_poles_prop { hc } { \l_coffin_tmp_tl }`

6924 `\prop_put:Nno \c_coffin_poles_prop { r } { \l_coffin_tmp_tl }`

6925 `\tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } { 0 pt } }`

6926 `\prop_put:Nno \c_coffin_poles_prop { b } { \l_coffin_tmp_tl }`

6927 `\prop_put:Nno \c_coffin_poles_prop { vc } { \l_coffin_tmp_tl }`

6928 `\prop_put:Nno \c_coffin_poles_prop { t } { \l_coffin_tmp_tl }`

6929 `\prop_put:Nno \c_coffin_poles_prop { B } { \l_coffin_tmp_tl }`

6930 `\prop_put:Nno \c_coffin_poles_prop { H } { \l_coffin_tmp_tl }`

6931 `\prop_put:Nno \c_coffin_poles_prop { T } { \l_coffin_tmp_tl }`

(End definition for `\c_coffin_poles_prop`. This function is documented on page ??.)

<p>\l_coffin_calc_a_fp</p> <p>\l_coffin_calc_b_fp</p> <p>\l_coffin_calc_c_fp</p> <p>\l_coffin_calc_d_fp</p> <p>\l_coffin_calc_result_fp</p>	<p>Used for calculations of intersections and in other internal places.</p> <p>6932 \fp_new:N \l_coffin_calc_a_fp</p> <p>6933 \fp_new:N \l_coffin_calc_b_fp</p> <p>6934 \fp_new:N \l_coffin_calc_c_fp</p> <p>6935 \fp_new:N \l_coffin_calc_d_fp</p> <p>6936 \fp_new:N \l_coffin_calc_result_fp</p> <p>(End definition for \l_coffin_calc_a_fp. This function is documented on page ??.)</p>
<p>\l_coffin_error_bool</p>	<p>For propagating errors so that parts of the code can work around them.</p> <p>6937 \bool_new:N \l_coffin_error_bool</p> <p>(End definition for \l_coffin_error_bool. This function is documented on page ??.)</p>
<p>\l_coffin_offset_x_dim</p> <p>\l_coffin_offset_y_dim</p>	<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> <p>6938 \dim_new:N \l_coffin_offset_x_dim</p> <p>6939 \dim_new:N \l_coffin_offset_y_dim</p> <p>(End definition for \l_coffin_offset_x_dim. This function is documented on page ??.)</p>
<p>\l_coffin_pole_a_tl</p> <p>\l_coffin_pole_b_tl</p>	<p>Needed for finding the intersection of two poles.</p> <p>6940 \tl_new:N \l_coffin_pole_a_tl</p> <p>6941 \tl_new:N \l_coffin_pole_b_tl</p> <p>(End definition for \l_coffin_pole_a_tl. This function is documented on page ??.)</p>
<p>\l_coffin_sin_fp</p> <p>\l_coffin_cos_fp</p>	<p>Used for rotations to get the sine and cosine values.</p> <p>6942 \fp_new:N \l_coffin_sin_fp</p> <p>6943 \fp_new:N \l_coffin_cos_fp</p> <p>(End definition for \l_coffin_sin_fp. This function is documented on page ??.)</p>
<p>\l_coffin_x_dim</p> <p>\l_coffin_y_dim</p> <p>\l_coffin_x_prime_dim</p> <p>\l_coffin_y_prime_dim</p>	<p>For calculating intersections and so forth.</p> <p>6944 \dim_new:N \l_coffin_x_dim</p> <p>6945 \dim_new:N \l_coffin_y_dim</p> <p>6946 \dim_new:N \l_coffin_x_prime_dim</p> <p>6947 \dim_new:N \l_coffin_y_prime_dim</p> <p>(End definition for \l_coffin_x_dim. This function is documented on page ??.)</p>
<p>\l_coffin_x_fp</p> <p>\l_coffin_y_fp</p> <p>\l_coffin_x_prime_fp</p> <p>\l_coffin_y_prime_fp</p>	<p>Used for calculations where there are clear x- and y-components, for example during vector rotation.</p> <p>6948 \fp_new:N \l_coffin_x_fp</p> <p>6949 \fp_new:N \l_coffin_y_fp</p> <p>6950 \fp_new:N \l_coffin_x_prime_fp</p> <p>6951 \fp_new:N \l_coffin_y_prime_fp</p> <p>(End definition for \l_coffin_x_fp. This function is documented on page ??.)</p>
<p>\l_coffin_Depth_dim</p> <p>\l_coffin_Height_dim</p> <p>\l_coffin_TotalHeight_dim</p> <p>\l_coffin_Width_dim</p>	<p>Dimensions for the various parts of a coffin.</p> <p>6952 \dim_new:N \l_coffin_Depth_dim</p> <p>6953 \dim_new:N \l_coffin_Height_dim</p> <p>6954 \dim_new:N \l_coffin_TotalHeight_dim</p> <p>6955 \dim_new:N \l_coffin_Width_dim</p>

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

```

\coffin_saved_Depth: Used to save the meaning of \Depth, \Height, \TotalHeight and \Width.
\coffin_saved_Height: 6956 \cs_new_nopar:Npn \coffin_saved_Depth: { }
\coffin_saved_TotalHeight: 6957 \cs_new_nopar:Npn \coffin_saved_Height: { }
\coffin_saved_Width: 6958 \cs_new_nopar:Npn \coffin_saved_TotalHeight: { }
6959 \cs_new_nopar:Npn \coffin_saved_Width: { }
(End definition for \coffin_saved_Depth:. This function is documented on page ??.)

```

194.2 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.

```

6960 \cs_new_protected:Npn \coffin_if_exist:NT #1#2
6961 {
6962   \cs_if_exist:NTF #1
6963   {
6964     \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
6965     { #2 }
6966     {
6967       \msg_kernel_error:nnx { coffins } { unknown-coffin }
6968       { \token_to_str:N #1 }
6969     }
6970   }
6971   {
6972     \msg_kernel_error:nnx { coffins } { unknown-coffin }
6973     { \token_to_str:N #1 }
6974   }
6975 }
(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 6976 \cs_new_protected_nopar:Npn \coffin_clear:N #1
6977 {
6978   \coffin_if_exist:NT #1
6979   {
6980     \box_clear:N #1
6981     \coffin_reset_structure:N #1
6982   }
6983 }
6984 \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.

```

6985 \cs_new_protected_nopar:Npn \coffin_new:N #1
6986 {
6987   \box_new:N #1
6988   \prop_clear_new:c { l_coffin_corners_ \int_value:w #1 _prop }
6989   \prop_clear_new:c { l_coffin_poles_ \int_value:w #1 _prop }
6990   \prop_gset_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
6991     \c_coffin_corners_prop
6992   \prop_gset_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
6993     \c_coffin_poles_prop
6994 }
6995 \cs_generate_variant:Nn \coffin_new:N { c }

```

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

`\hcoffin_set:Nn` Horizontal coffins are relatively easy: set the appropriate box, reset the structures then
`\hcoffin_set:cn` update the handle positions.

```

6996 \cs_new_protected:Npn \hcoffin_set:Nn #1#2
6997 {
6998   \coffin_if_exist:NT #1
6999   {
7000     \hbox_set:Nn #1
7001     {
7002       \color_group_begin:
7003       \color_ensure_current:
7004       #2
7005       \color_group_end:
7006     }
7007     \coffin_reset_structure:N #1
7008     \coffin_update_poles:N #1
7009     \coffin_update_corners:N #1
7010   }
7011 }
7012 \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.

```

7013 \cs_new_protected:Npn \vcoffin_set:Nnn #1#2#3
7014 {
7015   \coffin_if_exist:NT #1
7016   {
7017     \vbox_set:Nn #1
7018     {

```

```

7019         \dim_set:Nn \tex_hsize:D {#2}
7020         \color_group_begin:
7021         \color_ensure_current:
7022         #3
7023         \color_group_end:
7024     }
7025     \coffin_reset_structure:N #1
7026     \coffin_update_poles:N #1
7027     \coffin_update_corners:N #1
7028     \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
7029     \coffin_set_pole:Nnx #1 { T }
7030     {
7031         { 0 pt }
7032         { \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box } }
7033         { 1000 pt }
7034         { 0 pt }
7035     }
7036     \box_clear:N \l_coffin_tmp_box
7037 }
7038 }
7039 \cs_generate_variant:Nn \vcoffin_set:Nnn { c }

```

(End definition for \vcoffin_set:Nnn and \vcoffin_set:cnn. These functions are documented on page ??.)

\hcoffin_set:Nw These are the “begin”/“end” versions of the above: watch the grouping!

```

\hcoffin_set:cw 7040 \cs_new_protected_nopar:Npn \hcoffin_set:Nw #1
\hcoffin_set_end: 7041 {
7042     \coffin_if_exist:NT #1
7043     {
7044         \hbox_set:Nw #1 \color_group_begin: \color_ensure_current:
7045         \cs_set_protected_nopar:Npn \hcoffin_set_end:
7046         {
7047             \color_group_end:
7048             \hbox_set_end:
7049             \coffin_reset_structure:N #1
7050             \coffin_update_poles:N #1
7051             \coffin_update_corners:N #1
7052         }
7053     }
7054 }
7055 \cs_new_protected_nopar:Npn \hcoffin_set_end: { }
7056 \cs_generate_variant:Nn \hcoffin_set:Nw { c }

```

(End definition for \hcoffin_set:Nw and \hcoffin_set:cw. These functions are documented on page ??.)

\vcoffin_set:Nnw The same for vertical coffins.

```

\vcoffin_set:cnw 7057 \cs_new_protected_nopar:Npn \vcoffin_set:Nnw #1#2
\vcoffin_set_end: 7058 {
7059     \coffin_if_exist:NT #1

```

```

7060 {
7061   \vbox_set:Nw #1
7062   \dim_set:Nn \tex_hsize:D {#2}
7063   \color_group_begin: \color_ensure_current:
7064   \cs_set_protected:Npn \vcoffin_set_end:
7065   {
7066     \color_group_end:
7067     \vbox_set_end:
7068     \coffin_reset_structure:N #1
7069     \coffin_update_poles:N #1
7070     \coffin_update_corners:N #1
7071     \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
7072     \coffin_set_pole:Nnx #1 { T }
7073     {
7074       { 0 pt }
7075       {
7076         \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box }
7077       }
7078       { 1000 pt }
7079       { 0 pt }
7080     }
7081     \box_clear:N \l_coffin_tmp_box
7082   }
7083 }
7084 }
7085 \cs_new_protected_nopar:Npn \vcoffin_set_end: { }
7086 \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 7087 \cs_new_protected_nopar:Npn \coffin_set_eq:NN #1#2
\coffin_set_eq:cN 7088 {
\coffin_set_eq:cc 7089   \coffin_if_exist:NT #1
7090   {
7091     \box_set_eq:NN #1 #2
7092     \coffin_set_eq_structure:NN #1 #2
7093   }
7094 }
7095 \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
\l_coffin_aligned_coffin coffin is set as a box as the full coffin-setting system needs some material which is not
\l_coffin_aligned_internal_coffin yet available.

```

7096 \coffin_new:N \c_empty_coffin
7097 \hbox_set:Nn \c_empty_coffin { }
7098 \coffin_new:N \l_coffin_aligned_coffin
7099 \coffin_new:N \l_coffin_aligned_internal_coffin

```

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

194.3 Measuring coffins

`\coffin_dp:N` Coffins are just boxes when it comes to measurement. However, semantically a separate set of functions are required.

```
\coffin_dp:c
\coffin_ht:N
\coffin_ht:c
\coffin_wd:N
\coffin_wd:c
7100 \cs_new_eq:NN \coffin_dp:N \box_dp:N
7101 \cs_new_eq:NN \coffin_dp:c \box_dp:c
7102 \cs_new_eq:NN \coffin_ht:N \box_ht:N
7103 \cs_new_eq:NN \coffin_ht:c \box_ht:c
7104 \cs_new_eq:NN \coffin_wd:N \box_wd:N
7105 \cs_new_eq:NN \coffin_wd:c \box_wd:c
```

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

194.4 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.

```
7106 \cs_new_protected_nopar:Npn \coffin_get_pole:NnN #1#2#3
7107 {
7108   \prop_get:cnNF
7109     { l_coffin_poles_ \int_value:w #1 _prop } {#2} #3
7110   {
7111     \msg_kernel_error:nxxx { coffins } { unknown-coffin-pole }
7112     {#2} { \token_to_str:N #1 }
7113     \tl_set:Nn #3 { { 0 pt } { 0 pt } { 0 pt } { 0 pt } }
7114   }
7115 }
```

(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.

```
7116 \cs_new_protected_nopar:Npn \coffin_reset_structure:N #1
7117 {
7118   \prop_set_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7119     \c_coffin_corners_prop
7120   \prop_set_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7121     \c_coffin_poles_prop
7122 }
```

(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
7123 \cs_new_protected_nopar:Npn \coffin_set_eq_structure:NN #1#2
7124 {
7125   \prop_set_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7126     { l_coffin_corners_ \int_value:w #2 _prop }
7127   \prop_set_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7128     { l_coffin_poles_ \int_value:w #2 _prop }
7129 }
7130 \cs_new_protected_nopar:Npn \coffin_gset_eq_structure:NN #1#2
7131 {
```

```

7132 \prop_gset_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7133 { l_coffin_corners_ \int_value:w #2 _prop }
7134 \prop_gset_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7135 { l_coffin_poles_ \int_value:w #2 _prop }
7136 }

```

(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
7137 \cs_new_protected_nopar:Npn \coffin_set_user_dimensions:N #1
7138 {
7139   \cs_set_eq:NN \coffin_saved_Height: \Height
7140   \cs_set_eq:NN \coffin_saved_Depth: \Depth
7141   \cs_set_eq:NN \coffin_saved_TotalHeight: \TotalHeight
7142   \cs_set_eq:NN \coffin_saved_Width: \Width
7143   \cs_set_eq:NN \Height \l_coffin_Height_dim
7144   \cs_set_eq:NN \Depth \l_coffin_Depth_dim
7145   \cs_set_eq:NN \TotalHeight \l_coffin_TotalHeight_dim
7146   \cs_set_eq:NN \Width \l_coffin_Width_dim
7147   \dim_set:Nn \Height { \box_ht:N #1 }
7148   \dim_set:Nn \Depth { \box_dp:N #1 }
7149   \dim_set:Nn \TotalHeight { \box_ht:N #1 + \box_dp:N #1 }
7150   \dim_set:Nn \Width { \box_wd:N #1 }
7151 }
7152 \cs_new_protected_nopar:Npn \coffin_end_user_dimensions:
7153 {
7154   \cs_set_eq:NN \Height \coffin_saved_Height:
7155   \cs_set_eq:NN \Depth \coffin_saved_Depth:
7156   \cs_set_eq:NN \TotalHeight \coffin_saved_TotalHeight:
7157   \cs_set_eq:NN \Width \coffin_saved_Width:
7158 }

```

(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:cnm
\coffin_set_vertical_pole:Nnn
\coffin_set_vertical_pole:cnm
\coffin_set_pole:Nnn
\coffin_set_pole:Nnx
7159 \cs_new_protected_nopar:Npn \coffin_set_horizontal_pole:Nnn #1#2#3
7160 {
7161   \coffin_if_exist:NT #1
7162   {
7163     \coffin_set_user_dimensions:N #1
7164     \coffin_set_pole:Nnx #1 {#2}
7165     {
7166       { 0 pt } { \dim_eval:n {#3} }
7167       { 1000 pt } { 0 pt }
7168     }
7169     \coffin_end_user_dimensions:
7170   }
7171 }

```

```

7172 \cs_new_protected_nopar:Npn \coffin_set_vertical_pole:Nnn #1#2#3
7173 {
7174   \coffin_if_exist:NT #1
7175   {
7176     \coffin_set_user_dimensions:N #1
7177     \coffin_set_pole:Nnx #1 {#2}
7178     {
7179       { \dim_eval:n {#3} } { 0 pt }
7180       { 0 pt } { 1000 pt }
7181     }
7182     \coffin_end_user_dimensions:
7183   }
7184 }
7185 \cs_new_protected_nopar:Npn \coffin_set_pole:Nnn #1#2#3
7186 { \prop_put:cnn { l_coffin_poles_ \int_value:w #1 _prop } {#2} {#3} }
7187 \cs_generate_variant:Nn \coffin_set_horizontal_pole:Nnn { c }
7188 \cs_generate_variant:Nn \coffin_set_vertical_pole:Nnn { c }
7189 \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.

```

7190 \cs_new_protected_nopar:Npn \coffin_update_corners:N #1
7191 {
7192   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tl }
7193   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7194   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tr }
7195   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7196   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { bl }
7197   { { 0 pt } { \dim_eval:n { - \box_dp:N #1 } } }
7198   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { br }
7199   { { \dim_use:N \box_wd:N #1 } { \dim_eval:n { - \box_dp:N #1 } } }
7200 }

```

(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.

```

7201 \cs_new_protected_nopar:Npn \coffin_update_poles:N #1
7202 {
7203   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { hc }
7204   {
7205     { \dim_eval:n { 0.5 \box_wd:N #1 } }
7206     { 0 pt } { 0 pt } { 1000 pt }
7207   }
7208   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { r }
7209   {

```

```

7210     { \dim_use:N \box_wd:N #1 }
7211     { 0 pt } { 0 pt } { 1000 pt }
7212   }
7213   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { vc }
7214   {
7215     { 0 pt }
7216     { \dim_eval:n { ( \box_ht:N #1 - \box_dp:N #1 ) / 2 } }
7217     { 1000 pt }
7218     { 0 pt }
7219   }
7220   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { t }
7221   {
7222     { 0 pt }
7223     { \dim_use:N \box_ht:N #1 }
7224     { 1000 pt }
7225     { 0 pt }
7226   }
7227   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { b }
7228   {
7229     { 0 pt }
7230     { \dim_eval:n { - \box_dp:N #1 } }
7231     { 1000 pt }
7232     { 0 pt }
7233   }
7234 }

```

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

194.5 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 `\coffin_calculate_intersection:nnnnnnnn` an error trap is needed.

```

7235 \cs_new_protected_nopar:Npn \coffin_calculate_intersection:Nnn #1#2#3
7236 {
7237   \coffin_get_pole:NnN #1 {#2} \l_coffin_pole_a_tl
7238   \coffin_get_pole:NnN #1 {#3} \l_coffin_pole_b_tl
7239   \bool_set_false:N \l_coffin_error_bool
7240   \exp_last_two_unbraced:Noo
7241     \coffin_calculate_intersection:nnnnnnnn
7242     \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7243   \bool_if:NT \l_coffin_error_bool
7244   {
7245     \msg_kernel_error:nn { coffins } { no-pole-intersection }
7246     \dim_zero:N \l_coffin_x_dim
7247     \dim_zero:N \l_coffin_y_dim
7248   }
7249 }

```

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.

```

7250 \cs_new_protected_nopar:Npn \coffin_calculate_intersection:nnnnnnnn
7251   #1#2#3#4#5#6#7#8
7252   {
7253     \dim_compare:nNnTF {#3} = { \c_zero_dim }

```

The case where the first pole is vertical. So the x -component of the interaction will be at a . There is then a test on the second pole: if it is also vertical then there is an error.

```

7254   {
7255     \dim_set:Nn \l_coffin_x_dim {#1}
7256     \dim_compare:nNnTF {#7} = \c_zero_dim
7257     { \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.

```

7258   {
7259     \dim_compare:nNnTF {#8} = \c_zero_dim
7260     { \dim_set:Nn \l_coffin_y_dim {#6} }
7261     {
7262       \coffin_calculate_intersection_aux:nnnnnN
7263       {#1} {#5} {#6} {#7} {#8} \l_coffin_y_dim
7264     }
7265   }
7266 }

```

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.

```

7267   {
7268     \dim_compare:nNnTF {#4} = \c_zero_dim
7269     {
7270       \dim_set:Nn \l_coffin_y_dim {#2}
7271       \dim_compare:nNnTF {#8} = { \c_zero_dim }
7272       { \bool_set_true:N \l_coffin_error_bool }
7273       {
7274         \dim_compare:nNnTF {#7} = \c_zero_dim
7275         { \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.

```

7276         {
7277             \coffin_calculate_intersection_aux:nnnnnN
7278             {#2} {#6} {#5} {#8} {#7} \l_coffin_x_dim
7279         }
7280     }
7281 }

```

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.

```

7282 {
7283     \dim_compare:nNnTF {#7} = \c_zero_dim
7284     {
7285         \dim_set:Nn \l_coffin_x_dim {#5}
7286         \coffin_calculate_intersection_aux:nnnnnN
7287         {#5} {#1} {#2} {#3} {#4} \l_coffin_y_dim
7288     }
7289     {
7290         \dim_compare:nNnTF {#8} = \c_zero_dim
7291         {
7292             \dim_set:Nn \l_coffin_x_dim {#6}
7293             \coffin_calculate_intersection_aux:nnnnnN
7294             {#6} {#2} {#1} {#4} {#3} \l_coffin_x_dim
7295         }

```

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.

```

7296 {
7297     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#3}
7298     \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#4}
7299     \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#7}
7300     \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#8}
7301     \fp_div:Nn \l_coffin_calc_b_fp \l_coffin_calc_a_fp
7302     \fp_div:Nn \l_coffin_calc_d_fp \l_coffin_calc_c_fp
7303     \fp_compare:nNnTF
7304         \l_coffin_calc_b_fp = \l_coffin_calc_d_fp
7305     { \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.

```

7306 {
7307     \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#6}
7308     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7309     \fp_sub:Nn \l_coffin_calc_result_fp

```

```

7310         { \l_coffin_calc_a_fp }
7311     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#1}
7312     \fp_mul:Nn \l_coffin_calc_a_fp
7313         { \l_coffin_calc_b_fp }
7314     \fp_add:Nn \l_coffin_calc_result_fp
7315         { \l_coffin_calc_a_fp }
7316     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#5}
7317     \fp_mul:Nn \l_coffin_calc_a_fp
7318         { \l_coffin_calc_d_fp }
7319     \fp_sub:Nn \l_coffin_calc_result_fp
7320         { \l_coffin_calc_a_fp }
7321     \fp_sub:Nn \l_coffin_calc_b_fp
7322         { \l_coffin_calc_d_fp }
7323     \fp_div:Nn \l_coffin_calc_result_fp
7324         { \l_coffin_calc_b_fp }
7325     \dim_set:Nn \l_coffin_x_dim
7326         { \fp_to_dim:N \l_coffin_calc_result_fp }
7327     \coffin_calculate_intersection_aux:nnnnnN
7328         { \l_coffin_x_dim }
7329         {#5} {#6} {#8} {#7} \l_coffin_y_dim
7330     }
7331 }
7332 }
7333 }
7334 }
7335 }

```

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.

```

7336 \cs_new_protected_nopar:Npn \coffin_calculate_intersection_aux:nnnnnN
7337     #1#2#3#4#5#6
7338     {
7339         \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#1}
7340         \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7341         \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#3}
7342         \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#4}
7343         \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#5}
7344         \fp_sub:Nn \l_coffin_calc_result_fp { \l_coffin_calc_a_fp }
7345         \fp_div:Nn \l_coffin_calc_result_fp { \l_coffin_calc_d_fp }
7346         \fp_mul:Nn \l_coffin_calc_result_fp { \l_coffin_calc_c_fp }
7347         \fp_add:Nn \l_coffin_calc_result_fp { \l_coffin_calc_b_fp }
7348         \dim_set:Nn #6 { \fp_to_dim:N \l_coffin_calc_result_fp }
7349     }

```

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

194.6 Aligning and typesetting of coffins

`\coffin_join:NnnNnnnn` 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.

```

7350 \cs_new_protected_nopar:Npn \coffin_join:NnnNnnnn #1#2#3#4#5#6#7#8
7351 {
7352   \coffin_align:NnnNnnnnN
7353   #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.

```

7354   \hbox_set:Nn \l_coffin_aligned_coffin
7355   {
7356     \dim_compare:nNnT { \l_coffin_offset_x_dim } < \c_zero_dim
7357     { \tex_kern:D -\l_coffin_offset_x_dim }
7358     \hbox_unpack:N \l_coffin_aligned_coffin
7359     \dim_set:Nn \l_coffin_tmp_dim
7360     { \l_coffin_offset_x_dim - \box_wd:N #1 + \box_wd:N #4 }
7361     \dim_compare:nNnT \l_coffin_tmp_dim < \c_zero_dim
7362     { \tex_kern:D -\l_coffin_tmp_dim }
7363   }

```

The coffin structure is reset, and the corners are cleared: only those from the two parent coffins are needed.

```

7364   \coffin_reset_structure:N \l_coffin_aligned_coffin
7365   \prop_clear:c
7366   { \l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _ prop }
7367   \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.

```

7368   \dim_compare:nNnTF \l_coffin_offset_x_dim < \c_zero_dim
7369   {
7370     \coffin_offset_poles:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7371     \coffin_offset_poles:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7372     \coffin_offset_corners:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7373     \coffin_offset_corners:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7374   }
7375   {
7376     \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7377     \coffin_offset_poles:Nnn #4
7378     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7379     \coffin_offset_corners:Nnn #1 { 0 pt } { 0 pt }
7380     \coffin_offset_corners:Nnn #4
7381     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }

```

```

7382     }
7383     \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7384     \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7385   }
7386   \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:cnNnnnnnn The function used when marking a position is hear also as it is similar but without the structure updates.
 \coffin_attach:Nnncnnnn
 \coffin_attach_mark:NnnNnnnn

```

7387   \cs_new_protected_nopar:Npn \coffin_attach:NnnNnnnn #1#2#3#4#5#6#7#8
7388   {
7389     \coffin_align:NnnNnnnnN
7390     #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7391     \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7392     \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7393     \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7394     \coffin_reset_structure:N \l_coffin_aligned_coffin
7395     \prop_set_eq:cc
7396     { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7397     { l_coffin_corners_ \int_value:w #1 _prop }
7398     \coffin_update_poles:N \l_coffin_aligned_coffin
7399     \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7400     \coffin_offset_poles:Nnn #4
7401     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7402     \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7403     \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7404   }
7405   \cs_new_protected_nopar:Npn \coffin_attach_mark:NnnNnnnn #1#2#3#4#5#6#7#8
7406   {
7407     \coffin_align:NnnNnnnnN
7408     #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7409     \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7410     \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7411     \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7412     \box_set_eq:NN #1 \l_coffin_aligned_coffin
7413   }
7414   \cs_generate_variant:Nn \coffin_attach:NnnNnnnn { c , Nnnc , cnnc }
  
```

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

\coffin_align:NnnNnnnnN The internal function aligns the two coffins into a third one, but performs no corrections on the resulting coffin poles. The process begins by finding the points of intersection for the poles for each of the input coffins. Those for the first coffin are worked out after those for the second coffin, as this allows the ‘primed’ storage area to be used for the second coffin. The ‘real’ box offsets are then calculated, before using these to re-box the input

coffins. The default poles are then set up, but the final result will depend on how the bounding box is being handled.

```

7415 \cs_new_protected_nopar:Npn \coffin_align:NnnNnnnnN #1#2#3#4#5#6#7#8#9
7416 {
7417   \coffin_calculate_intersection:Nnn #4 {#5} {#6}
7418   \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7419   \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7420   \coffin_calculate_intersection:Nnn #1 {#2} {#3}
7421   \dim_set:Nn \l_coffin_offset_x_dim
7422     { \l_coffin_x_dim - \l_coffin_x_prime_dim + #7 }
7423   \dim_set:Nn \l_coffin_offset_y_dim
7424     { \l_coffin_y_dim - \l_coffin_y_prime_dim + #8 }
7425   \hbox_set:Nn \l_coffin_aligned_internal_coffin
7426     {
7427     \box_use:N #1
7428     \tex_kern:D -\box_wd:N #1
7429     \tex_kern:D \l_coffin_offset_x_dim
7430     \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #4 }
7431   }
7432   \coffin_set_eq:NN #9 \l_coffin_aligned_internal_coffin
7433 }

```

(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.

```

7434 \cs_new_protected_nopar:Npn \coffin_offset_poles:Nnn #1#2#3
7435 {
7436   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7437     { \coffin_offset_pole:Nnnnnnn #1 {##1} ##2 {#2} {#3} }
7438 }
7439 \cs_new_protected_nopar:Npn \coffin_offset_pole:Nnnnnnn #1#2#3#4#5#6#7#8
7440 {
7441   \dim_set:Nn \l_coffin_x_dim { #3 + #7 }
7442   \dim_set:Nn \l_coffin_y_dim { #4 + #8 }
7443   \tl_if_in:nnTF {#2} { - }
7444     { \tl_set:Nn \l_coffin_tmp_tl { {#2} } }
7445     { \tl_set:Nn \l_coffin_tmp_tl { { #1 - #2 } } }
7446   \exp_last_unbraced:NNo \coffin_set_pole:Nnx \l_coffin_aligned_coffin
7447     { \l_coffin_tmp_tl }
7448   {
7449     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7450     {#5} {#6}
7451   }
7452 }

```

(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.

```

7453 \cs_new_protected_nopar:Npn \coffin_offset_corners:Nnn #1#2#3
7454 {
7455   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7456   { \coffin_offset_corner:Nnnnn #1 {##1} ##2 {#2} {#3} }
7457 }
7458 \cs_new_protected_nopar:Npn \coffin_offset_corner:Nnnnn #1#2#3#4#5#6
7459 {
7460   \prop_put:cnx
7461   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7462   { #1 - #2 }
7463   {
7464     { \dim_eval:n { #3 + #5 } }
7465     { \dim_eval:n { #4 + #6 } }
7466   }
7467 }

```

(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.

```

7468 \cs_new_protected_nopar:Npn \coffin_update_vertical_poles:NNN #1#2#3
7469 {
7470   \coffin_get_pole:NnN #3 { #1 -T } \l_coffin_pole_a_tl
7471   \coffin_get_pole:NnN #3 { #2 -T } \l_coffin_pole_b_tl
7472   \exp_last_two_unbraced:Noo \coffin_update_T:nnnnnnnnN
7473   \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7474   \coffin_get_pole:NnN #3 { #1 -B } \l_coffin_pole_a_tl
7475   \coffin_get_pole:NnN #3 { #2 -B } \l_coffin_pole_b_tl
7476   \exp_last_two_unbraced:Noo \coffin_update_B:nnnnnnnnN
7477   \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7478 }
7479 \cs_new_protected_nopar:Npn \coffin_update_T:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7480 {
7481   \dim_compare:nNnTF {#2} < {#6}
7482   {
7483     \coffin_set_pole:Nnx #9 { T }
7484     { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7485   }
7486   {
7487     \coffin_set_pole:Nnx #9 { T }
7488     { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7489   }
7490 }
7491 \cs_new_protected_nopar:Npn \coffin_update_B:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7492 {

```

```

7493 \dim_compare:nNnTF {#2} < {#6}
7494 {
7495   \coffin_set_pole:Nnx #9 { B }
7496   { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7497 }
7498 {
7499   \coffin_set_pole:Nnx #9 { B }
7500   { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7501 }
7502 }

```

(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_typeset:cnnnn` 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.

```

7503 \cs_new_protected_nopar:Npn \coffin_typeset:Nnnnn #1#2#3#4#5
7504 {
7505   \coffin_align:NnnNnnnnN \c_empty_coffin { H } { 1 }
7506   #1 {#2} {#3} {#4} {#5} \l_coffin_aligned_coffin
7507   \hbox_unpack:N \c_empty_box
7508   \box_use:N \l_coffin_aligned_coffin
7509 }
7510 \cs_generate_variant:Nn \coffin_typeset:Nnnnn { c }

```

(End definition for `\coffin_typeset:Nnnnn` and `\coffin_typeset:cnnnn`. These functions are documented on page ??.)

194.7 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.

```

7511 \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.

```

7512 \dim_new:N \l_coffin_bounding_shift_dim

```

(End definition for `\l_coffin_bounding_shift_dim`. This function is documented on page ??.)

`\l_coffin_left_corner_dim` These are used to hold maxima for the various corner values: these thus define the
`\l_coffin_right_corner_dim` minimum size of the bounding box after rotation.
`\l_coffin_bottom_corner_dim`
`\l_coffin_top_corner_dim`

```

7513 \dim_new:N \l_coffin_left_corner_dim
7514 \dim_new:N \l_coffin_right_corner_dim
7515 \dim_new:N \l_coffin_bottom_corner_dim
7516 \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.

```

7517 \cs_new_protected_nopar:Npn \coffin_rotate:Nn #1#2
7518 {
7519   \fp_set:Nn \l_coffin_tmp_fp {#2}
7520   \fp_div:Nn \l_coffin_tmp_fp { 180 }
7521   \fp_mul:Nn \l_coffin_tmp_fp { \c_pi_fp }
7522   \fp_sin:Nn \l_coffin_sin_fp { \l_coffin_tmp_fp }
7523   \fp_cos:Nn \l_coffin_cos_fp { \l_coffin_tmp_fp }

```

The corners and poles of the coffin can now be rotated around the origin. This is best achieved using mapping functions.

```

7524 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7525 { \coffin_rotate_corner:Nnnn #1 {##1} ##2 }
7526 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7527 { \coffin_rotate_pole:Nnnnn #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.

```

7528 \coffin_set_bounding:N #1
7529 \prop_map_inline:Nn \l_coffin_bounding_prop
7530 { \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.

```

7531 \coffin_find_corner_maxima:N #1
7532 \coffin_find_bounding_shift:
7533 \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.

```

7534 \hbox_set:Nn #1
7535 {
7536   \tex_kern:D \l_coffin_bounding_shift_dim
7537   \tex_kern:D -\l_coffin_left_corner_dim
7538   \box_move_down:nn { \l_coffin_bottom_corner_dim }
7539   { \box_use:N #1 }
7540 }

```

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.

```

7541 \box_set_ht:Nn #1
7542 { \l_coffin_top_corner_dim - \l_coffin_bottom_corner_dim }

```

```

7543 \box_set_dp:Nn #1 { 0 pt }
7544 \box_set_wd:Nn #1
7545 { \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.

```

7546 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7547 { \coffin_shift_corner:Nnnn #1 {##1} ##2 }
7548 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7549 { \coffin_shift_pole:Nnnnnn #1 {##1} ##2 }
7550 }
7551 \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.

```

7552 \cs_new_protected_nopar:Npn \coffin_set_bounding:N #1
7553 {
7554   \prop_put:Nnx \l_coffin_bounding_prop { tl }
7555   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7556   \prop_put:Nnx \l_coffin_bounding_prop { tr }
7557   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7558   \dim_set:Nn \l_coffin_tmp_dim { - \box_dp:N #1 }
7559   \prop_put:Nnx \l_coffin_bounding_prop { bl }
7560   { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } }
7561   \prop_put:Nnx \l_coffin_bounding_prop { br }
7562   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_tmp_dim } }
7563 }

```

(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 from the reference point. The same treatment is used for the corners of the material itself and the bounding box.

\coffin_rotate_corner:Nnnn

```

7564 \cs_new_protected_nopar:Npn \coffin_rotate_bounding:nnn #1#2#3
7565 {
7566   \coffin_rotate_vector:nnNN {#2} {#3} \l_coffin_x_dim \l_coffin_y_dim
7567   \prop_put:Nnx \l_coffin_bounding_prop {#1}
7568   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7569 }
7570 \cs_new_protected_nopar:Npn \coffin_rotate_corner:Nnnn #1#2#3#4
7571 {
7572   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7573   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7574   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7575 }

```

(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.

```

7576 \cs_new_protected_nopar:Npn \coffin_rotate_pole:Nnnnnn #1#2#3#4#5#6
7577 {
7578   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7579   \coffin_rotate_vector:nnNN {#5} {#6}
7580   \l_coffin_x_prime_dim \l_coffin_y_prime_dim
7581   \coffin_set_pole:Nnx #1 {#2}
7582   {
7583     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7584     { \dim_use:N \l_coffin_x_prime_dim }
7585     { \dim_use:N \l_coffin_y_prime_dim }
7586   }
7587 }

```

(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.

```

7588 \cs_new_protected_nopar:Npn \coffin_rotate_vector:nnNN #1#2#3#4
7589 {
7590   \fp_set_from_dim:Nn \l_coffin_x_fp {#1}
7591   \fp_set_from_dim:Nn \l_coffin_y_fp {#2}
7592   \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
7593   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7594   \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
7595   \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
7596   \fp_sub:Nn \l_coffin_x_prime_fp { \l_coffin_sin_fp }
7597   \fp_sub:Nn \l_coffin_y_prime_fp { \l_coffin_sin_fp }
7598   \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
7599   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7600   \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_sin_fp }
7601   \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_sin_fp }
7602   \fp_add:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
7603   \fp_add:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
7604 }

```

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

`\coffin_find_corner_maxima:N`
`\coffin_find_corner_maxima_aux:nn` The idea here is to find the extremities of the content of the coffin. This is done by 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.

```

7605 \cs_new_protected_nopar:Npn \coffin_find_corner_maxima:N #1
7606 {
7607   \dim_set:Nn \l_coffin_top_corner_dim { -\c_max_dim }
7608   \dim_set:Nn \l_coffin_right_corner_dim { -\c_max_dim }
7609   \dim_set:Nn \l_coffin_bottom_corner_dim { \c_max_dim }

```

```

7610 \dim_set:Nn \l_coffin_left_corner_dim { \c_max_dim }
7611 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7612 { \coffin_find_corner_maxima_aux:nn ##2 }
7613 }
7614 \cs_new_protected_nopar:Npn \coffin_find_corner_maxima_aux:nn #1#2
7615 {
7616 \dim_set_min:Nn \l_coffin_left_corner_dim {#1}
7617 \dim_set_max:Nn \l_coffin_right_corner_dim {#1}
7618 \dim_set_min:Nn \l_coffin_bottom_corner_dim {#2}
7619 \dim_set_max:Nn \l_coffin_top_corner_dim {#2}
7620 }

```

(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. However, there is only one value needed here and a fixed input property list, so things are a bit clearer.

```

7621 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift:
7622 {
7623 \dim_set:Nn \l_coffin_bounding_shift_dim { \c_max_dim }
7624 \prop_map_inline:Nn \l_coffin_bounding_prop
7625 { \coffin_find_bounding_shift_aux:nn ##2 }
7626 }
7627 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift_aux:nn #1#2
7628 { \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 the x - and y -components. For the poles, this means that the direction vector is unchanged.

```

7629 \cs_new_protected_nopar:Npn \coffin_shift_corner:Nnnn #1#2#3#4
7630 {
7631 \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7632 {
7633 { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7634 { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7635 }
7636 }
7637 \cs_new_protected_nopar:Npn \coffin_shift_pole:Nnnnnn #1#2#3#4#5#6
7638 {
7639 \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } {#2}
7640 {
7641 { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7642 { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7643 {#5} {#6}
7644 }
7645 }

```

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

194.8 Resizing coffins

`\l_coffin_scale_x_fp` Storage for the scaling factors in x and y , respectively.

```
\l_coffin_scale_y_fp 7646 \fp_new:N \l_coffin_scale_x_fp
7647 \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 7648 \dim_new:N \l_coffin_scaled_total_height_dim
7649 \dim_new:N \l_coffin_scaled_width_dim
(End definition for \l_coffin_scaled_total_height_dim. This function is documented on page ??.)
```

`\coffin_resize:Nnn` Resizing a coffin begins by setting up the user-friendly names for the dimensions of the coffin box. The new sizes are then turned into scale factor. This is the same operation as takes place for the underlying box, but that operation is grouped and so the same calculation is done here.

```
\coffin_resize:cnn 7650 \cs_new_protected_nopar:Npn \coffin_resize:Nnn #1#2#3
7651 {
7652   \coffin_set_user_dimensions:N #1
7653   \box_resize:Nnn #1 {#2} {#3}
7654   \fp_set_from_dim:Nn \l_coffin_scale_x_fp {#2}
7655   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \Width }
7656   \fp_div:Nn \l_coffin_scale_x_fp { \l_coffin_tmp_fp }
7657   \fp_set_from_dim:Nn \l_coffin_scale_y_fp {#3}
7658   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \TotalHeight }
7659   \fp_div:Nn \l_coffin_scale_y_fp { \l_coffin_tmp_fp }
7660   \coffin_resize_common:Nnn #1 {#2} {#3}
7661 }
7662 \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.

```
7663 \cs_new_protected_nopar:Npn \coffin_resize_common:Nnn #1#2#3
7664 {
7665   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7666   { \coffin_scale_corner:Nnnn #1 {##1} ##2 }
7667   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7668   { \coffin_scale_pole:Nnnnnn #1 {##1} ##2 }
```

Negative x -scaling values will place the poles in the wrong location: this is corrected here.

```
7669 \fp_compare:NNNT \l_coffin_scale_x_fp < \c_zero_fp
7670 {
7671   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7672   { \coffin_x_shift_corner:Nnnn #1 {##1} ##2 }
7673   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
```

```

7674         { \coffin_x_shift_pole:Nnnnnn #1 {##1} ##2 }
7675     }
7676     \coffin_end_user_dimensions:
7677 }

```

(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 \TeX way as this works properly with floating point values without needing to use the `fp` module.

```

7678 \cs_new_protected_nopar:Npn \coffin_scale:Nnn #1#2#3
7679 {
7680     \box_scale:Nnn #1 {#2} {#3}
7681     \coffin_set_user_dimensions:N #1
7682     \fp_set:Nn \l_coffin_scale_x_fp {#2}
7683     \fp_set:Nn \l_coffin_scale_y_fp {#3}
7684     \fp_compare:NNNTF \l_coffin_scale_y_fp > \c_zero_fp
7685         { \l_coffin_scaled_total_height_dim #3 \TotalHeight }
7686         { \l_coffin_scaled_total_height_dim -#3 \TotalHeight }
7687     \fp_compare:NNNTF \l_coffin_scale_x_fp > \c_zero_fp
7688         { \l_coffin_scaled_width_dim -#2 \Width }
7689         { \l_coffin_scaled_width_dim #2 \Width }
7690     \coffin_resize_common:Nnn #1
7691         { \l_coffin_scaled_width_dim } { \l_coffin_scaled_total_height_dim }
7692 }
7693 \cs_generate_variant:Nn \coffin_scale:Nnn { c }

```

(End definition for \coffin_scale:Nnn and \coffin_scale:cnn. These functions are documented on page ??.)

`\coffin_scale_vector:nnNN` This functions scales a vector from the origin using the pre-set scale factors in x and y . This is a much less complex operation than rotation, and as a result the code is a lot clearer.

```

7694 \cs_new_protected_nopar:Npn \coffin_scale_vector:nnNN #1#2#3#4
7695 {
7696     \fp_set_from_dim:Nn \l_coffin_tmp_fp {#1}
7697     \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_x_fp }
7698     \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_tmp_fp }
7699     \fp_set_from_dim:Nn \l_coffin_tmp_fp {#2}
7700     \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_y_fp }
7701     \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_tmp_fp }
7702 }

```

(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`

```

7703 \cs_new_protected_nopar:Npn \coffin_scale_corner:Nnnn #1#2#3#4
7704 {
7705     \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7706     \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7707         { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }

```

```

7708 }
7709 \cs_new_protected_nopar:Npn \coffin_scale_pole:Nnnnnn #1#2#3#4#5#6
7710 {
7711   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7712   \coffin_set_pole:Nnx #1 {#2}
7713   {
7714     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7715     {#5} {#6}
7716   }
7717 }

```

(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.

```

7718 \cs_new_protected_nopar:Npn \coffin_x_shift_corner:Nnnn #1#2#3#4
7719 {
7720   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7721   {
7722     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7723   }
7724 }
7725 \cs_new_protected_nopar:Npn \coffin_x_shift_pole:Nnnnnn #1#2#3#4#5#6
7726 {
7727   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } {#2}
7728   {
7729     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7730     {#5} {#6}
7731   }
7732 }

```

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

194.9 Coffin diagnostics

\l_coffin_display_coffin Used for printing coffins with data structures attached.

```

\l_coffin_display_coord_coffin 7733 \coffin_new:N \l_coffin_display_coffin
\l_coffin_display_pole_coffin 7734 \coffin_new:N \l_coffin_display_coord_coffin
7735 \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.

```

7736 \prop_new:N \l_coffin_display_handles_prop
7737 \prop_put:Nnn \l_coffin_display_handles_prop { tl }
7738 { { b } { r } { -1 } { 1 } }
7739 \prop_put:Nnn \l_coffin_display_handles_prop { thc }
7740 { { b } { hc } { 0 } { 1 } }
7741 \prop_put:Nnn \l_coffin_display_handles_prop { tr }
7742 { { b } { l } { 1 } { 1 } }

```

```

7743 \prop_put:Nnn \l_coffin_display_handles_prop { vcl }
7744 { { vc } { r } { -1 } { 0 } }
7745 \prop_put:Nnn \l_coffin_display_handles_prop { vhc }
7746 { { vc } { hc } { 0 } { 0 } }
7747 \prop_put:Nnn \l_coffin_display_handles_prop { vcr }
7748 { { vc } { l } { 1 } { 0 } }
7749 \prop_put:Nnn \l_coffin_display_handles_prop { bl }
7750 { { t } { r } { -1 } { -1 } }
7751 \prop_put:Nnn \l_coffin_display_handles_prop { bhc }
7752 { { t } { hc } { 0 } { -1 } }
7753 \prop_put:Nnn \l_coffin_display_handles_prop { br }
7754 { { t } { l } { 1 } { -1 } }
7755 \prop_put:Nnn \l_coffin_display_handles_prop { Tl }
7756 { { t } { r } { -1 } { -1 } }
7757 \prop_put:Nnn \l_coffin_display_handles_prop { Thc }
7758 { { t } { hc } { 0 } { -1 } }
7759 \prop_put:Nnn \l_coffin_display_handles_prop { Tr }
7760 { { t } { l } { 1 } { -1 } }
7761 \prop_put:Nnn \l_coffin_display_handles_prop { Hl }
7762 { { vc } { r } { -1 } { 1 } }
7763 \prop_put:Nnn \l_coffin_display_handles_prop { Hhc }
7764 { { vc } { hc } { 0 } { 1 } }
7765 \prop_put:Nnn \l_coffin_display_handles_prop { Hr }
7766 { { vc } { l } { 1 } { 1 } }
7767 \prop_put:Nnn \l_coffin_display_handles_prop { Bl }
7768 { { b } { r } { -1 } { -1 } }
7769 \prop_put:Nnn \l_coffin_display_handles_prop { Bhc }
7770 { { b } { hc } { 0 } { -1 } }
7771 \prop_put:Nnn \l_coffin_display_handles_prop { Br }
7772 { { 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.

```

7773 \dim_new:N \l_coffin_display_offset_dim
7774 \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.

```

7775 \dim_new:N \l_coffin_display_x_dim
7776 \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.

```

7777 \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.

```

7778 \tl_new:N \l_coffin_display_font_tl
7779 <*initex>
7780 \tl_set:Nn \l_coffin_display_font_tl { } % TODO
7781 </initex>
7782 <*package>
7783 \tl_set:Nn \l_coffin_display_font_tl { \sfamily \tiny }
7784 </package>
(End definition for \l_coffin_display_font_tl. This function is documented on page ??.)

```

`\l_coffin_handles_tmp_prop` Used for displaying coffins, as the handles need to be stored in this case, at least temporarily.

```

7785 \prop_new:N \l_coffin_handles_tmp_prop
(End definition for \l_coffin_handles_tmp_prop. This function is documented on page ??.)

```

`\coffin_mark_handle:Nnnn` Marking a single handle is relatively easy. The standard attachment function is used, meaning that there are two calculations for the location. However, this is likely to be okay given the load expected. Contrast with the more optimised version for showing all handles which comes next.

`\coffin_mark_handle:cnnn`

`\coffin_mark_handle_aux:nnnnNnn`

```

7786 \cs_new_protected_nopar:Npn \coffin_mark_handle:Nnnn #1#2#3#4
7787 {
7788   \hcoffin_set:Nn \l_coffin_display_pole_coffin
7789   {
7790     <*initex>
7791     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
7792     </initex>
7793     <*package>
7794     \color {#4}
7795     \rule { 1 pt } { 1 pt }
7796   </package>
7797   }
7798   \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7799   \l_coffin_display_pole_coffin { hc } { vc } { 0 pt } { 0 pt }
7800   \hcoffin_set:Nn \l_coffin_display_coord_coffin
7801   {
7802     <*initex>
7803     % TODO
7804     </initex>
7805     <*package>
7806     \color {#4}
7807   </package>
7808   \l_coffin_display_font_tl
7809   ( \tl_to_str:n { #2 , #3 } )
7810   }
7811   \prop_get:NnN \l_coffin_display_handles_prop
7812   { #2 #3 } \l_coffin_tmp_tl
7813   \quark_if_no_value:NTF \l_coffin_tmp_tl
7814   {
7815     \prop_get:NnN \l_coffin_display_handles_prop

```

```

7816 { #3 #2 } \l_coffin_tmp_tl
7817 \quark_if_no_value:NTF \l_coffin_tmp_tl
7818 {
7819   \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7820   \l_coffin_display_coord_coffin { 1 } { vc }
7821   { 1 pt } { 0 pt }
7822 }
7823 {
7824   \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7825   \l_coffin_tmp_tl #1 {#2} {#3}
7826 }
7827 }
7828 {
7829   \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7830   \l_coffin_tmp_tl #1 {#2} {#3}
7831 }
7832 }
7833 \cs_new_protected_nopar:Npn \coffin_mark_handle_aux:nnnnNnn #1#2#3#4#5#6#7
7834 {
7835   \coffin_attach_mark:NnnNnnnn #5 {#6} {#7}
7836   \l_coffin_display_coord_coffin {#1} {#2}
7837   { #3 \l_coffin_display_offset_dim }
7838   { #4 \l_coffin_display_offset_dim }
7839 }
7840 \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
\coffin_display_handles:cn
  \coffin_display_handles_aux:nnnnnn
  \coffin_display_handles_aux:nnnn
\coffin_display_attach:Nnnnn

```

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.

```

7841 \cs_new_protected_nopar:Npn \coffin_display_handles:Nn #1#2
7842 {
7843   \hcoffin_set:Nn \l_coffin_display_pole_coffin
7844   {
7845     <*initex>
7846     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
7847     </initex>
7848     <*package>
7849     \color {#2}
7850     \rule { 1 pt } { 1 pt }
7851     </package>
7852   }
7853   \prop_set_eq:Nc \l_coffin_display_poles_prop
7854   { \l_coffin_poles_ \int_value:w #1 _prop }
7855   \coffin_get_pole:NnN #1 { H } \l_coffin_pole_a_tl
7856   \coffin_get_pole:NnN #1 { T } \l_coffin_pole_b_tl
7857   \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl

```

```

7858     { \prop_del:Nn \l_coffin_display_poles_prop { T } }
7859 \coffin_get_pole:NnN #1 { B } \l_coffin_pole_b_tl
7860 \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7861   { \prop_del:Nn \l_coffin_display_poles_prop { B } }
7862 \coffin_set_eq:NN \l_coffin_display_coffin #1
7863 \prop_clear:N \l_coffin_handles_tmp_prop
7864 \prop_map_inline:Nn \l_coffin_display_poles_prop
7865   {
7866     \prop_del:Nn \l_coffin_display_poles_prop {##1}
7867     \coffin_display_handles_aux:nnnnnn {##1} ##2 {##2}
7868   }
7869 \box_use:N \l_coffin_display_coffin
7870 }

```

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.

```

7871 \cs_new_protected_nopar:Npn \coffin_display_handles_aux:nnnnnn #1#2#3#4#5#6
7872 {
7873   \prop_map_inline:Nn \l_coffin_display_poles_prop
7874   {
7875     \bool_set_false:N \l_coffin_error_bool
7876     \coffin_calculate_intersection:nnnnnnnn {#2} {#3} {#4} {#5} ##2
7877     \bool_if:NF \l_coffin_error_bool
7878     {
7879       \dim_set:Nn \l_coffin_display_x_dim { \l_coffin_x_dim }
7880       \dim_set:Nn \l_coffin_display_y_dim { \l_coffin_y_dim }
7881       \coffin_display_attach:Nnnnn
7882         \l_coffin_display_pole_coffin { hc } { vc }
7883       { 0 pt } { 0 pt }
7884       \hcoffin_set:Nn \l_coffin_display_coord_coffin
7885       {
7886         <*initex>
7887           % TODO
7888         </initex>
7889         <*package>
7890           \color {#6}
7891         </package>
7892         \l_coffin_display_font_tl
7893         ( \tl_to_str:n { #1 , ##1 } )
7894       }
7895       \prop_get:NnN \l_coffin_display_handles_prop
7896       { #1 ##1 } \l_coffin_tmp_tl
7897       \quark_if_no_value:NTF \l_coffin_tmp_tl
7898       {
7899         \prop_get:NnN \l_coffin_display_handles_prop
7900         { ##1 #1 } \l_coffin_tmp_tl
7901         \quark_if_no_value:NTF \l_coffin_tmp_tl
7902         {
7903           \coffin_display_attach:Nnnnn

```

```

7904         \l_coffin_display_coord_coffin { 1 } { vc }
7905         { 1 pt } { 0 pt }
7906     }
7907     {
7908         \exp_last_unbraced:No
7909         \coffin_display_handles_aux:nnnn
7910         \l_coffin_tmp_tl
7911     }
7912 }
7913 {
7914     \exp_last_unbraced:No \coffin_display_handles_aux:nnnn
7915     \l_coffin_tmp_tl
7916 }
7917 }
7918 }
7919 }
7920 \cs_new_protected_nopar:Npn \coffin_display_handles_aux:nnnn #1#2#3#4
7921 {
7922     \coffin_display_attach:Nnnnn
7923     \l_coffin_display_coord_coffin {#1} {#2}
7924     { #3 \l_coffin_display_offset_dim }
7925     { #4 \l_coffin_display_offset_dim }
7926 }
7927 \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.

```

7928 \cs_new_protected_nopar:Npn \coffin_display_attach:Nnnnn #1#2#3#4#5
7929 {
7930     \coffin_calculate_intersection:Nnn #1 {#2} {#3}
7931     \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7932     \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7933     \dim_set:Nn \l_coffin_offset_x_dim
7934     { \l_coffin_display_x_dim - \l_coffin_x_prime_dim + #4 }
7935     \dim_set:Nn \l_coffin_offset_y_dim
7936     { \l_coffin_display_y_dim - \l_coffin_y_prime_dim + #5 }
7937     \hbox_set:Nn \l_coffin_aligned_coffin
7938     {
7939         \box_use:N \l_coffin_display_coffin
7940         \tex_kern:D -\box_wd:N \l_coffin_display_coffin
7941         \tex_kern:D \l_coffin_offset_x_dim
7942         \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #1 }
7943     }
7944     \box_set_ht:Nn \l_coffin_aligned_coffin
7945     { \box_ht:N \l_coffin_display_coffin }
7946     \box_set_dp:Nn \l_coffin_aligned_coffin
7947     { \box_dp:N \l_coffin_display_coffin }
7948     \box_set_wd:Nn \l_coffin_aligned_coffin
7949     { \box_wd:N \l_coffin_display_coffin }

```

```

7950     \box_set_eq:NN \l_coffin_display_coffin \l_coffin_aligned_coffin
7951   }

```

(End definition for `\coffin_display_handles:Nn` and `\coffin_display_handles:cn`. These functions are documented on page ??.)

`\coffin_show_structure:N` For showing the various internal structures attached to a coffin in a way that keeps things relatively readable. If there is no apparent structure then the code complains.

```

\coffin_show_structure:c
  \coffin_show_aux:n
  \coffin_show_aux:w
7952 \cs_new_protected_nopar:Npn \coffin_show_structure:N #1
7953 {
7954   \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
7955   {
7956     \iow_term:x
7957     {
7958       \iow_newline:
7959       Size-of~coffin~\token_to_str:N #1 : \iow_newline:
7960       > ~ ht~~~\dim_use:N \box_ht:N #1 \iow_newline:
7961       > ~ dp~~~\dim_use:N \box_dp:N #1 \iow_newline:
7962       > ~ wd~~~\dim_use:N \box_wd:N #1 \iow_newline:
7963     }
7964     \iow_term:x { Poles-of~coffin~\token_to_str:N #1 : }
7965     \tl_set:Nx \l_coffin_tmp_tl
7966     {
7967       \prop_map_function:cn
7968       { l_coffin_poles_ \int_value:w #1 _prop }
7969       \coffin_show_aux:nn
7970     }
7971     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
7972     { \exp_after:wN \coffin_show_aux:w \l_coffin_tmp_tl }
7973   }
7974   {
7975     \iow_term:x { ----No~poles~found---- }
7976     \tl_show:n { Is~this~really~a~coffin? }
7977   }
7978 }
7979 \cs_new:Npn \coffin_show_aux:nn #1#2
7980 {
7981   \iow_newline: > \c_space_tl \c_space_tl
7982   #1 \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
7983 }
7984 \cs_new_nopar:Npn \coffin_show_aux:w #1 > ~ { }
7985 \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 ??.)

194.10 Messages

```

7986 \msg_kernel_new:nnnn { coffins } { no-pole-intersection }
7987 { No~intersection~between~coffin~poles. }
7988 {

```

```

7989 \c_msg_coding_error_text_tl
7990 LaTeX~was~asked~to~find~the~intersection~between~two~poles,~
7991 but~they~do~not~have~a~unique~meeting~point:~
7992 the~value~(0~pt,~0~pt)~will~be~used.
7993 }
7994 \msg_kernel_new:nnnn { coffins } { unknown-coffin }
7995 { Unknown-coffin~'#1'. }
7996 { The~coffin~'#1'~was~never~defined. }
7997 \msg_kernel_new:nnnn { coffins } { unknown-coffin-pole }
7998 { Pole~'#1'~unknown~for~coffin~'#2'. }
7999 {
8000 \c_msg_coding_error_text_tl
8001 LaTeX~was~asked~to~find~a~typesetting~pole~for~a~coffin,~
8002 but~either~the~coffin~does~not~exist~or~the~pole~name~is~wrong.
8003 }
8004 </initex | package>

```

195 l3color Implementation

```

8005 <*initex | package>
8006 <*package>
8007 \ProvidesExplPackage
8008 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8009 \package_check_loaded_expl:
8010 </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.

```

8011 \cs_new_eq:NN \color_group_begin: \group_begin:
8012 \cs_new_protected_nopar:Npn \color_group_end:
8013 {
8014 \tex_par:D
8015 \group_end:
8016 }

```

(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”.

```

8017 <*initex>
8018 \cs_new_protected_nopar:Npn \color_ensure_current:
8019 { \driver_color_ensure_current: }
8020 </initex>
8021 <*package>
8022 \cs_new_protected_nopar:Npn \color_ensure_current: { \set@color }
8023 </package>
8024 </initex | package>

```

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

196 l3io implementation

```
8025 <*initex | package>
8026 <*package>
8027 \ProvidesExplPackage
8028   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
8029 \package_check_loaded_expl:
8030 </package>
```

196.1 Primitives

`\if_eof:w` The primitive conditional

```
8031 \cs_new_eq:NN \if_eof:w \tex_ifeof:D
      (End definition for \if_eof:w. This function is documented on page 143.)
```

196.2 Variables and constants

`\c_iow_term_stream` Here we allocate two output streams for writing to the transcript file only (`\c_iow_log_stream`) and to both the terminal and transcript file (`\c_iow_term_stream`). Both can be used to read from and have equivalent `\c_ior` versions.

```
\c_ior_term_stream
\c_ior_log_stream
\c_iow_log_stream
8032 \cs_new_eq:NN \c_iow_term_stream \c_sixteen
8033 \cs_new_eq:NN \c_ior_term_stream \c_sixteen
8034 \cs_new_eq:NN \c_iow_log_stream \c_minus_one
8035 \cs_new_eq:NN \c_ior_log_stream \c_minus_one
      (End definition for \c_iow_term_stream and \c_ior_term_stream. These functions are documented on page ??.)
```

`\c_iow_streams_tl` The list of streams available, by number.

```
\c_ior_streams_tl
8036 \tl_const:Nn \c_iow_streams_tl
8037 {
8038   \c_zero
8039   \c_one
8040   \c_two
8041   \c_three
8042   \c_four
8043   \c_five
8044   \c_six
8045   \c_seven
8046   \c_eight
8047   \c_nine
8048   \c_ten
8049   \c_eleven
8050   \c_twelve
8051   \c_thirteen
8052   \c_fourteen
8053   \c_fifteen
8054 }
8055 \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.

```
8056 \prop_new:N \g_iow_streams_prop
8057 \prop_new:N \g_ior_streams_prop
8058 \*package>
8059 \prop_put:Nnn \g_iow_streams_prop { 0 } { LaTeX2e~reserved }
8060 \prop_put:Nnn \g_iow_streams_prop { 1 } { LaTeX2e~reserved }
8061 \prop_put:Nnn \g_iow_streams_prop { 2 } { LaTeX2e~reserved }
8062 \prop_put:Nnn \g_ior_streams_prop { 0 } { LaTeX2e~reserved }
8063 \*package>
```

(End definition for `\g_iow_streams_prop` and `\g_ior_streams_prop`. These functions are documented 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.

```
8064 \int_new:N \l_iow_stream_int
8065 \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 ??.)

196.3 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      8066 \*initex>
\iow_raw_new:c      8067 \alloc_setup_type:nnn { ior } \c_zero \c_sixteen
                    8068 \cs_new_protected_nopar:Npn \ior_raw_new:N #1
                    8069 { \alloc_reg:nnn { ior } \tex_chardef:D #1 }
                    8070 \alloc_setup_type:nnn { iow } \c_zero \c_sixteen
                    8071 \cs_new_protected_nopar:Npn \iow_raw_new:N #1
                    8072 { \alloc_reg:nnn { iow } \tex_chardef:D #1 }
                    8073 \*initex>
                    8074 \*package>
                    8075 \cs_set_eq:NN \iow_raw_new:N \newwrite
                    8076 \cs_set_eq:NN \ior_raw_new:N \newread
                    8077 \*package>
                    8078 \cs_generate_variant:Nn \ior_raw_new:N { c }
                    8079 \cs_generate_variant:Nn \iow_raw_new:N { c }
```

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

`\ior_new:N` Reserving a new stream is done by defining the stream as the 16 (which must be in error).

```
\ior_new:c      8080 \cs_new_protected_nopar:Npn \ior_new:N #1
\iow_new:N      8081 {
\iow_new:c      8082   \chk_if_free_cs:N #1
```



```

8083 \tex_global:D \tex_chardef:D #1 16 ~
8084 }
8085 \cs_new_eq:NN \iow_new:N \ior_new:N
8086 \cs_generate_variant:Nn \ior_new:N { c }
8087 \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.

```

8088 \cs_new_protected_nopar:Npn \ior_open:Nn #1#2
8089 {
8090   \ior_close:N #1
8091   \int_set:Nn \l_ior_stream_int \c_sixteen
8092   \tl_map_function:NN \c_ior_streams_tl \ior_alloc_read:n
8093   \int_compare:nNnTF \l_ior_stream_int = \c_sixteen
8094     { \msg_kernel_error:nn { ior } { streams-exhausted } }
8095     {
8096       \ior_stream_alloc:N #1
8097       \prop_gput:NVn \g_ior_streams_prop \l_ior_stream_int {#2}
8098       \tex_openin:D #1#2 \scan_stop:
8099     }
8100 }
8101 \cs_new_protected_nopar:Npn \iow_open:Nn #1#2
8102 {
8103   \iow_close:N #1
8104   \int_set:Nn \l_iow_stream_int \c_sixteen
8105   \tl_map_function:NN \c_iow_streams_tl \iow_alloc_write:n
8106   \int_compare:nNnTF \l_iow_stream_int = \c_sixteen
8107     { \msg_kernel_error:nn { iow } { streams-exhausted } }
8108     {
8109       \iow_stream_alloc:N #1
8110       \prop_gput:NVn \g_iow_streams_prop \l_iow_stream_int {#2}
8111       \tex_immediate:D \tex_openout:D #1#2 \scan_stop:
8112     }
8113 }
8114 \cs_generate_variant:Nn \ior_open:Nn { c }
8115 \cs_generate_variant:Nn \iow_open:Nn { c }

```

(End definition for \ior_open:Nn and \iow_open:cn. These functions are documented on page ??.)

\ior_alloc_read:n These functions are used to see if a particular stream is available. The property list
\iow_alloc_write:n contains file names for streams in use, so any unused ones are for the taking.

```

8116 \cs_new_protected_nopar:Npn \iow_alloc_write:n #1
8117 {
8118   \prop_if_in:NnF \g_iow_streams_prop {#1}
8119   {
8120     \int_set:Nn \l_iow_stream_int {#1}

```

```

8121         \tl_map_break:
8122     }
8123 }
8124 \cs_new_protected_nopar:Npn \ior_alloc_read:n #1
8125 {
8126     \prop_if_in:NnF \g_iow_streams_prop {#1}
8127     {
8128         \int_set:Nn \l_ior_stream_int {#1}
8129         \tl_map_break:
8130     }
8131 }

```

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

\iow_stream_alloc:N Allocating a raw stream is much easier in IniTeX mode than for the package. For the
\ior_stream_alloc:N format, all streams will be allocated by l3io and so there is a simple check to see if a
\iow_stream_alloc_aux: raw stream is actually available. On the other hand, for the package there will be non-
\ior_stream_alloc_aux: managed streams. So if the managed one is not open, a check is made to see if some
\g_iow_tmp_stream other managed stream is available before deciding to open a new one. If a new one is
\g_ior_tmp_stream needed, we get the number allocated by L^AT_EX 2_ε to get “back on track” with allocation.

```

8132 \cs_new_protected_nopar:Npn \iow_stream_alloc:N #1
8133 {
8134     \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _stream }
8135     { \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _stream } }
8136     {
8137 <*package>
8138         \iow_stream_alloc_aux:
8139         \int_compare:nNnT \l_iow_stream_int = \c_sixteen
8140         {
8141             \iow_raw_new:N \g_iow_tmp_stream
8142             \int_set:Nn \l_iow_stream_int { \g_iow_tmp_stream }
8143             \cs_gset_eq:cN
8144             { g_iow_ \int_use:N \l_iow_stream_int _stream }
8145             \g_iow_tmp_stream
8146         }
8147 </package>
8148 <*initex>
8149         \iow_raw_new:c { g_iow_ \int_use:N \l_iow_stream_int _stream }
8150 </initex>
8151         \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _stream }
8152     }
8153 }
8154 <*package>
8155 \cs_new_protected_nopar:Npn \iow_stream_alloc_aux:
8156 {
8157     \int_incr:N \l_iow_stream_int
8158     \int_compare:nNnT \l_iow_stream_int < \c_sixteen
8159     {
8160         \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _stream }
8161         {

```

```

8162         \prop_if_in:NVT \g_iow_streams_prop \l_iow_stream_int
8163         { \ior_stream_alloc_aux: }
8164     }
8165     { \ior_stream_alloc_aux: }
8166 }
8167 }
8168 </package>
8169 \cs_new_protected_nopar:Npn \ior_stream_alloc:N #1
8170 {
8171     \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _stream }
8172     { \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _stream } }
8173     {
8174         <*package>
8175         \ior_stream_alloc_aux:
8176         \int_compare:nNnT \l_ior_stream_int = \c_sixteen
8177         {
8178             \ior_raw_new:N \g_ior_tmp_stream
8179             \int_set:Nn \l_ior_stream_int { \g_ior_tmp_stream }
8180             \cs_gset_eq:cN
8181             { g_ior_ \int_use:N \l_iow_stream_int _stream }
8182             \g_ior_tmp_stream
8183         }
8184         </package>
8185         <*initex>
8186         \ior_raw_new:c { g_ior_ \int_use:N \l_ior_stream_int _stream }
8187         </initex>
8188         \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _stream }
8189     }
8190 }
8191 <*package>
8192 \cs_new_protected_nopar:Npn \ior_stream_alloc_aux:
8193 {
8194     \int_incr:N \l_ior_stream_int
8195     \int_compare:nNnT \l_ior_stream_int < \c_sixteen
8196     {
8197         \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _stream }
8198         {
8199             \prop_if_in:NVT \g_iow_streams_prop \l_ior_stream_int
8200             { \ior_stream_alloc_aux: }
8201         }
8202         { \ior_stream_alloc_aux: }
8203     }
8204 }
8205 </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

```

8206 \cs_new_protected_nopar:Npn \ior_close:N #1
8207 {
8208   \cs_if_exist:NT #1
8209   {
8210     \int_compare:nNnF #1 = \c_minus_one
8211     {
8212       \int_compare:nNnF #1 = \c_sixteen
8213       { \tex_closein:D #1 }
8214       \prop_gdel:NV \g_ior_streams_prop #1
8215       \tex_global:D \tex_chardef:D #1 16 ~
8216     }
8217   }
8218 }
8219 \cs_new_protected_nopar:Npn \iow_close:N #1
8220 {
8221   \cs_if_exist:NT #1
8222   {
8223     \int_compare:nNnF #1 = \c_minus_one
8224     {
8225       \int_compare:nNnF #1 = \c_sixteen
8226       { \tex_closein:D #1 }
8227       \prop_gdel:NV \g_iow_streams_prop #1
8228       \tex_global:D \tex_chardef:D #1 16 ~
8229     }
8230   }
8231 }
8232 \cs_generate_variant:Nn \ior_close:N { c }
8233 \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”.
\iow_list_streams: 8234 \cs_new_protected_nopar:Npn \ior_list_streams:
\iow_show_aux:nn 8235 {
\ior_show_aux:nn 8236   \prop_if_empty:NTF \g_ior_streams_prop
8237   {
8238     \iow_term:x { No~input~streams~are~open }
8239     \tl_show:n { }
8240   }
8241   {
8242     \iow_term:x { The~following~input~streams~are~in~use: }
8243     \tl_set:Nx \l_prop_show_tl
8244     { \prop_map_function:NN \g_ior_streams_prop \ior_show_aux:nn }
8245     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
8246     { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
8247   }
8248 }
8249 \cs_new:Npn \ior_show_aux:nn #1#2
8250 {
8251   \iow_newline: > \c_space_tl \c_space_tl

```

```

8252     #1 \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
8253   }
8254   \cs_new_protected_nopar:Npn \iow_list_streams:
8255   {
8256     \prop_if_empty:NTF \g_iow_streams_prop
8257     {
8258       \iow_term:x { No~output~streams~are~open }
8259       \tl_show:n { }
8260     }
8261     {
8262       \iow_term:x { The~following~output~streams~are~in~use: }
8263       \tl_set:Nx \l_prop_show_tl
8264       { \prop_map_function:NN \g_iow_streams_prop \iow_show_aux:nn }
8265       \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
8266       { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
8267     }
8268   }
8269   \cs_new_eq:NN \iow_show_aux:nn \ior_show_aux:nn
      (End definition for \ior_list_streams:. This function is documented on page ??.)
      Text for the error messages.
8270   \msg_kernel_new:nnnn { iow } { streams-exhausted }
8271   { Output~streams~exhausted }
8272   {
8273     TeX~can~only~open~up~to~16~output~streams~at~one~time.\\
8274     All~16 are currently~in~use,~and~something~wanted~to~open
8275     another~one.
8276   }
8277   \msg_kernel_new:nnnn { ior } { streams-exhausted }
8278   { Input~streams~exhausted }
8279   {
8280     TeX~can~only~open~up~to~16~input~streams~at~one~time.\\
8281     All~16 are currently~in~use,~and~something~wanted~to~open
8282     another~one.
8283   }

```

196.4 Deferred writing

`\iow_shipout_x:Nn` First the easy part, this is the primitive.

```

\iow_shipout_x:Nx 8284 \cs_new_eq:NN \iow_shipout_x:Nn \tex_write:D
8285 \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 8286 \cs_new_protected_nopar:Npn \iow_shipout:Nn #1#2
8287 { \iow_shipout_x:Nn #1 { \exp_not:n {#2} } }
8288 \cs_generate_variant:Nn \iow_shipout:Nn { Nx }

```

(End definition for `\iow_shipout:Nn` and `\iow_shipout:Nx`. These functions are documented on page ??.)

196.5 Immediate writing

`\iow_now:Nx` An abbreviation for an often used operation, which immediately writes its second argument expanded to the output stream.

```
8289 \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.

```
8290 \cs_new_protected_nopar:Npn \iow_now:Nn #1#2
8291 { \iow_now:Nx #1 { \exp_not:n {#2} } }
```

(End definition for \iow_now:Nn. This function is documented on page 139.)

`\iow_log:n` Writing to the log and the terminal directly are relatively easy.

```
\iow_log:x 8292 \cs_set_protected_nopar:Npn \iow_log:x { \iow_now:Nx \c_iow_log_stream }
\iow_term:n 8293 \cs_new_protected_nopar:Npn \iow_log:n { \iow_now:Nn \c_iow_log_stream }
\iow_term:x 8294 \cs_set_protected_nopar:Npn \iow_term:x { \iow_now:Nx \c_iow_term_stream }
8295 \cs_new_protected_nopar:Npn \iow_term:n { \iow_now:Nn \c_iow_term_stream }
```

(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 8296 \cs_new_protected_nopar:Npn \iow_now_when_avail:Nn #1
8297 { \cs_if_free:NTF #1 { \use_none:n } { \iow_now:Nn #1 } }
8298 \cs_new_protected_nopar:Npn \iow_now_when_avail:Nx #1
8299 { \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 ??.)

196.6 Special characters for writing

`\iow_newline:` Global variable holding the character that forces a new line when something is written to an output stream

```
8300 \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.

```
8301 \cs_new_eq:NN \iow_char:N \cs_to_str:N
```

(End definition for \iow_char:N. This function is documented on page 140.)

196.7 Hard-wrapping lines based on length

The code here implements a generic hard-wrapping function. This is used by the messaging system, but is designed such that it is available for other uses.

<code>\l_iow_line_length_int</code>	<p>This is the “raw” length of a line which can be written to a file. The standard value is the line length typically used by <code>T_EXLive</code> and <code>MikT_EX</code>.</p> <pre> 8302 \int_new:N \l_iow_line_length_int 8303 \int_set:Nn \l_iow_line_length_int { 78 } </pre> <p><i>(End definition for <code>\l_iow_line_length_int</code>. This function is documented on page 141.)</i></p>
<code>\l_iow_target_length_int</code>	<p>This stores the target line length: the full length minus any part for a leader at the start of each line.</p> <pre> 8304 \int_new:N \l_iow_target_length_int </pre> <p><i>(End definition for <code>\l_iow_target_length_int</code>. This function is documented on page ??.)</i></p>
<code>\l_iow_current_line_int</code> <code>\l_iow_current_word_int</code> <code>\l_iow_current_indentation_int</code>	<p>These store the number of characters in the line and word currently being constructed, and the current indentation, respectively.</p> <pre> 8305 \int_new:N \l_iow_current_line_int 8306 \int_new:N \l_iow_current_word_int 8307 \int_new:N \l_iow_current_indentation_int </pre> <p><i>(End definition for <code>\l_iow_current_line_int</code>, <code>\l_iow_current_word_int</code>, and <code>\l_iow_current_indentation_int</code>. These functions are documented on page ??.)</i></p>
<code>\l_iow_current_line_tl</code> <code>\l_iow_current_word_tl</code> <code>\l_iow_current_indentation_tl</code>	<p>These hold the current line of text and current word, and a number of spaces for indentation, respectively.</p> <pre> 8308 \tl_new:N \l_iow_current_line_tl 8309 \tl_new:N \l_iow_current_word_tl 8310 \tl_new:N \l_iow_current_indentation_tl </pre> <p><i>(End definition for <code>\l_iow_current_line_tl</code>, <code>\l_iow_current_word_tl</code>, and <code>\l_iow_current_indentation_tl</code>. These functions are documented on page ??.)</i></p>
<code>\l_iow_wrap_tl</code>	<p>Used for the expansion step before detokenizing.</p> <pre> 8311 \tl_new:N \l_iow_wrap_tl </pre> <p><i>(End definition for <code>\l_iow_wrap_tl</code>. This function is documented on page ??.)</i></p>
<code>\l_iow_wrapped_tl</code>	<p>The output from wrapping text: fully expanded and with lines which are not overly long.</p> <pre> 8312 \tl_new:N \l_iow_wrapped_tl </pre> <p><i>(End definition for <code>\l_iow_wrapped_tl</code>. This function is documented on page ??.)</i></p>
<code>\l_iow_line_start_bool</code>	<p>Boolean to avoid adding a space at the beginning of forced newlines.</p> <pre> 8313 \bool_new:N \l_iow_line_start_bool </pre> <p><i>(End definition for <code>\l_iow_line_start_bool</code>. This function is documented on page ??.)</i></p>

`\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.

```
8314 \group_begin:
8315   \char_set_catcode_other:N \*
8316   \char_set_lccode:nn {'\*} {'\ }
8317   \tl_to_lowercase:n { \tl_const:Nn \c_catcode_other_space_tl { * } }
8318 \group_end:
      (End definition for \c_catcode_other_space_tl. This function is documented on page 141.)
```

`\c_iow_wrap_marker_tl` Every special action of the wrapping code is preceeded by the same recognizable string, `\c_iow_wrap_end_marker_tl` `\c_iow_wrap_marker_tl`. Upon seeing that “word”, the wrapping code reads one space-delimited argument to know what operation to perform. The setting of `\escapechar` here is not very important, but makes `\c_iow_wrap_marker_tl` look nicer. Note that `\iow_wrap_new_marker:n` does not survive the group, but all constants are defined globally.

```
8319 \group_begin:
8320   \int_set_eq:NN \tex_escapechar:D \c_minus_one
8321   \tl_const:Nx \c_iow_wrap_marker_tl
8322     { \tl_to_str:n { \^^I \^^O \^^W \^^_ \^^W \^^R \^^A \^^P } }
8323   \cs_set:Npn \iow_wrap_new_marker:n #1
8324     {
8325       \tl_const:cx { c_iow_wrap_ #1 _marker_tl }
8326       {
8327         \c_catcode_other_space_tl
8328         \c_iow_wrap_marker_tl
8329         \c_catcode_other_space_tl
8330         #1
8331         \c_catcode_other_space_tl
8332       }
8333     }
8334   \iow_wrap_new_marker:n { end }
8335   \iow_wrap_new_marker:n { newline }
8336   \iow_wrap_new_marker:n { indent }
8337   \iow_wrap_new_marker:n { unindent }
8338 \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.

```
8339 \cs_new_protected:Npn \iow_indent:n #1 { }
8340 \cs_new:Npx \iow_indent_expandable:n #1
8341   {
8342     \c_iow_wrap_indent_marker_tl
8343     #1
8344     \c_iow_wrap_unindent_marker_tl
8345   }
      (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.

```

8346 \cs_new_protected:Npn \iow_wrap:xnnnN #1#2#3#4#5
8347 {
8348   \group_begin:
8349     \int_set:Nn \l_iow_target_length_int { \l_iow_line_length_int - ( #3 ) }
8350     \int_zero:N \l_iow_current_indentation_int
8351     \tl_clear:N \l_iow_current_indentation_tl
8352     \int_zero:N \l_iow_current_line_int
8353     \tl_clear:N \l_iow_current_line_tl
8354     \tl_clear:N \l_iow_wrap_tl
8355     \bool_set_true:N \l_iow_line_start_bool
8356     \int_set_eq:NN \tex_escapechar:D \c_minus_one
8357     \cs_set_nopar:Npx \{ { \token_to_str:N \{ }
8358     \cs_set_nopar:Npx \# { \token_to_str:N \# }
8359     \cs_set_nopar:Npx \} { \token_to_str:N \} }
8360     \cs_set_nopar:Npx \% { \token_to_str:N \% }
8361     \cs_set_nopar:Npx \~ { \token_to_str:N \~ }
8362     \int_set:Nn \tex_escapechar:D { 92 }
8363     \cs_set_eq:NN \\ \c_iow_wrap_newline_marker_tl
8364     \cs_set_eq:NN \_ \c_catcode_other_space_tl
8365     \cs_set_eq:NN \iow_indent:n \iow_indent_expandable:n
8366     #4
8367   <*initex>
8368     \tl_set:Nx \l_iow_wrap_tl {#1}
8369   </initex>
8370   <*package>
8371     \protected@edef \l_iow_wrap_tl {#1}
8372   </package>
8373     \cs_set:Npn \\ { \iow_newline: #2 }
8374     \use:x
8375     {
8376       \iow_wrap_loop:w
8377       \tl_to_str:N \l_iow_wrap_tl
8378       \tl_to_str:N \c_iow_wrap_end_marker_tl
8379       \c_space_tl \c_space_tl
8380       \exp_not:N \q_stop
8381     }
8382     \exp_args:NNo \group_end:
8383     #5 \l_iow_wrapped_tl
8384   }

```

(End definition for `\iow_wrap:xnnnN`. This function is documented on page 141.)

`\iow_wrap_loop:w` The loop grabs one word in the input, and checks whether it is the special marker, or a normal word.

```

8385 \cs_new_protected:Npn \iow_wrap_loop:w #1 ~ %
8386 {
8387   \tl_set:Nn \l_iow_current_word_tl {#1}
8388   \tl_if_eq:NNTF \l_iow_current_word_tl \c_iow_wrap_marker_tl
8389     { \iow_wrap_special:w }
8390     { \iow_wrap_word: }
8391 }

```

(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.

```

8392 \cs_new_protected_nopar:Npn \iow_wrap_word:
8393 {
8394   \int_set:Nn \l_iow_current_word_int
8395     { \str_length_skip_spaces:N \l_iow_current_word_tl }
8396   \int_add:Nn \l_iow_current_line_int { \l_iow_current_word_int }
8397   \int_compare:nNnTF \l_iow_current_line_int < \l_iow_target_length_int
8398     { \iow_wrap_word_fits: }
8399     { \iow_wrap_word_newline: }
8400   \iow_wrap_loop:w
8401 }
8402 \cs_new_protected_nopar:Npn \iow_wrap_word_fits:
8403 {
8404   \bool_if:NNTF \l_iow_line_start_bool
8405     {
8406       \bool_set_false:N \l_iow_line_start_bool
8407       \tl_put_right:Nx \l_iow_current_line_tl
8408         { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8409       \int_add:Nn \l_iow_current_line_int
8410         { \l_iow_current_indentation_int }
8411     }
8412     {
8413       \tl_put_right:Nx \l_iow_current_line_tl
8414         { ~ \l_iow_current_word_tl }
8415       \int_incr:N \l_iow_current_line_int
8416     }
8417 }
8418 \cs_new_protected_nopar:Npn \iow_wrap_word_newline:
8419 {
8420   \tl_put_right:Nx \l_iow_wrapped_tl
8421     { \l_iow_current_line_tl \\ }
8422   \int_set:Nn \l_iow_current_line_int
8423     {
8424     \l_iow_current_word_int

```

```

8425         + \l_iow_current_indentation_int
8426     }
8427     \tl_set:Nx \l_iow_current_line_tl
8428     { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8429 }

```

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

`\iow_wrap_special:w` When the “special” marker is encountered, read what operation to perform, as a space-delimited argument, perform it, and remember to loop. In fact, to avoid spurious spaces when two special actions follow each other, we look ahead for another copy of the marker.

`\iow_wrap_newline:w` Forced newlines are almost identical to those caused by overflow, except that here the word is empty. To indent more, add four spaces to the start of the indentation token list.

`\iow_wrap_indent:w` To reduce indentation, rebuild the indentation token list using `\prg_replicate:nn`. At the end, we simply save the last line (without the run-on text), and prevent the loop.

`\iow_wrap_unindent:w`

`\iow_wrap_end:w`

```

8430 \cs_new_protected_nopar:Npn \iow_wrap_special:w #1 ~ #2 ~ #3 ~ %
8431 {
8432     \cs_if_exist:cTF { iow_wrap_#1: }
8433     { \use:c { iow_wrap_#1: } }
8434     { \msg_expandable_error:n {#1} }
8435     \str_if_eq:xxTF { #2~#3 } { ~ \c_iow_wrap_marker_tl }
8436     { \iow_wrap_special:w }
8437     { \iow_wrap_loop:w #2 ~ #3 ~ }
8438 }
8439 \cs_new_protected_nopar:Npn \iow_wrap_newline:
8440 {
8441     \tl_put_right:Nx \l_iow_wrapped_tl
8442     { \l_iow_current_line_tl \\ }
8443     \int_zero:N \l_iow_current_line_int
8444     \tl_clear:N \l_iow_current_line_tl
8445     \bool_set_true:N \l_iow_line_start_bool
8446 }
8447 \cs_new_protected_nopar:Npx \iow_wrap_indent:
8448 {
8449     \int_add:Nn \l_iow_current_indentation_int \c_four
8450     \tl_put_right:Nx \exp_not:N \l_iow_current_indentation_tl
8451     { \c_space_tl \c_space_tl \c_space_tl \c_space_tl }
8452 }
8453 \cs_new_protected_nopar:Npn \iow_wrap_unindent:
8454 {
8455     \int_sub:Nn \l_iow_current_indentation_int \c_four
8456     \tl_set:Nx \l_iow_current_indentation_tl
8457     { \prg_replicate:nn \l_iow_current_indentation_int { ~ } }
8458 }
8459 \cs_new_protected_nopar:Npn \iow_wrap_end:
8460 {
8461     \tl_put_right:Nx \l_iow_wrapped_tl
8462     { \l_iow_current_line_tl }
8463     \use_none_delimit_by_q_stop:w
8464 }

```

(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.

```

8465 \cs_new_nopar:Npn \str_length_skip_spaces:N
8466   { \exp_args:No \str_length_skip_spaces:n }
8467 \cs_new:Npn \str_length_skip_spaces:n #1
8468   {
8469     \int_value:w \int_eval:w
8470     \exp_after:wN \str_length_loop:NNNNNNNNN \tl_to_str:n {#1}
8471     {X8}{X7}{X6}{X5}{X4}{X3}{X2}{X1}{X0} \q_stop
8472     \int_eval_end:
8473   }
8474 \cs_new:Npn \str_length_loop:NNNNNNNNN #1#2#3#4#5#6#7#8#9
8475   {
8476     \if_catcode:w X #9
8477     \exp_after:wN \use_none_delimit_by_q_stop:w
8478     \else:
8479       9 +
8480     \exp_after:wN \str_length_loop:NNNNNNNNN
8481     \fi:
8482   }

```

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

196.8 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.

```

8483 \prg_new_conditional:Nnn \ior_if_eof:N { p , T , F , TF }
8484   {
8485     \cs_if_exist:NTF #1
8486     {
8487       \if_int_compare:w #1 = \c_sixteen
8488       \prg_return_true:
8489       \else:
8490         \if_eof:w #1
8491         \prg_return_true:
8492         \else:
8493         \prg_return_false:
8494         \fi:
8495       \fi:
8496     }
8497     { \prg_return_true: }
8498   }

```

(End definition for \ior_if_eof_p:N. This function is documented on page 143.)

\ior_to:NN
\ior_gto:NN

And here we read from files.

```

8499 \cs_new_protected_nopar:Npn \ior_to:NN #1#2
8500 { \tex_read:D #1 to #2 }
8501 \cs_new_protected_nopar:Npn \ior_gto:NN #1#2
8502 { \tex_global:D \tex_read:D #1 to #2 }
      (End definition for \ior_to:NN. This function is documented on page 142.)

```

\ior_str_to:NN Reading as strings is also a primitive wrapper.

```

\ior_str_gto:NN 8503 \cs_new_protected_nopar:Npn \ior_str_to:NN #1#2
8504 { \etex_readline:D #1 to #2 }
8505 \cs_new_protected_nopar:Npn \ior_str_gto:NN #1#2
8506 { \tex_global:D \etex_readline:D #1 to #2 }
      (End definition for \ior_str_to:NN. This function is documented on page 142.)

```

196.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

\iow_now_buffer_safe:Nn This is much more easily done using the wrapping system: there is an expansion there,
 \iow_now_buffer_safe:Nx so a bit of a hack is needed.

```

8507 <*deprecated>
8508 \cs_new_protected:Npn \iow_now_buffer_safe:Nn #1#2
8509 { \iow_wrap:xnnnN { \exp_not:n {#2} } { } \c_zero { } \iow_now:Nn #1 }
8510 \cs_new_protected:Npn \iow_now_buffer_safe:Nx #1#2
8511 { \iow_wrap:xnnnN {#2} { } \c_zero { } \iow_now:Nn #1 }
8512 </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: 8513 <*deprecated>
8514 \cs_new_eq:NN \ior_open_streams: \ior_list_streams:
8515 \cs_new_eq:NN \iow_open_streams: \iow_list_streams:
8516 </deprecated>
      (End definition for \ior_open_streams:. This function is documented on page ??.)
8517 </initex | package>

```

197 l3msg implementation

```

8518 <*initex | package>
8519 <*package>
8520 \ProvidesExplPackage
8521 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8522 \package_check_loaded_expl:
8523 </package>

```

\l_msg_tmp_tl A general scratch for the module.

```

8524 \tl_new:N \l_msg_tmp_tl
      (End definition for \l_msg_tmp_tl. This function is documented on page ??.)

```

198 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
\c_msg_more_text_prefix_tl
```

Locations for the text of messages.

```
8525 \tl_const:Nn \c_msg_text_prefix_tl { msg-text~>~ }
8526 \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
\msg_new:nnn
\msg_gset:nnnn
\msg_gset:nnn
\msg_set:nnnn
\msg_set:nnn
```

Setting a message simply means saving the appropriate text into two functions. A sanity check first.

```
8527 \cs_new_protected:Npn \msg_new:nnnn #1#2
8528 {
8529   \cs_if_exist:cT { \c_msg_text_prefix_tl #1 / #2 }
8530   {
8531     \msg_kernel_error:nn { msg } { message-already-defined }
8532     {#1} {#2}
8533   }
8534   \msg_gset:nnnn {#1} {#2}
8535 }
8536 \cs_new_protected:Npn \msg_new:nnn #1#2#3
8537 { \msg_new:nnnn {#1} {#2} {#3} { } }
8538 \cs_new_protected:Npn \msg_set:nnnn #1#2#3#4
8539 {
8540   \cs_set:cpn { \c_msg_text_prefix_tl #1 / #2 }
8541   ##1##2##3##4 {#3}
8542   \cs_set:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8543   ##1##2##3##4 {#4}
8544 }
8545 \cs_new_protected:Npn \msg_set:nnn #1#2#3
8546 { \msg_set:nnnn {#1} {#2} {#3} { } }
8547 \cs_new_protected:Npn \msg_gset:nnnn #1#2#3#4
8548 {
8549   \cs_gset:cpn { \c_msg_text_prefix_tl #1 / #2 }
8550   ##1##2##3##4 {#3}
8551   \cs_gset:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8552   ##1##2##3##4 {#4}
8553 }
8554 \cs_new_protected:Npn \msg_gset:nnn #1#2#3
8555 { \msg_gset:nnnn {#1} {#2} {#3} { } }
      (End definition for \msg_new:nnnn and \msg_new:nnn. These functions are documented on page
      ??.)
```

198.1 Messages: support functions and text

```

\c_msg_coding_error_text_tl Simple pieces of text for messages.
\c_msg_continue_text_tl      8556 \tl_const:Nn \c_msg_coding_error_text_tl
\c_msg_critical_text_tl      8557 {
\c_msg_fatal_text_tl         8558   This~is~a~coding~error.
\c_msg_help_text_tl          8559   \\ \\
\c_msg_no_info_text_tl       8560 }
\c_msg_on_line_text_tl       8561 \tl_const:Nn \c_msg_continue_text_tl
\c_msg_return_text_tl        8562 { Type~<return>~to~continue }
\c_msg_trouble_text_tl       8563 \tl_const:Nn \c_msg_critical_text_tl
                             8564 { Reading~the~current~file~will~stop }
                             8565 \tl_const:Nn \c_msg_fatal_text_tl
                             8566 { This~is~a~fatal~error:~LaTeX~will~abort }
                             8567 \tl_const:Nn \c_msg_help_text_tl
                             8568 { For~immediate~help~type~H~<return> }
                             8569 \tl_const:Nn \c_msg_no_info_text_tl
                             8570 {
                             8571   LaTeX~does~not~know~anything~more~about~this~error,~sorry.
                             8572   \c_msg_return_text_tl
                             8573 }
                             8574 \tl_const:Nn \c_msg_on_line_text_tl { on~line }
                             8575 \tl_const:Nn \c_msg_return_text_tl
                             8576 {
                             8577   \\ \\
                             8578   Try~typing~<return>~to~proceed.
                             8579   \\
                             8580   If~that~doesn't~work,~type~X~<return>~to~quit.
                             8581 }
                             8582 \tl_const:Nn \c_msg_trouble_text_tl
                             8583 {
                             8584   \\ \\
                             8585   More~errors~will~almost~certainly~follow: \\
                             8586   the~LaTeX~run~should~be~aborted.
                             8587 }
      (End definition for \c_msg_coding_error_text_tl and others. These functions are documented
      on page 145.)

\msg_newline: New lines are printed in the same way as for low-level file writing.
\msg_two_newlines: 8588 \cs_new_nopar:Npn \msg_newline: { ^^J }
                   8589 \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: 8590 \cs_new_nopar:Npn \msg_line_number: { \int_use:N \tex_inputlineno:D }
                   8591 \cs_set_nopar:Npn \msg_line_context:
                   8592 {
                   8593   \c_msg_on_line_text_tl
                   8594   \c_space_tl

```

```

8595     \msg_line_number:
8596   }
      (End definition for \msg_line_number:. This function is documented on page ??.)

```

198.2 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.

```

8597 \char_set_catcode_letter:N \.
8598 \tl_new:N
8599   \c_msg_hide_tl.....
8600 \tl_const:Nn \c_msg_hide_tl
8601 { \c_msg_hide_tl..... }
8602 \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.

```

8603 \tl_new:N \l_msg_text_tl
      (End definition for \l_msg_text_tl. This function is documented on page ??.)

```

`\msg_interrupt:xxx` The low-level interruption macro is rather opaque, unfortunately. The idea here is to create a message which hides all of T_EX’s own information by filling the output up with dots. To achieve this, dots have to be letters. The odd `\c_msg_hide_tl<dots>` actually does the hiding: it is the large run of dots in the name that is important here. The meaning of `\` is altered so that the explanation text is a simple run whilst the initial error has line-continuation shown.

```

8604 \cs_new_protected:Npn \msg_interrupt:xxx #1#2#3
8605 {
8606   \group_begin:
8607   \tl_if_empty:nTF {#3}
8608     { \msg_interrupt_no_details:xx {#1} {#2} }
8609     { \msg_interrupt_details:xxx {#1} {#2} {#3} }
8610   \msg_interrupt_aux:
8611   \group_end:
8612 }

8613 % Depending on the availability of more information there is a choice of
8614 % how to set up the further help. The extra help text has to be set
8615 % before the message itself can be issued. Everything is done using
8616 % \texttt{x}-type expansion as the new line markers are different for
8617 % the two type of text and need to be correctly set up.
8618 %   \begin{macrocode}
8619 \cs_new_protected:Npn \msg_interrupt_no_details:xx #1#2
8620 {
8621   \iow_wrap:xnnnN
8622   { \ \c_msg_no_info_text_tl }
8623   { |~ } { 2 } { } \msg_interrupt_more_text:n

```



```

8624 \iow_wrap:xnnnN { #1 \\\ #2 \\\ \c_msg_continue_text_tl }
8625 { ! ~ } { 2 } {} \msg_interrupt_text:n
8626 }
8627 \cs_new_protected:Npn \msg_interrupt_details:xxx #1#2#3
8628 {
8629 \iow_wrap:xnnnN
8630 { \\\ #3 }
8631 { |~ } { 2 } {} \msg_interrupt_more_text:n
8632 \iow_wrap:xnnnN { #1 \\\ #2 \\\ \c_msg_help_text_tl }
8633 { ! ~ } { 2 } {} \msg_interrupt_text:n
8634 }
8635 \cs_new_protected:Npn \msg_interrupt_text:n #1
8636 { \tl_set:Nn \l_msg_text_tl {#1} }
8637 \cs_new_protected:Npn \msg_interrupt_more_text:n #1
8638 {
8639 <*initex>
8640 \tl_set:Nx \l_msg_tmp_tl
8641 </initex>
8642 <*package>
8643 \protected@edef \l_msg_tmp_tl
8644 </package>
8645 {
8646 |,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
8647 #1
8648 \msg_newline:
8649 |.....
8650 }
8651 \tex_errhelp:D \exp_after:wN { \l_msg_tmp_tl }
8652 }

```

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.

```

8653 \group_begin: % {
8654 \char_set_lccode:nn {'\} } {'\ }
8655 \char_set_lccode:nn {'&} {'\!}
8656 \char_set_catcode_active:N \&
8657 \tl_to_lowercase:n
8658 {
8659 \group_end:
8660 \cs_new_protected:Npn \msg_interrupt_aux:
8661 {
8662 \iow_term:x
8663 {
8664 \iow_newline:
8665 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
8666 \iow_newline:
8667 !

```

```

8668     }
8669     \tl_put_right:No \l_msg_text_tl { \c_msg_hide_tl }
8670     \cs_set_protected_nopar:Npx &
8671     { \tex_errmessage:D { \exp_not:o { \l_msg_text_tl } } }
8672     &
8673   }
8674 }

```

(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.

```

8675 \cs_new_protected:Npn \msg_log:x #1
8676 {
8677   \iow_log:x { ..... }
8678   \iow_wrap:xnnnN { . ~ #1 } { . ~ } { 2 } { }
8679   \iow_log:x
8680   \iow_log:x { ..... }
8681 }
8682 \cs_new_protected:Npn \msg_term:x #1
8683 {
8684   \iow_term:x { ***** }
8685   \iow_wrap:xnnnN { * ~ #1 } { * ~ } { 2 } { }
8686   \iow_term:x
8687   \iow_term:x { ***** }
8688 }

```

(End definition for `\msg_log:x`. This function is documented on page 149.)

198.3 Displaying messages

L^AT_EX is handling error messages and so the T_EX ones are disabled.

```

8689 \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` 8690 `\cs_new_nopar:Npn \msg_fatal_text:n #1 { Fatal~#1~error }`
`\msg_error_text:n` 8691 `\cs_new_nopar:Npn \msg_critical_text:n #1 { Critical~#1~error }`
`\msg_warning_text:n` 8692 `\cs_new_nopar:Npn \msg_error_text:n #1 { #1~error }`
`\msg_info_text:n` 8693 `\cs_new_nopar:Npn \msg_warning_text:n #1 { #1~warning }`
8694 `\cs_new_nopar:Npn \msg_info_text:n #1 { #1~info }`

(End definition for `\msg_fatal_text:n` and others. These functions are documented on page 146.)

`\msg_see_documentation_text:n` Contextual footer information.

```

8695 \cs_new_nopar:Npn \msg_see_documentation_text:n #1
8696 { \ \ \ See-the-#1-documentation-for-further-information. }

```

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

`\l_msg_redirect_classes_prop` For filtering messages, a list of all messages and of those which have to be modified is
`\l_msg_redirect_names_prop` required.

```

8697 \prop_new:N \l_msg_redirect_classes_prop
8698 \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.

```

8699 \cs_new_protected_nopar:Npn \msg_class_set:nn #1#2
8700 {
8701   \prop_clear_new:c { l_msg_redirect_ #1 _prop }
8702   \cs_set_protected:cpn { msg_ #1 :nnxxxx } ##1##2##3##4##5##6
8703   { \msg_use:nnnnxxxx {#1} {#2} {##1} {##2} {##3} {##4} {##5} {##6} }
8704   \cs_set_protected:cpx { msg_ #1 :nnxxx } ##1##2##3##4##5
8705   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} {##5} { } }
8706   \cs_set_protected:cpx { msg_ #1 :nnxx } ##1##2##3##4
8707   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} { } { } }
8708   \cs_set_protected:cpx { msg_ #1 :nnx } ##1##2##3
8709   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} { } { } { } }
8710   \cs_set_protected:cpx { msg_ #1 :nn } ##1##2
8711   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} { } { } { } { } }
8712 }

```

(End definition for `\msg_class_set:nn`. This function is documented on page 146.)

`\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 8713 \prg_set_conditional:Npnn \msg_if_more_text:N #1 { p , T , F , TF }
\msg_no_more_text:xxxx 8714 {
8715   \cs_if_eq:NNTF #1 \msg_no_more_text:xxxx
8716   { \prg_return_false: }
8717   { \prg_return_true: }
8718 }
8719 \cs_new:Npn \msg_no_more_text:xxxx #1#2#3#4 { }
8720 \cs_generate_variant:Nn \msg_if_more_text_p:N { c }
8721 \cs_generate_variant:Nn \msg_if_more_text_NT { c }
8722 \cs_generate_variant:Nn \msg_if_more_text_NF { c }
8723 \cs_generate_variant:Nn \msg_if_more_text_NTF { c }

```

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

`\msg_fatal:nnxxxx` For fatal errors, after the error message T_EX bails out.

```

\msg_fatal:nnxxxx 8724 \msg_class_set:nn { fatal }
\msg_fatal:nnxx 8725 {
\msg_fatal:nnx 8726   \msg_interrupt:xxx
\msg_fatal:nn 8727   { \msg_fatal_text:n {#1} : ~ "#2" }
8728   {
8729     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8730     \msg_see_documentation_text:n {#1}
8731   }
8732   { \c_msg_fatal_text_tl }
8733   \tex_end:D
8734 }

```

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

`\msg_critical:nnxxxx` Not quite so bad: just end the current file.

```

\msg_critical:nnxxx 8735 \msg_class_set:nn { critical }
\msg_critical:nnxx 8736 {
\msg_critical:nnx 8737 \msg_interrupt:xxx
\msg_critical:nn 8738 { \msg_critical_text:n {#1} : ~ "#2" }
8739 {
8740 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8741 \msg_see_documentation_text:n {#1}
8742 }
8743 { \c_msg_critical_text_tl }
8744 \tex_endinput:D
8745 }

```

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

`\msg_error:nnxxxx` For an error, the interrupt routine is called, then any recovery code is tried.

```

\msg_error:nnxxx 8746 \msg_class_set:nn { error }
\msg_error:nnxx 8747 {
\msg_error:nnx 8748 \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl #1 / #2 }
\msg_error:nn 8749 {
8750 \msg_interrupt:xxx
8751 { \msg_error_text:n {#1} : ~ "#2" }
8752 {
8753 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8754 \msg_see_documentation_text:n {#1}
8755 }
8756 { \use:c { \c_msg_more_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8757 }
8758 {
8759 \msg_interrupt:xxx
8760 { \msg_error_text:n {#1} : ~ "#2" }
8761 {
8762 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8763 \msg_see_documentation_text:n {#1}
8764 }
8765 { }
8766 }
8767 }

```

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

`\msg_warning:nnxxxx` Warnings are printed to the terminal.

```

\msg_warning:nnxxx 8768 \msg_class_set:nn { warning }
\msg_warning:nnxx 8769 {
\msg_warning:nnx 8770 \msg_term:x
\msg_warning:nn 8771 {
8772 \msg_warning_text:n {#1} : ~ "#2" \\ \\
8773 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8774 }
8775 }

```

(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 8776 \msg_class_set:nn { info }
\msg_info:nnxxx 8777 {
\msg_info:nnx 8778 \msg_log:x
\msg_info:nn 8779 {
8780 \msg_info_text:n {#1} : ~ "#2" \\ \\
8781 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8782 }
8783 }
```

(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 8784 \msg_class_set:nn { log }
\msg_log:nnxxx 8785 {
\msg_log:nnx 8786 \msg_log:x
\msg_log:nn 8787 { \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8788 }
```

(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 8789 \msg_class_set:nn { none } { }
\msg_none:nnxxx
\msg_none:nnx
\msg_none:nn
```

(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 8790 \seq_new:N \l_msg_redirect_classes_seq
\l_msg_current_class_tl 8791 \tl_new:N \l_msg_class_tl
\l_msg_current_module_tl 8792 \tl_new:N \l_msg_current_class_tl
8793 \tl_new:N \l_msg_current_module_tl
```

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

\msg_use:nnnnxxxx The main message-using macro creates two auxiliary functions: one containing the code for the message, and the second a loop function. There is then a hand-off to the system for checking if redirection is needed.

```
\msg_use_loop_check:nn 8794 \cs_new_protected:Npn \msg_use:nnnnxxxx #1#2#3#4#5#6#7#8
\msg_use_code: 8795 {
\msg_use_loop:n 8796 \cs_set_protected_nopar:Npx \msg_use_code:
\msg_use_loop:o 8797 {
8798 \seq_clear:N \exp_not:N \l_msg_redirect_classes_seq
8799 \exp_not:n {#2}
8800 }
8801 \cs_set_protected:Npx \msg_use_loop:n ##1
8802 {
8803 \seq_if_in:NnTF \exp_not:n \l_msg_redirect_classes_seq {#1}
8804 { \msg_kernel_error:nn { msg } { message-loop } {#1} }
8805 }
```

```

8806         \seq_put_right:Nn \exp_not:N \l_msg_redirect_classes_seq {#1}
8807         \exp_not:N \cs_if_exist:cTF { msg_ ##1 :nnxxxx }
8808         {
8809             \exp_not:N \use:c { msg_ ##1 :nnxxxx }
8810             \exp_not:n { {#3} {#4} {#5} {#6} {#7} {#8} }
8811         }
8812         {
8813             \msg_kernel_error:nnx { msg } { message-class-unknown } {##1}
8814         }
8815     }
8816 }
8817 \cs_if_exist:cTF { \c_msg_text_prefix_tl #3 / #4 }
8818 { \msg_use_aux:nnn {#1} {#3} {#4} }
8819 { \msg_kernel_error:nnxx { msg } { message-unknown } {#3} {#4} }
8820 }

```

The first auxiliary macro looks for a match by name: the most restrictive check.

```

8821 \cs_new_protected_nopar:Npn \msg_use_aux:nnn #1#2#3
8822 {
8823     \tl_set:Nn \l_msg_current_class_tl {#1}
8824     \tl_set:Nn \l_msg_current_module_tl {#2}
8825     \prop_if_in:NnTF \l_msg_redirect_names_prop { // #2 / #3 / }
8826     { \msg_use_loop_check:nn { names } { // #2 / #3 / } }
8827     { \msg_use_aux:nn {#1} {#2} }
8828 }

```

The second function checks for general matches by module or for all modules.

```

8829 \cs_new_protected_nopar:Npn \msg_use_aux:nn #1#2
8830 {
8831     \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } {#2}
8832     { \msg_use_loop_check:nn {#1} {#2} }
8833     {
8834         \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } { * }
8835         { \msg_use_loop_check:nn {#1} { * } }
8836         { \msg_use_code: }
8837     }
8838 }

```

When checking whether to loop, the same code is needed in a few places.

```

8839 \cs_new_protected:Npn \msg_use_loop_check:nn #1#2
8840 {
8841     \prop_get:cnN { l_msg_redirect_ #1 _prop } {#2} \l_msg_class_tl
8842     \tl_if_eq:NNTF \l_msg_current_class_tl \l_msg_class_tl
8843     {
8844         { \msg_use_code: }
8845         { \msg_use_loop:o \l_msg_class_tl }
8846     }
8847 }
8848 \cs_new_protected_nopar:Npn \msg_use_code: { }
8849 \cs_new_protected:Npn \msg_use_loop:n #1 { }
8850 \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.

```
8851 \cs_new_protected_nopar:Npn \msg_redirect_class:nn #1#2
8852 { \prop_put:cnn { l_msg_redirect_ #1 _prop } { * } {#2} }
      (End definition for \msg_redirect_class:nn. This function is documented on page 148.)
```

\msg_redirect_module:nnn For when all messages of a class should be altered for a given module.

```
8853 \cs_new_protected_nopar:Npn \msg_redirect_module:nnn #1#2#3
8854 { \prop_put:cnn { l_msg_redirect_ #2 _prop } {#1} {#3} }
      (End definition for \msg_redirect_module:nnn. This function is documented on page 148.)
```

\msg_redirect_name:nnn Named message will always use the given class.

```
8855 \cs_new_protected_nopar:Npn \msg_redirect_name:nnn #1#2#3
8856 { \prop_put:Nnn \l_msg_redirect_names_prop { // #1 / #2 / } {#3} }
      (End definition for \msg_redirect_name:nnn. This function is documented on page 148.)
```

198.4 Kernel-specific functions

\msg_kernel_new:nnnn The kernel needs some messages of its own. These are created using pre-built functions.
 \msg_kernel_new:nnn Two functions are provided: one more general and one which only has the short text part.
 \msg_kernel_set:nnnn
 \msg_kernel_set:nnn

```
8857 \cs_new_protected_nopar:Npn \msg_kernel_new:nnnn #1#2
8858 { \msg_new:nnnn { LaTeX } { #1 / #2 } }
8859 \cs_new_protected_nopar:Npn \msg_kernel_new:nnn #1#2
8860 { \msg_new:nnn { LaTeX } { #1 / #2 } }
8861 \cs_new_protected_nopar:Npn \msg_kernel_set:nnnn #1#2
8862 { \msg_set:nnnn { LaTeX } { #1 / #2 } }
8863 \cs_new_protected_nopar:Npn \msg_kernel_set:nnn #1#2
8864 { \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 8865 \cs_new_protected:Npn \msg_kernel_fatal:nnxxxx #1#2#3#4#5#6
\msg_kernel_fatal:nnxx 8866 {
\msg_kernel_fatal:nnx 8867   \msg_interrupt:xxx
\msg_kernel_fatal:nn 8868   { \msg_fatal_text:n { LaTeX } : ~ "#1 / #2" }
8869   {
8870     \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8871     {#3} {#4} {#5} {#6}
8872     \msg_see_documentation_text:n { LaTeX3 }
8873   }
8874   { \c_msg_fatal_text_tl }
8875   \tex_end:D
8876 }
8877 \cs_new_protected:Npn \msg_kernel_fatal:nnxxx #1#2#3#4#5
8878 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8879 \cs_new_protected:Npn \msg_kernel_fatal:nnxx #1#2#3#4
```

```

8880 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8881 \cs_new_protected:Npn \msg_kernel_fatal:nnx #1#2#3
8882 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} { } { } { } }
8883 \cs_new_protected:Npn \msg_kernel_fatal:nn #1#2
8884 { \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 8885 \cs_new_protected:Npn \msg_kernel_error:nnxxxx #1#2#3#4#5#6
\msg_kernel_error:nnxxx 8886 {
\msg_kernel_error:nnx 8887   \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
\msg_kernel_error:nn 8888   {
8889     \msg_interrupt:xxx
8890     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8891     {
8892       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8893       {#3} {#4} {#5} {#6}
8894       \msg_see_documentation_text:n { LaTeX3 }
8895     }
8896     {
8897       \use:c { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
8898       {#3} {#4} {#5} {#6}
8899     }
8900   }
8901   {
8902     \msg_interrupt:xxx
8903     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8904     {
8905       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8906       {#3} {#4} {#5} {#6}
8907       \msg_see_documentation_text:n { LaTeX3 }
8908     }
8909     { }
8910   }
8911 }
8912 \cs_new_protected:Npn \msg_kernel_error:nnxxx #1#2#3#4#5
8913 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8914 \cs_set_protected:Npn \msg_kernel_error:nnxx #1#2#3#4
8915 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8916 \cs_set_protected:Npn \msg_kernel_error:nnx #1#2#3
8917 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} { } { } { } }
8918 \cs_set_protected:Npn \msg_kernel_error:nn #1#2
8919 { \msg_kernel_error:nnxxxx {#1} {#2} { } { } { } { } }
(End definition for \msg_kernel_error:nnxxxx. This function is documented on page ??.)

```

\msg_kernel_warning:nnxxxx Kernel messages which can be redirected.

```

\msg_kernel_warning:nnxxx 8920 \prop_new:N \l_msg_redirect_kernel_warning_prop
\msg_kernel_warning:nnxxx 8921 \cs_new_protected:Npn \msg_kernel_warning:nnxxxx #1#2#3#4#5#6
\msg_kernel_warning:nnx 8922 {
\msg_kernel_warning:nn
\msg_kernel_info:nnxxxx
\msg_kernel_info:nnxxx
\msg_kernel_info:nnxx
\msg_kernel_info:nnx
\msg_kernel_info:nn

```



```

8923 \msg_use:nnnnxxxx { warning }
8924 {
8925     \msg_term:x
8926     {
8927         \msg_warning_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
8928         \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8929         {#3} {#4} {#5} {#6}
8930     }
8931 }
8932 { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
8933 }
8934 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5
8935 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8936 \cs_new_protected:Npn \msg_kernel_warning:nnxx #1#2#3#4
8937 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8938 \cs_new_protected:Npn \msg_kernel_warning:nnx #1#2#3
8939 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} { } { } { } }
8940 \cs_new_protected:Npn \msg_kernel_warning:nn #1#2
8941 { \msg_kernel_warning:nnxxxx {#1} {#2} { } { } { } { } }
8942 \prop_new:N \l_msg_redirect_kernel_info_prop
8943 \cs_new_protected:Npn \msg_kernel_info:nnxxxx #1#2#3#4#5#6
8944 {
8945     \msg_use:nnnnxxxx { info }
8946     {
8947         \msg_log:x
8948         {
8949             \msg_info_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
8950             \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8951             {#3} {#4} {#5} {#6}
8952         }
8953     }
8954     { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
8955 }
8956 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5
8957 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8958 \cs_new_protected:Npn \msg_kernel_info:nnxx #1#2#3#4
8959 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8960 \cs_new_protected:Npn \msg_kernel_info:nnx #1#2#3
8961 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} { } { } { } }
8962 \cs_new_protected:Npn \msg_kernel_info:nn #1#2
8963 { \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.

```

8964 \msg_kernel_new:nnnn { msg } { message-already-defined }
8965 { Message~'#2'~for~module~'#1'~already-defined. }
8966 {
8967     \c_msg_coding_error_text_tl
8968     LaTeX~was~asked~to~define~a~new~message~called~'#2'
8969     by~the~module~'#1'~module:\\

```

```

8970     this-message~already~exists.
8971     \c_msg_return_text_tl
8972   }
8973   \msg_kernel_new:nnnn { msg } { message-unknown }
8974   { Unknown~message~'#2'~for~module~'#1'. }
8975   {
8976     \c_msg_coding_error_text_tl
8977     LaTeX~was~asked~to~display~a~message~called~'#2'\\
8978     by~the~module~'#1'~module:~this~message~does~not~exist.
8979     \c_msg_return_text_tl
8980   }
8981   \msg_kernel_new:nnnn { msg } { message-class-unknown }
8982   { Unknown~message~class~'#1'. }
8983   {
8984     LaTeX~has~been~asked~to~redirect~messages~to~a~class~'#1':\\
8985     this~was~never~defined.
8986     \c_msg_return_text_tl
8987   }
8988   \msg_kernel_new:nnnn { msg } { redirect-loop }
8989   { Message~redirection~loop~for~message~class~'#1'. }
8990   {
8991     LaTeX~has~been~asked~to~redirect~messages~in~an~infinite~loop.\\
8992     The~original~message~here~has~been~lost.
8993     \c_msg_return_text_tl
8994   }

```

Messages for earlier kernel modules.

```

8995   \msg_kernel_new:nnnn { kernel } { bad-number-of-arguments }
8996   { Function~'#1'~cannot~be~defined~with~#2~arguments. }
8997   {
8998     \c_msg_coding_error_text_tl
8999     LaTeX~has~been~asked~to~define~a~function~'#1'~with~
9000     #2~arguments. \\
9001     TeX~allows~between~0~and~9~arguments~for~a~single~function.
9002   }
9003   \msg_kernel_new:nnnn { kernel } { command-already-defined }
9004   { Control~sequence~#1~already~defined. }
9005   {
9006     \c_msg_coding_error_text_tl
9007     LaTeX~has~been~asked~to~create~a~new~control~sequence~'#1'~
9008     but~this~name~has~already~been~used~elsewhere. \\ \\
9009     The~current~meaning~is:\\
9010     \ \ #2
9011   }
9012   \msg_kernel_new:nnnn { kernel } { command-not-defined }
9013   { Control~sequence~#1~undefined. }
9014   {
9015     \c_msg_coding_error_text_tl
9016     LaTeX~has~been~asked~to~use~a~command~#1,~but~this~has~not~
9017     been~defined~yet.

```

```

9018 }
9019 \msg_kernel_new:nnnn { kernel } { variable-not-defined }
9020 { Variable~#1~undefined. }
9021 {
9022   \c_msg_coding_error_text_tl
9023   LaTeX~has~been~asked~to~show~a~variable~#1,~but~this~has~not~
9024   been~defined~yet.
9025 }
9026 \msg_kernel_new:nnnn { seq } { empty-sequence }
9027 { Empty~sequence~#1. }
9028 {
9029   \c_msg_coding_error_text_tl
9030   LaTeX~has~been~asked~to~recover~an~entry~from~a~sequence~that~
9031   has~no~content:~that~cannot~happen!
9032 }
9033 \msg_kernel_new:nnnn { tl } { empty-search-pattern }
9034 { Empty~search~pattern. }
9035 {
9036   \c_msg_coding_error_text_tl
9037   LaTeX~has~been~asked~to~replace~an~empty~pattern~by~'~#1':~that~%
9038   would~lead~to~an~infinite~loop!
9039 }

```

```

\msg_kernel_bug:x
\c_msg_kernel_bug_text_tl
\c_msg_kernel_bug_more_text_tl

```

The L^AT_EX coding bug error gets re-visited here.

```

9040 \cs_set_protected:Npn \msg_kernel_bug:x #1
9041 {
9042   \msg_interrupt:xxx { \c_msg_kernel_bug_text_tl }
9043   {
9044     #1
9045     \msg_see_documentation_text:n { LaTeXX3 }
9046   }
9047   { \c_msg_kernel_bug_more_text_tl }
9048 }
9049 \tl_const:Nn \c_msg_kernel_bug_text_tl
9050 { This~is~a~LaTeX~bug:~check~coding! }
9051 \tl_const:Nn \c_msg_kernel_bug_more_text_tl
9052 {
9053   There~is~a~coding~bug~somewhere~around~here. \\
9054   This~probably~needs~examining~by~an~expert.
9055   \c_msg_return_text_tl
9056 }

```

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

198.5 Expandable errors

```
\msg_expandable_error:n
```

In expansion only context, we cannot use the normal means of reporting errors. Instead, we feed T_EX 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`.

```

9057 \group_begin:
9058 \char_set_catcode_math_superscript:N \^
9059 \char_set_lccode:nn {'~} {'\ }
9060 \char_set_lccode:nn {'L} {'L}
9061 \char_set_lccode:nn {'T} {'T}
9062 \char_set_lccode:nn {'X} {'X}
9063 \tl_to_lowercase:n
9064 {
9065   \cs_new:Npx \msg_expandable_error:n #1
9066   {
9067     \exp_not:n
9068     {
9069       \tex_romannumeral:D
9070       \exp_after:wN \exp_after:wN
9071       \exp_after:wN \msg_expandable_error_aux:w
9072       \exp_after:wN \exp_after:wN
9073       \exp_after:wN \c_zero
9074     }
9075     \exp_not:N \use:n { \exp_not:c { LaTeX3~error: } ^ #1 }
9076     \exp_not:N \q_stop
9077   }
9078   \cs_new:Npn \msg_expandable_error_aux:w #1 ^ #2 \q_stop { #1 }
9079 }
9080 \group_end:

```

(End definition for `\msg_expandable_error:n`. This function is documented on page 151.)

198.6 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.

```

9081 <*deprecated>
9082 \cs_new_eq:NN \msg_class_new:nn \msg_class_set:nn
9083 </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:nnx
\msg_trace:nn
\msg_trace:nn

```

9084 <*deprecated>

```

9085 \cs_new_eq:NN \msg_trace:nnxxxx \msg_log:nnxxxx
9086 \cs_new_eq:NN \msg_trace:nnxxx \msg_log:nnxxx
9087 \cs_new_eq:NN \msg_trace:nnxx \msg_log:nnxx
9088 \cs_new_eq:NN \msg_trace:nnx \msg_log:nnx
9089 \cs_new_eq:NN \msg_trace:nn \msg_log:nn
9090 </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 9091 <*deprecated>
\msg_generic_set:nnn 9092 \cs_new_protected:Npn \msg_generic_new:nnn #1#2#3 { \deprecated }
\msg_generic_set:nn 9093 \cs_new_protected:Npn \msg_generic_new:nn #1#2 { \deprecated }
\msg_direct_interrupt:xxxxx 9094 \cs_new_protected:Npn \msg_generic_set:nnn #1#2#3 { \deprecated }
\msg_direct_log:xx 9095 \cs_new_protected:Npn \msg_generic_set:nn #1#2 { \deprecated }
\msg_direct_term:xx 9096 \cs_new_protected:Npn \msg_direct_interrupt:xxxxx #1#2#3#4#5 { \deprecated }
9097 \cs_new_protected:Npn \msg_direct_log:xx #1#2 { \deprecated }
9098 \cs_new_protected:Npn \msg_direct_term:xx #1#2 { \deprecated }
9099 </deprecated>
(End definition for \msg_generic_new:nnn. This function is documented on page ??.)
9100 </initex | package>

```

199 l3keys Implementation

```

9101 <*initex | package>
9102 <*package>
9103 \ProvidesExplPackage
9104 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
9105 \package_check_loaded_expl:
9106 </package>

```

199.1 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.
9107 \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 9108 \tl_new:N \l_keyval_key_tl
9109 \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 9110 \tl_new:N \l_keyval_sanitise_tl
9111 \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.

```

9112 \group_begin:
9113   \char_set_catcode_active:n { '\= }
9114   \char_set_catcode_active:n { '\, }
9115   \char_set_lccode:nn { '\8 } { '\= }
9116   \char_set_lccode:nn { '\9 } { '\, }
9117 \tl_to_lowercase:n
9118 {
9119   \group_end:
9120   \cs_new_protected:Npn \keyval_parse:n #1
9121   {
9122     \group_begin:
9123     \tl_clear:N \l_keyval_sanitise_tl
9124     \tl_set:Nn \l_keyval_sanitise_tl {#1}
9125     \tl_replace_all:Nnn \l_keyval_sanitise_tl { = } { 8 }
9126     \tl_replace_all:Nnn \l_keyval_sanitise_tl { , } { 9 }
9127     \tl_clear:N \l_keyval_parse_tl
9128     \exp_after:wN \keyval_parse_elt:w \exp_after:wN
9129       \q_no_value \l_keyval_sanitise_tl 9 \q_nil 9
9130     \exp_after:wN \group_end:
9131     \l_keyval_parse_tl
9132   }
9133 }

```

(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.

```

9134 \cs_new_protected:Npn \keyval_parse_elt:w #1 ,
9135 {
9136   \tl_if_blank:oTF { \use_none:n #1 }
9137   { \keyval_parse_elt:w \q_no_value }
9138   {
9139     \quark_if_nil:oF { \use_ii:nn #1 }
9140     {
9141       \keyval_split_key_value:w #1 = = \q_stop
9142       \keyval_parse_elt:w \q_no_value
9143     }
9144   }
9145 }

```

(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.

```

9146 \cs_new_protected:Npn \keyval_split_key_value:w #1 = #2 \q_stop
9147 {
9148   \keyval_split_key:w #1 \q_stop
9149   \str_if_eq:nnTF {#2} { = }
9150   {
9151     \tl_put_right:Nx \l_keyval_parse_tl
9152     {
9153       \exp_not:c
9154       { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n }
9155       { \exp_not:o \l_keyval_key_tl }
9156     }
9157   }
9158   {
9159     \keyval_split_key_value_aux:wTF #2 \q_no_value \q_stop
9160     { \keyval_split_value:w \q_nil #2 }
9161     { \msg_kernel_error:nn { keyval } { misplaced-equals-sign } }
9162   }
9163 }
9164 \cs_new:Npn \keyval_split_key_value_aux:wTF #1 = #2#3 \q_stop
9165 { \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`.

```

9166 \cs_new_protected:Npn \keyval_split_key:w #1 \q_stop
9167 {
9168   \tl_set:Nx \l_keyval_key_tl
9169   { \exp_after:wN \tl_trim_spaces:n \exp_after:wN { \use_none:n #1 } }
9170 }

```

(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.

```

9171 \cs_new_protected:Npn \keyval_split_value:w #1 = =
9172 {
9173   \tl_put_right:Nx \l_keyval_parse_tl
9174   {
9175     \exp_not:c
9176     { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn }
9177     { \exp_not:o \l_keyval_key_tl }
9178   }

```

```

9179 \tl_set:Nx \l_keyval_value_tl
9180 { \exp_not:o { \use_none:nnn #1 \q_nil \q_nil } }
9181 \tl_if_empty:NTF \l_keyval_value_tl
9182 { \tl_put_right:Nn \l_keyval_parse_tl { { } } }
9183 {
9184   \quark_if_nil:NTF \l_keyval_value_tl
9185   {
9186     \tl_put_right:Nx \l_keyval_parse_tl
9187     { { \exp_not:o { \use_ii:nnn #1 \q_nil } } }
9188   }
9189   { \keyval_split_value_aux:w #1 \q_stop }
9190 }
9191 }

```

A similar idea to the key code: remove the spaces from each end and deal with one set of braces.

```

9192 \cs_new_protected:Npn \keyval_split_value_aux:w \q_nil #1 \q_stop
9193 {
9194   \tl_set:Nx \l_keyval_value_tl { \tl_trim_spaces:n {#1} }
9195   \tl_put_right:Nx \l_keyval_parse_tl
9196   { { \exp_not:o \l_keyval_value_tl } }
9197 }

```

(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.

```

9198 \cs_new_protected:Npn \keyval_parse:NNn #1#2#3
9199 {
9200   \int_gincr:N \g_keyval_level_int
9201   \cs_gset_eq:cN
9202   { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n } #1
9203   \cs_gset_eq:cN
9204   { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn } #2
9205   \keyval_parse:n {#3}
9206   \int_gdecr:N \g_keyval_level_int
9207 }

```

(End definition for \keyval_parse:NNn. This function is documented on page 162.)

One message for the low level parsing system.

```

9208 \msg_kernel_new:nnnn { keyval } { misplaced-equals-sign }
9209 { Misplaced~equals~sign~in~key~value~input~\msg_line_number: }
9210 {
9211   LaTeX~is~attempting~to~parse~some~key~value~input~but~found~
9212   two~equals~signs~not~separated~by~a~comma.
9213 }

```

199.2 Constants and variables

`\c_keys_code_root_tl` The prefixes for the code and variables of the keys themselves.

```

\c_keys_vars_root_tl 9214 \tl_const:Nn \c_keys_code_root_tl { key~code~>~ }
9215 \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 ??.)

<code>\c_keys_props_root_tl</code>	<p>The prefix for storing properties.</p> <pre>9216 \tl_const:Nn \c_keys_props_root_tl { key~prop~>~ }</pre> <p>(End definition for <code>\c_keys_props_root_tl</code>. This function is documented on page ??.)</p>
<code>\c_keys_value_forbidden_tl</code> <code>\c_keys_value_required_tl</code>	<p>Two marker token lists.</p> <pre>9217 \tl_const:Nn \c_keys_value_forbidden_tl { forbidden } 9218 \tl_const:Nn \c_keys_value_required_tl { required }</pre> <p>(End definition for <code>\c_keys_value_forbidden_tl</code> and <code>\c_keys_value_required_tl</code>. These functions are documented on page ??.)</p>
<code>\l_keys_choice_int</code> <code>\l_keys_choices_tl</code>	<p>Publicly accessible data on which choice is being used when several are generated as a set.</p> <pre>9219 \int_new:N \l_keys_choice_int 9220 \tl_new:N \l_keys_choices_tl</pre> <p>(End definition for <code>\l_keys_choice_int</code> and <code>\l_keys_choices_tl</code>. These functions are documented on page ??.)</p>
<code>\l_keys_key_tl</code>	<p>The name of a key itself: needed when setting keys.</p> <pre>9221 \tl_new:N \l_keys_key_tl</pre> <p>(End definition for <code>\l_keys_key_tl</code>. This function is documented on page 160.)</p>
<code>\l_keys_module_tl</code>	<p>The module for an entire set of keys.</p> <pre>9222 \tl_new:N \l_keys_module_tl</pre> <p>(End definition for <code>\l_keys_module_tl</code>. This function is documented on page ??.)</p>
<code>\l_keys_no_value_bool</code>	<p>A marker is needed internally to show if only a key or a key plus a value was seen: this is recorded here.</p> <pre>9223 \bool_new:N \l_keys_no_value_bool</pre> <p>(End definition for <code>\l_keys_no_value_bool</code>. This function is documented on page ??.)</p>
<code>\l_keys_path_tl</code>	<p>The “path” of the current key is stored here: this is available to the programmer and so is public.</p> <pre>9224 \tl_new:N \l_keys_path_tl</pre> <p>(End definition for <code>\l_keys_path_tl</code>. This function is documented on page 160.)</p>
<code>\l_keys_property_tl</code>	<p>The “property” begin set for a key at definition time is stored here.</p> <pre>9225 \tl_new:N \l_keys_property_tl</pre> <p>(End definition for <code>\l_keys_property_tl</code>. This function is documented on page ??.)</p>
<code>\l_keys_unknown_clist</code>	<p>Used when setting only known keys to store those left over.</p> <pre>9226 \tl_new:N \l_keys_unknown_clist</pre> <p>(End definition for <code>\l_keys_unknown_clist</code>. This function is documented on page ??.)</p>
<code>\l_keys_value_tl</code>	<p>The value given for a key: may be empty if no value was given.</p> <pre>9227 \tl_new:N \l_keys_value_tl</pre> <p>(End definition for <code>\l_keys_value_tl</code>. This function is documented on page 160.)</p>

199.3 The key defining mechanism

`\keys_define:nn` The public function for definitions is just a wrapper for the lower level mechanism, more or less. The outer function is designed to keep a track of the current module, to allow safe nesting. The module is set removing any leading / (which is not needed here).

```

9228 \cs_new_protected:Npn \keys_define:nn
9229 { \keys_define_aux:onn \l_keys_module_tl }
9230 \cs_new_protected:Npn \keys_define_aux:nnn #1#2#3
9231 {
9232   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9233   \keyval_parse:NNn \keys_define_elt:n \keys_define_elt:nn {#3}
9234   \tl_set:Nn \l_keys_module_tl {#1}
9235 }
9236 \cs_generate_variant:Nn \keys_define_aux:nnn { o }

```

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

`\keys_define_elt:n` The outer functions here record whether a value was given and then converge on a common internal mechanism. There is first a search for a property in the current key name, then a check to make sure it is known before the code hands off to the next step.

```

9237 \cs_new_protected_nopar:Npn \keys_define_elt:n #1
9238 {
9239   \bool_set_true:N \l_keys_no_value_bool
9240   \keys_define_elt_aux:nn {#1} { }
9241 }
9242 \cs_new_protected:Npn \keys_define_elt:nn #1#2
9243 {
9244   \bool_set_false:N \l_keys_no_value_bool
9245   \keys_define_elt_aux:nn {#1} {#2}
9246 }
9247 \cs_new_protected:Npn \keys_define_elt_aux:nn #1#2 {
9248   \keys_property_find:n {#1}
9249   \cs_if_exist:cTF { \c_keys_props_root_tl \l_keys_property_tl }
9250   { \keys_define_key:n {#2} }
9251   {
9252     \msg_kernel_error:nxxx { keys } { property-unknown }
9253     { \l_keys_property_tl } { \l_keys_path_tl }
9254   }
9255 }

```

(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 and after it. Everything is turned into strings, so there is no problem using an x-type expansion.

```

9256 \cs_new_protected_nopar:Npn \keys_property_find:n #1
9257 {
9258   \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / }
9259   \tl_if_in:nnTF {#1} { . }
9260   { \keys_property_find_aux:w #1 \q_stop }
9261   { \msg_kernel_error:nnx { keys } { key-no-property } {#1} }

```

```

9262 }
9263 \cs_new_protected_nopar:Npn \keys_property_find_aux:w #1 . #2 \q_stop
9264 {
9265   \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl \tl_to_str:n {#1} }
9266   \tl_if_in:nnTF {#2} { . }
9267   {
9268     \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl . }
9269     \keys_property_find_aux:w #2 \q_stop
9270   }
9271   { \tl_set:Nn \l_keys_property_tl { . #2 } }
9272 }

```

(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, 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.

\keys_define_key_aux:w

```

9273 \cs_new_protected:Npn \keys_define_key:n #1
9274 {
9275   \bool_if:NTF \l_keys_no_value_bool
9276   {
9277     \exp_after:wN \keys_define_key_aux:w
9278     \l_keys_property_tl \q_stop
9279     { \use:c { \c_keys_props_root_tl \l_keys_property_tl } }
9280     {
9281       \msg_kernel_error:nnxx { keys }
9282       { property-requires-value } { \l_keys_property_tl }
9283       { \l_keys_path_tl }
9284     }
9285   }
9286   { \use:c { \c_keys_props_root_tl \l_keys_property_tl } {#1} }
9287 }
9288 \cs_new_protected:Npn \keys_define_key_aux:w #1 : #2 \q_stop
9289 { \tl_if_empty:nTF {#2} }

```

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

199.4 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.

```

9290 \cs_new_nopar:Npn \keys_bool_set:NN #1#2
9291 {
9292   \cs_if_exist:NF #1 { \bool_new:N #1 }
9293   \keys_choice_make:
9294   \keys_cmd_set:nx { \l_keys_path_tl / true }
9295   { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9296   \keys_cmd_set:nx { \l_keys_path_tl / false }
9297   { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }

```

```

9298 \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9299 {
9300     \msg_kernel_error:nxx { keys } { boolean-values-only }
9301     { \l_keys_key_tl }
9302 }
9303 \keys_default_set:n { true }
9304 }

```

(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.

```

9305 \cs_new_nopar:Npn \keys_bool_set_inverse:NN #1#2
9306 {
9307     \cs_if_exist:NF #1 { \bool_new:N #1 }
9308     \keys_choice_make:
9309     \keys_cmd_set:nx { \l_keys_path_tl / true }
9310     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9311     \keys_cmd_set:nx { \l_keys_path_tl / false }
9312     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9313     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9314     {
9315         \msg_kernel_error:nxx { keys } { boolean-values-only }
9316         { \l_keys_key_tl }
9317     }
9318     \keys_default_set:n { true }
9319 }

```

(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.

```

9320 \cs_new_protected_nopar:Npn \keys_choice_make:
9321 {
9322     \keys_cmd_set:nn { \l_keys_path_tl }
9323     { \keys_choice_find:n {##1} }
9324     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9325     {
9326         \msg_kernel_error:nxxx { keys } { choice-unknown }
9327         { \l_keys_path_tl } {##1}
9328     }
9329 }

```

(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.

```

9330 \cs_new_protected:Npn \keys_choices_make:nn #1#2
9331 {
9332     \keys_choice_make:
9333     \int_zero:N \l_keys_choice_int
9334     \clist_map_inline:nn {#1}
9335     {
9336         \keys_cmd_set:nx { \l_keys_path_tl / ##1 }

```

```

9337         {
9338             \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9339             \int_set:Nn \exp_not:N \l_keys_choice_int
9340             { \int_use:N \l_keys_choice_int }
9341             \exp_not:n {#2}
9342         }
9343     \int_incr:N \l_keys_choice_int
9344 }
9345 }

```

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

`\keys_choices_generate:n` Creating multiple-choices means setting up the “indicator” code, then applying whatever the user wanted.

`\keys_choices_generate_aux:n`

```

9346 \cs_new_protected:Npn \keys_choices_generate:n #1
9347 {
9348     \cs_if_exist:cTF
9349     { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9350     {
9351         \keys_choice_make:
9352         \int_zero:N \l_keys_choice_int
9353         \clist_map_function:nN {#1} \keys_choices_generate_aux:n
9354     }
9355     {
9356         \msg_kernel_error:nnx { keys }
9357         { generate-choices-before-code } { \l_keys_path_tl }
9358     }
9359 }
9360 \cs_new_protected_nopar:Npn \keys_choices_generate_aux:n #1
9361 {
9362     \keys_cmd_set:nx { \l_keys_path_tl / #1 }
9363     {
9364         \tl_set:Nn \exp_not:N \l_keys_choice_tl {#1}
9365         \int_set:Nn \exp_not:N \l_keys_choice_int
9366         { \int_use:N \l_keys_choice_int }
9367         \exp_not:v
9368         { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9369     }
9370     \int_incr:N \l_keys_choice_int
9371 }

```

(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.

```

9372 \cs_new_protected:Npn \keys_choice_code_store:x #1
9373 {
9374     \cs_if_exist:cF
9375     { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9376     {
9377         \tl_new:c
9378         { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }

```

```

9379     }
9380     \tl_set:cx { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9381     {#1}
9382 }

```

(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 9383 \cs_new_protected:Npn \keys_cmd_set:nn #1#2
9384 {
9385   \keys_cmd_set_aux:n {#1}
9386   \cs_set:cpn { \c_keys_code_root_tl #1 } ##1 {#2}
9387 }
9388 \cs_new_protected:Npn \keys_cmd_set:nx #1#2
9389 {
9390   \keys_cmd_set_aux:n {#1}
9391   \cs_set:cpx { \c_keys_code_root_tl #1 } ##1 {#2}
9392 }
9393 \cs_new_protected_nopar:Npn \keys_cmd_set_aux:n #1
9394 {
9395   \tl_clear_new:c { \c_keys_vars_root_tl #1 .default }
9396   \tl_set:cn { \c_keys_vars_root_tl #1 .default } { \q_no_value }
9397   \tl_clear_new:c { \c_keys_vars_root_tl #1 .req }
9398 }

```

(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 9399 \cs_new_protected:Npn \keys_default_set:n #1
9400 { \tl_set:cn { \c_keys_vars_root_tl \l_keys_path_tl .default } {#1} }
9401 \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 9402 \cs_new_protected_nopar:Npn \keys_meta_make:n #1
9403 {
9404   \exp_args:NNo \keys_cmd_set:nn \l_keys_path_tl
9405   { \exp_after:wN \keys_set:nn \exp_after:wN { \l_keys_module_tl } {#1} }
9406 }
9407 \cs_new_protected_nopar:Npn \keys_meta_make:x #1
9408 {
9409   \keys_cmd_set:nx { \l_keys_path_tl }
9410   { \exp_not:N \keys_set:nn { \l_keys_module_tl } {#1} }
9411 }

```

(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.

```

9412 \cs_new_nopar:Npn \keys_multichoice_find:n #1
9413 { \clist_map_function:nN {#1} \keys_choice_find:n }
9414 \cs_new_protected_nopar:Npn \keys_multichoice_make:
9415 {
9416   \keys_cmd_set:nn { \l_keys_path_tl }
9417   { \keys_multichoice_find:n {##1} }
9418   \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9419   {
9420     \msg_kernel_error:nnxx { keys } { choice-unknown }
9421     { \l_keys_path_tl } {##1}
9422   }
9423 }
9424 \cs_new_protected:Npn \keys_multichoices_make:nn #1#2
9425 {
9426   \keys_multichoice_make:
9427   \int_zero:N \l_keys_choice_int
9428   \clist_map_inline:nn {#1}
9429   {
9430     \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9431     {
9432       \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9433       \int_set:Nn \exp_not:N \l_keys_choice_int
9434       { \int_use:N \l_keys_choice_int }
9435       \exp_not:n {#2}
9436     }
9437     \int_incr:N \l_keys_choice_int
9438   }
9439 }

```

(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.

```

9440 \cs_new_protected_nopar:Npn \keys_value_requirement:n #1
9441 {
9442   \tl_set_eq:cc
9443   { \c_keys_vars_root_tl \l_keys_path_tl .req }
9444   { c_keys_value_ #1 _tl }
9445 }

```

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

`\keys_variable_set:NnNN` Setting a variable takes the type and scope separately so that it is easy to make a new
`\keys_variable_set:cnNN` variable if needed. The three-argument version is set up so that the use of { } as an
`\keys_variable_set:NnN` N-type variable is only done once!
`\keys_variable_set:cnN`

```

9446 \cs_new_protected_nopar:Npn \keys_variable_set:NnNN #1#2#3#4
9447 {
9448   \cs_if_exist:NF #1 { \use:c { #2 _new:N } #1 }
9449   \keys_cmd_set:nx { \l_keys_path_tl }

```

```

9450     { \exp_not:c { #2 _ #3 set:N #4 } \exp_not:N #1 {##1} }
9451   }
9452   \cs_new_protected_nopar:Npn \keys_variable_set:NnN #1#2#3
9453     { \keys_variable_set:NnN #1 {#2} { } #3 }
9454   \cs_generate_variant:Nn \keys_variable_set:NnN { c }
9455   \cs_generate_variant:Nn \keys_variable_set:NnN { c }

```

(End definition for \keys_variable_set:NnN and \keys_variable_set:cnN. These functions are documented on page ??.)

199.5 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.

```

9456 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_set:N } #1
9457   { \keys_bool_set:NN #1 { } }
9458 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_gset:N } #1
9459   { \keys_bool_set:NN #1 g }

```

(End definition for .bool_set:N. This function is documented on page 154.)

`.bool_set_inverse:N` One function for this.

```

9460 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_set_inverse:N } #1
9461   { \keys_bool_set_inverse:NN #1 { } }
9462 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_gset_inverse:N } #1
9463   { \keys_bool_set_inverse:NN #1 g }

```

(End definition for .bool_set_inverse:N. This function is documented on page 154.)

`.choice:` Making a choice is handled internally, as it is also needed by `.generate_choices:n`.

```

9464 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .choice: }
9465   { \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.

```

9466 \cs_new_protected:cpn { \c_keys_props_root_tl .choices:nn } #1
9467   { \keys_choices_make:nn #1 }

```

(End definition for .choices:nn. This function is documented on page 154.)

`.code:n` Creating code is simply a case of passing through to the underlying `set` function.

```

9468 \cs_new_protected:cpn { \c_keys_props_root_tl .code:n } #1
9469   { \keys_cmd_set:nn { \l_keys_path_tl } {#1} }
9470 \cs_new_protected:cpn { \c_keys_props_root_tl .code:x } #1
9471   { \keys_cmd_set:nx { \l_keys_path_tl } {#1} }

```

(End definition for .code:n and .code:x. These functions are documented on page 155.)

`.choice_code:n` Storing the code for choices, using `\exp_not:n` to avoid needing two internal functions.

```
.choice_code:x 9472 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:n } #1
9473 { \keys_choice_code_store:x { \exp_not:n {#1} } }
9474 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:x } #1
9475 { \keys_choice_code_store:x {#1} }
```

(End definition for .choice_code:n and .choice_code:x. These functions are documented on page 154.)

```
.clist_set:N
.clist_set:c 9476 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_set:N } #1
.clist_gset:N 9477 { \keys_variable_set:NnN #1 { clist } n }
.clist_gset:c 9478 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_set:c } #1
9479 { \keys_variable_set:cnN {#1} { clist } n }
9480 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_gset:N } #1
9481 { \keys_variable_set:NnNN #1 { clist } g n }
9482 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_gset:c } #1
9483 { \keys_variable_set:cnNN {#1} { clist } g n }
```

(End definition for .clist_set:N and .clist_set:c. These functions are documented on page 154.)

`.default:n` Expansion is left to the internal functions.

```
.default:V 9484 \cs_new_protected:cpn { \c_keys_props_root_tl .default:n } #1
9485 { \keys_default_set:n {#1} }
9486 \cs_new_protected:cpn { \c_keys_props_root_tl .default:V } #1
9487 { \keys_default_set:V #1 }
```

(End definition for .default:n and .default:V. These functions are documented on page 155.)

`.dim_set:N` Setting a variable is very easy: just pass the data along.

```
.dim_set:c 9488 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_set:N } #1
9489 { \keys_variable_set:NnN #1 { dim } n }
9490 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_set:c } #1
9491 { \keys_variable_set:cnN {#1} { dim } n }
9492 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_gset:N } #1
9493 { \keys_variable_set:NnNN #1 { dim } g n }
9494 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_gset:c } #1
9495 { \keys_variable_set:cnNN {#1} { dim } g n }
```

(End definition for .dim_set:N and .dim_set:c. These functions are documented on page 155.)

`.fp_set:N` Setting a variable is very easy: just pass the data along.

```
.fp_set:c 9496 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_set:N } #1
9497 { \keys_variable_set:NnN #1 { fp } n }
9498 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_set:c } #1
9499 { \keys_variable_set:cnN {#1} { fp } n }
9500 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_gset:N } #1
9501 { \keys_variable_set:NnNN #1 { fp } g n }
9502 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_gset:c } #1
9503 { \keys_variable_set:cnNN {#1} { fp } g n }
```

(End definition for .fp_set:N and .fp_set:c. These functions are documented on page 155.)

`.generate_choices:n` Making choices is easy.

```
9504 \cs_new_protected:cpn { \c_keys_props_root_tl .generate_choices:n } #1
9505 { \keys_choices_generate:n {#1} }
(End definition for .generate_choices:n. This function is documented on page 156.)
```

`.int_set:N` Setting a variable is very easy: just pass the data along.

```
.int_set:c 9506 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_set:N } #1
.int_gset:N 9507 { \keys_variable_set:NnN #1 { int } n }
.int_gset:c 9508 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_set:c } #1
9509 { \keys_variable_set:cnN {#1} { int } n }
9510 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_gset:N } #1
9511 { \keys_variable_set:NnNN #1 { int } g n }
9512 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_gset:c } #1
9513 { \keys_variable_set:cnNN {#1} { int } g n }
(End definition for .int_set:N and .int_set:c. These functions are documented on page 156.)
```

`.meta:n` Making a meta is handled internally.

```
.meta:x 9514 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:n } #1
9515 { \keys_meta_make:n {#1} }
9516 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:x } #1
9517 { \keys_meta_make:x {#1} }
(End definition for .meta:n and .meta:x. These functions are documented on page 156.)
```

`.multichoice:` The same idea as `.choice:` and `.choices:nn`, but where more than one choice is allowed.

```
.multichoices:nn 9518 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .multichoice: }
9519 { \keys_multichoice_make: }
9520 \cs_new_protected:cpn { \c_keys_props_root_tl .multichoices:nn } #1
9521 { \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 9522 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_set:N } #1
.skip_gset:N 9523 { \keys_variable_set:NnN #1 { skip } n }
.skip_gset:c 9524 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_set:c } #1
9525 { \keys_variable_set:cnN {#1} { skip } n }
9526 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_gset:N } #1
9527 { \keys_variable_set:NnNN #1 { skip } g n }
9528 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_gset:c } #1
9529 { \keys_variable_set:cnNN {#1} { skip } g n }
(End definition for .skip_set:N and .skip_set:c. These functions are documented on page 156.)
```

`.tl_set:N` Setting a variable is very easy: just pass the data along.

```
.tl_set:c 9530 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set:N } #1
.tl_gset:N 9531 { \keys_variable_set:NnN #1 { tl } n }
.tl_gset:c 9532 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set:c } #1
9533 { \keys_variable_set:cnN {#1} { tl } n }
.tl_set_x:N 9534 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set_x:N } #1
.tl_set_x:c 9535 { \keys_variable_set:NnN #1 { tl } x }
.tl_gset_x:N
.tl_gset_x:c
```

```

9536 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set_x:c } #1
9537 { \keys_variable_set:cnN {#1} { tl } x }
9538 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset:N } #1
9539 { \keys_variable_set:NnNN #1 { tl } g n }
9540 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset:c } #1
9541 { \keys_variable_set:cnNN {#1} { tl } g n }
9542 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset_x:N } #1
9543 { \keys_variable_set:NnNN #1 { tl } g x }
9544 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset_x:c } #1
9545 { \keys_variable_set:cnNN {#1} { tl } g x }

```

(End definition for .tl_set:N and .tl_set:c. These functions are documented on page 157.)

.value_forbidden: These are very similar, so both call the same function.

```

.value_required: 9546 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_forbidden: }
9547 { \keys_value_requirement:n { forbidden } }
9548 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_required: }
9549 { \keys_value_requirement:n { required } }

```

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

199.6 Setting keys

\keys_set:nn A simple wrapper again.

```

\keys_set:nV 9550 \cs_new_protected:Npn \keys_set:nn
\keys_set:nv 9551 { \keys_set_aux:onN { \l_keys_module_tl } }
\keys_set:no 9552 \cs_new_protected:Npn \keys_set_aux:nnn #1#2#3
\keys_set_aux:nnn 9553 {
\keys_set_aux:onN 9554   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9555   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9556   \tl_set:Nn \l_keys_module_tl {#1}
9557 }
9558 \cs_generate_variant:Nn \keys_set:nn { nV , nv , no }
9559 \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 9560 \cs_new_protected:Npn \keys_set_known:nnN
\keys_set_known:nvN 9561 { \keys_set_known_aux:onN { \l_keys_module_tl } }
\keys_set_known:noN 9562 \cs_new_protected:Npn \keys_set_known_aux:nnnN #1#2#3#4
\keys_set_known_aux:nnnN 9563 {
\keys_set_known_aux:onN 9564   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9565   \clist_clear:N \l_keys_unknown_clist
9566   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_alt:
9567   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9568   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_std:
9569   \tl_set:Nn \l_keys_module_tl {#1}
9570   \clist_set_eq:NN #4 \l_keys_unknown_clist
9571 }
9572 \cs_generate_variant:Nn \keys_set_known:nnN { nV , nv , no }
9573 \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.

```

9574 \cs_new_protected_nopar:Npn \keys_set_elt:n #1
9575 {
9576   \bool_set_true:N \l_keys_no_value_bool
9577   \keys_set_elt_aux:nn {#1} { }
9578 }
9579 \cs_new_protected:Npn \keys_set_elt:nn #1#2
9580 {
9581   \bool_set_false:N \l_keys_no_value_bool
9582   \keys_set_elt_aux:nn {#1} {#2}
9583 }
9584 \cs_new_protected:Npn \keys_set_elt_aux:nn #1#2
9585 {
9586   \tl_set:Nx \l_keys_key_tl { \tl_to_str:n {#1} }
9587   \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / \l_keys_key_tl }
9588   \keys_value_or_default:n {#2}
9589   \bool_if:nTF
9590   {
9591     \keys_if_value_p:n { required } &&
9592     \l_keys_no_value_bool
9593   }
9594   {
9595     \msg_kernel_error:nnx { keys } { value-required }
9596     { \l_keys_path_tl }
9597   }
9598   {
9599     \bool_if:nTF
9600     {
9601       \keys_if_value_p:n { forbidden } &&
9602       ! \l_keys_no_value_bool
9603     }
9604     {
9605       \msg_kernel_error:nnxx { keys } { value-forbidden }
9606       { \l_keys_path_tl } { \l_keys_value_tl }
9607     }
9608     { \keys_execute: }
9609   }
9610 }

```

(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.

```

9611 \cs_new_protected:Npn \keys_value_or_default:n #1
9612 {

```

```

9613 \tl_set:Nn \l_keys_value_tl {#1}
9614 \bool_if:NT \l_keys_no_value_bool
9615 {
9616   \quark_if_no_value:cF { \c_keys_vars_root_tl \l_keys_path_tl .default }
9617   {
9618     \cs_if_exist:cT { \c_keys_vars_root_tl \l_keys_path_tl .default }
9619     {
9620       \tl_set_eq:Nc \l_keys_value_tl
9621       { \c_keys_vars_root_tl \l_keys_path_tl .default }
9622     }
9623   }
9624 }
9625 }

```

(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.

```

9626 \prg_new_conditional:Npnn \keys_if_value:n #1 { p }
9627 {
9628   \tl_if_eq:ccTF { c_keys_value_ #1 _tl }
9629   { \c_keys_vars_root_tl \l_keys_path_tl .req }
9630   { \prg_return_true: }
9631   { \prg_return_false: }
9632 }

```

(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
9633 \cs_new_nopar:Npn \keys_execute:
9634 { \keys_execute:nn { \l_keys_path_tl } { \keys_execute_unknown: } }
9635 \cs_new_nopar:Npn \keys_execute_unknown:
9636 {
9637   \keys_execute:nn { \l_keys_module_tl / unknown }
9638   {
9639     \msg_kernel_error:nxxx { keys } { key-unknown }
9640     { \l_keys_path_tl } { \l_keys_module_tl }
9641   }
9642 }
9643 \cs_new_eq:NN \keys_execute_unknown_std: \keys_execute_unknown:
9644 \cs_new_nopar:Npn \keys_execute_unknown_alt:
9645 {
9646   \clist_put_right:Nx \l_keys_unknown_clist
9647   {
9648     \exp_not:o \l_keys_key_tl
9649     \bool_if:NF \l_keys_no_value_bool
9650     { = { \exp_not:o \l_keys_value_tl } }
9651   }
9652 }
9653 \cs_new_nopar:Npn \keys_execute:nn #1#2

```

```

9654 {
9655   \cs_if_exist:cTF { \c_keys_code_root_tl #1 }
9656   {
9657     \exp_args:Nno \use:c { \c_keys_code_root_tl #1 }
9658     \l_keys_value_tl
9659   }
9660   {#2}
9661 }

```

(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.

```

9662 \cs_new_nopar:Npn \keys_choice_find:n #1
9663 {
9664   \keys_execute:nn { \l_keys_path_tl / \tl_to_str:n {#1} }
9665   { \keys_execute:nn { \l_keys_path_tl / unknown } { } }
9666 }

```

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

199.7 Utilities

\keys_if_exist:nn A utility for others to see if a key exists.

```

9667 \prg_new_conditional:Npnn \keys_if_exist:nn #1#2 { p , T , F , TF }
9668 {
9669   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 }
9670   { \prg_return_true: }
9671   { \prg_return_false: }
9672 }

```

(End definition for \keys_if_exist:nn. This function is documented on page 161.)

\keys_if_choice_exist:nnn Just an alternative view on \keys_if_exist:nn(TF).

```

9673 \prg_new_conditional:Npnn \keys_if_choice_exist:nnn #1#2#3 { p , T , F , TF }
9674 {
9675   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 / #3 }
9676   { \prg_return_true: }
9677   { \prg_return_false: }
9678 }

```

(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.

```

9679 \cs_new_nopar:Npn \keys_show:nn #1#2
9680 { \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 161.)

199.8 Messages

For when there is a need to complain.

```
9681 \msg_kernel_new:nnnn { keys } { boolean-values-only }
9682 { Key~'#1'~accepts~boolean~values~only. }
9683 { The~key~'#1'~only~accepts~the~values~'true'~and~'false'. }
9684 \msg_kernel_new:nnnn { keys } { choice-unknown }
9685 { Choice~'#2'~unknown~for~key~'#1'. }
9686 {
9687   The~key~'#1'~takes~a~limited~number~of~values.\\
9688   The~input~given,~'#2',~is~not~on~the~list~accepted.
9689 }
9690 \msg_kernel_new:nnnn { keys } { generate-choices-before-code }
9691 { No~code~available~to~generate~choices~for~key~'#1'. }
9692 {
9693   \c_msg_coding_error_text_tl
9694   Before~using~.generate_choices:n~the~code~should~be~defined~
9695   with~'.choice_code:n'~or~'.choice_code:x'.
9696 }
9697 \msg_kernel_new:nnnn { keys } { key-no-property }
9698 { No~property~given~in~definition~of~key~'#1'. }
9699 {
9700   \c_msg_coding_error_text_tl
9701   Inside~\keys_define:nn~each~key~name
9702   needs~a~property: \\
9703   ~ ~ #1 .<property> \\
9704   LaTeX~did~not~find~a~'. '~to~indicate~the~start~of~a~property.
9705 }
9706 \msg_kernel_new:nnnn { keys } { key-unknown }
9707 { The~key~'#1'~is~unknown~and~is~being~ignored. }
9708 {
9709   The~module~'#2'~does~not~have~a~key~called~'#1'.\\
9710   Check~that~you~have~spelled~the~key~name~correctly.
9711 }
9712 \msg_kernel_new:nnnn { keys } { option-unknown }
9713 { Unknown~option~'#1'~for~package~#2. }
9714 {
9715   LaTeX~has~been~asked~to~set~an~option~called~'#1'~
9716   but~the~#2~package~has~not~created~an~option~with~this~name.
9717 }
9718 \msg_kernel_new:nnnn { keys } { property-requires-value }
9719 { The~property~'#1'~requires~a~value. }
9720 {
9721   \c_msg_coding_error_text_tl
9722   LaTeX~was~asked~to~set~property~'#2'~for~key~'#1'.\\
9723   No~value~was~given~for~the~property,~and~one~is~required.
9724 }
9725 \msg_kernel_new:nnnn { keys } { property-unknown }
9726 { The~key~property~'#1'~is~unknown. }
9727 {
```

```

9728 \c_msg_coding_error_text_tl
9729 LaTeX-has-been-asked-to-set-the-property~'#1'~for-key~'#2':~
9730 this-property-is-not-defined.
9731 }
9732 \msg_kernel_new:nnnn { keys } { value-forbidden }
9733 { The-key~'~'~#1'~does-not-taken-a-value. }
9734 {
9735   The-key~'~'~#1'~should-be-given-without-a-value.\\
9736   LaTeX-will-ignore-the-given-value~'~'~#2'.
9737 }
9738 \msg_kernel_new:nnnn { keys } { value-required }
9739 { The-key~'~'~#1'~requires-a-value. }
9740 {
9741   The-key~'~'~#1'~must-have-a-value.\\
9742   No-value-was-present:~the-key-will-be-ignored.
9743 }

```

199.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

```

\KV_process_space_removal_sanitize:NNn There is just one function for this now.
\KV_process_space_removal_no_sanitize:NNn
\KV_process_no_space_removal_no_sanitize:NNn
9744 <*deprecated>
9745 \cs_new_eq:NN \KV_process_space_removal_sanitize:NNn \keyval_parse:NNn
9746 \cs_new_eq:NN \KV_process_space_removal_no_sanitize:NNn \keyval_parse:NNn
9747 \cs_new_eq:NN \KV_process_no_space_removal_no_sanitize:NNn \keyval_parse:NNn
9748 </deprecated>
(End definition for \KV_process_space_removal_sanitize:NNn. This function is documented on
page ??.)
9749 </initex | package>

```

200 l3file implementation

The following test files are used for this code: m3file001.

```

9750 <*initex | package>
9751 <*package>
9752 \ProvidesExplPackage
9753   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9754 \package_check_loaded_expl:
9755 </package>

\g_file_current_name_tl The name of the current file should be available at all times.
9756 \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.

```

9757 <*initex>
9758 \tex_everyjob:D \exp_after:wN
9759 {
9760   \tex_the:D \tex_everyjob:D
9761   \tl_gset:Nx \g_file_current_name_tl { \tex_jobname:D }
9762 }
9763 </initex>
9764 <*package>
9765 \tl_gset_eq:NN \g_file_current_name_tl \@currname
9766 </package>

```

(End definition for \g_file_current_name_tl. This function is documented on page 163.)

`\g_file_stack_seq` The input list of files is stored as a sequence stack.

```

9767 \seq_new:N \g_file_stack_seq

```

(End definition for \g_file_stack_seq. This function is documented on page 164.)

`\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.

```

9768 \seq_new:N \g_file_record_seq

```

The current file name should be included in the file list!

```

9769 <*initex>
9770 \tex_everyjob:D \exp_after:wN
9771 {
9772   \tex_the:D \tex_everyjob:D
9773   \seq_gput_right:NV \g_file_record_seq \g_file_current_name_tl
9774 }
9775 </initex>

```

(End definition for \g_file_record_seq. This function is documented on page 164.)

`\l_file_name_tl` Used to return the fully-qualified name of a file.

```

9776 \tl_new:N \l_file_name_tl

```

(End definition for \l_file_name_tl. This function is documented on page 164.)

`\l_file_search_path_seq` The current search path.

```

9777 \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_search_path_saved_seq` The current search path has to be saved for package use.

```

9778 <*package>
9779 \seq_new:N \l_file_search_path_saved_seq
9780 </package>

```

(End definition for \l_file_search_path_saved_seq. This function is documented on page 164.)

`\l_file_tmpa_seq` Scratch space for comma list conversion in package mode.

```

9781 <*package>
9782 \seq_new:N \l_file_tmpa_seq
9783 </package>
(End definition for \l_file_tmpa_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_test_stream` report end-of-file. For files which are in the current directory, this is straight-forward.
`\file_add_path_search:nN` For other locations, a search has to be made looking at each potential path in turn. The
first location is of course treated as the correct one. If nothing is found, #2 is returned
empty.

```

9784 \cs_new_protected_nopar:Npn \file_add_path:nN #1#2
9785 {
9786   \ior_open:Nn \g_file_test_stream {#1}
9787   \ior_if_eof:NTF \g_file_test_stream
9788     { \file_add_path_search:nN {#1} #2 }
9789     {
9790       \ior_close:N \g_file_test_stream
9791       \tl_set:Nx #2 {#1}
9792     }
9793 }
9794 \cs_new_protected_nopar:Npn \file_add_path_search:nN #1#2
9795 {
9796   \tl_clear:N #2
9797   <*package>
9798   \cs_if_exist:NT \input@path
9799   {
9800     \seq_set_eq:NN \l_file_search_path_saved_seq \l_file_search_path_seq
9801     \seq_set_from_clist:NN \l_file_tmpa_seq \input@path
9802     \seq_concat:NNN \l_file_search_path_seq
9803       \l_file_search_path_seq \l_file_tmpa_seq
9804   }
9805 </package>
9806   \seq_map_inline:Nn \l_file_search_path_seq
9807   {
9808     \ior_open:Nn \g_file_test_stream { ##1 #1 }
9809     \ior_if_eof:NF \g_file_test_stream
9810     {
9811       \tl_set:Nx #2 { ##1 #1 }
9812       \seq_map_break:
9813     }
9814   }
9815 <*package>
9816   \cs_if_exist:NT \input@path
9817   { \seq_set_eq:NN \l_file_search_path_seq \l_file_search_path_saved_seq }
9818 </package>
9819   \ior_close:N \g_file_test_stream
9820 }
(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.

```

9821 \prg_new_protected_conditional:Nnn \file_if_exist:n { T , F , TF }
9822 {
9823   \file_add_path:nN {#1} \l_file_name_tl
9824   \tl_if_empty:NTF \l_file_name_tl
9825     { \prg_return_false: }
9826     { \prg_return_true: }
9827 }

```

(End definition for `\file_if_exist:n`. This function is documented on page 163.)

`\file_input:n` Loading a file is done in a safe way, checking first that the file exists and loading only if it does.

```

9828 \cs_new_protected_nopar:Npn \file_input:n #1
9829 {
9830   \file_add_path:nN {#1} \l_file_name_tl
9831   \tl_if_empty:NF \l_file_name_tl
9832   {
9833     <*initex>
9834     \seq_gput_right:Nx \g_file_record_seq {#1}
9835     </initex>
9836     <*package>
9837     \@addtofilelist {#1}
9838     </package>
9839     \seq_gpush:NV \g_file_stack_seq \g_file_current_name_tl
9840     \tl_gset:Nn \g_file_current_name_tl {#1}
9841     \exp_after:wN \tex_input:D \l_file_name_tl \c_space_tl
9842     \seq_gpop:NN \g_file_stack_seq \g_file_current_name_tl
9843   }
9844 }

```

(End definition for `\file_input:n`. This function is documented on page 164.)

`\file_path_include:n` Wrapper functions to manage the search path.

```

\file_path_remove:n
9845 \cs_new_protected_nopar:Npn \file_path_include:n #1
9846 {
9847   \seq_if_in:NnF \l_file_search_path_seq {#1}
9848     { \seq_put_right:Nn \l_file_search_path_seq {#1} }
9849 }
9850 \cs_new_protected_nopar:Npn \file_path_remove:n #1
9851 { \seq_remove_all:Nn \l_file_search_path_seq {#1} }

```

(End definition for `\file_path_include:n`. This function is documented on page 164.)

`\file_list:` A function to list all files used to the log.

```

9852 \cs_new_protected_nopar:Npn \file_list:
9853 {
9854   \seq_remove_duplicates:N \g_file_record_seq
9855   \iow_log:n { *~File~List~* }

```

```

9856 \seq_map_inline:Nn \g_file_record_seq { \iow_log:n {##1} }
9857 \iow_log:n { ***** }
9858 }

```

(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.

```

9859 <*package>
9860 \AtBeginDocument
9861 {
9862   \seq_set_from_clist:NN \l_file_tmpa_seq \@filelist
9863   \seq_gconcat:NNN \g_file_record_seq \g_file_record_seq \l_file_tmpa_seq
9864 }
9865 </package>
9866 </initex | package>

```

201 l3fp Implementation

The following test files are used for this code: m3fp003.lvt.

```

9867 <*initex | package>
9868 <*package>
9869 \ProvidesExplPackage
9870   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9871 \package_check_loaded_expl:
9872 </package>

```

201.1 Constants

\c_forty_four There is some speed to gain by moving numbers into fixed positions.

```

\c_one_million 9873 \int_const:Nn \c_forty_four { 44 }
\c_one_hundred_million 9874 \int_const:Nn \c_one_million { 1 000 000 }
\c_five_hundred_million 9875 \int_const:Nn \c_one_hundred_million { 100 000 000 }
\c_one_thousand_million 9876 \int_const:Nn \c_five_hundred_million { 500 000 000 }
9877 \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 9878 \int_new:N \c_fp_pi_by_four_decimal_int
\c_fp_pi_decimal_int 9879 \int_set:Nn \c_fp_pi_by_four_decimal_int { 785 398 158 }
\c_fp_pi_extended_int 9880 \int_new:N \c_fp_pi_by_four_extended_int
\c_fp_two_pi_decimal_int 9881 \int_set:Nn \c_fp_pi_by_four_extended_int { 897 448 310 }
\c_fp_two_pi_extended_int 9882 \int_new:N \c_fp_pi_decimal_int
9883 \int_set:Nn \c_fp_pi_decimal_int { 141 592 653 }
9884 \int_new:N \c_fp_pi_extended_int
9885 \int_set:Nn \c_fp_pi_extended_int { 589 793 238 }
9886 \int_new:N \c_fp_two_pi_decimal_int
9887 \int_set:Nn \c_fp_two_pi_decimal_int { 283 185 307 }

```

9888 `\int_new:N \c_fp_two_pi_extended_int`
9889 `\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”.

9890 `\tl_const:Nn \c_e_fp { + 2.718281828 e 0 }`
(End definition for \c_e_fp. This function is documented on page 172.)

`\c_one_fp` The constant value 1: used for fast comparisons.

9891 `\tl_const:Nn \c_one_fp { + 1.000000000 e 0 }`
(End definition for \c_one_fp. This function is documented on page 172.)

`\c_pi_fp` The value π as a “machine number”.

9892 `\tl_const:Nn \c_pi_fp { + 3.141592654 e 0 }`
(End definition for \c_pi_fp. This function is documented on page 172.)

`\c_undefined_fp` A marker for undefined values.

9893 `\tl_const:Nn \c_undefined_fp { X 0.000000000 e 0 }`
(End definition for \c_undefined_fp. This function is documented on page 172.)

`\c_zero_fp` The constant zero value.

9894 `\tl_const:Nn \c_zero_fp { + 0.000000000 e 0 }`
(End definition for \c_zero_fp. This function is documented on page 172.)

201.2 Variables

`\l_fp_arg_tl` A token list to store the formalised representation of the input for transcendental functions.

9895 `\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.

9896 `\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.

9897 `\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` 9898 `\int_new:N \l_fp_exp_integer_int`
9899 `\int_new:N \l_fp_exp_decimal_int`
`\l_fp_exp_extended_int` 9900 `\int_new:N \l_fp_exp_extended_int`
`\l_fp_exp_exponent_int` 9901 `\int_new:N \l_fp_exp_exponent_int`
(End definition for \l_fp_exp_integer_int. This function is documented on page ??.)

<pre> \l_fp_input_a_sign_int \l_fp_input_a_integer_int \l_fp_input_a_decimal_int \l_fp_input_a_exponent_int \l_fp_input_b_sign_int \l_fp_input_b_integer_int \l_fp_input_b_decimal_int \l_fp_input_b_exponent_int </pre>	<p>Storage for the input: two storage areas as there are at most two inputs.</p> <pre> 9902 \int_new:N \l_fp_input_a_sign_int 9903 \int_new:N \l_fp_input_a_integer_int 9904 \int_new:N \l_fp_input_a_decimal_int 9905 \int_new:N \l_fp_input_a_exponent_int 9906 \int_new:N \l_fp_input_b_sign_int 9907 \int_new:N \l_fp_input_b_integer_int 9908 \int_new:N \l_fp_input_b_decimal_int 9909 \int_new:N \l_fp_input_b_exponent_int </pre> <p><i>(End definition for \l_fp_input_a_sign_int. This function is documented on page ??.)</i></p>
<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> 9910 \int_new:N \l_fp_input_a_extended_int 9911 \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_a_vi_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> 9912 \int_new:N \l_fp_mul_a_i_int 9913 \int_new:N \l_fp_mul_a_ii_int 9914 \int_new:N \l_fp_mul_a_iii_int 9915 \int_new:N \l_fp_mul_a_iv_int 9916 \int_new:N \l_fp_mul_a_v_int 9917 \int_new:N \l_fp_mul_a_vi_int 9918 \int_new:N \l_fp_mul_b_i_int 9919 \int_new:N \l_fp_mul_b_ii_int 9920 \int_new:N \l_fp_mul_b_iii_int 9921 \int_new:N \l_fp_mul_b_iv_int 9922 \int_new:N \l_fp_mul_b_v_int 9923 \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> 9924 \int_new:N \l_fp_mul_output_int 9925 \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> 9926 \int_new:N \l_fp_output_sign_int 9927 \int_new:N \l_fp_output_integer_int 9928 \int_new:N \l_fp_output_decimal_int 9929 \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> 9930 \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>

`\l_fp_round_carry_bool` To indicate that a digit needs to be carried forward.

9931 `\bool_new:N \l_fp_round_carry_bool`
(End definition for \l_fp_round_carry_bool. This function is documented on page ??.)

`\l_fp_round_decimal_tl` A temporary store when rounding, to build up the decimal part without needing to do any maths.

9932 `\tl_new:N \l_fp_round_decimal_tl`
(End definition for \l_fp_round_decimal_tl. This function is documented on page ??.)

`\l_fp_round_position_int` Used to check the position for rounding.

`\l_fp_round_target_int` 9933 `\int_new:N \l_fp_round_position_int`
9934 `\int_new:N \l_fp_round_target_int`
(End definition for \l_fp_round_position_int. This function is documented on page ??.)

`\l_fp_sign_tl` There are places where the sign needs to be set up “early”, so that the registers can be re-used.

9935 `\tl_new:N \l_fp_sign_tl`
(End definition for \l_fp_sign_tl. This function is documented on page ??.)

`\l_fp_split_sign_int` 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.

9936 `\int_new:N \l_fp_split_sign_int`
(End definition for \l_fp_split_sign_int. This function is documented on page ??.)

`\l_fp_tmp_int` A scratch int: used only where the value is not carried forward.

9937 `\int_new:N \l_fp_tmp_int`
(End definition for \l_fp_tmp_int. This function is documented on page ??.)

`\l_fp_tmp_tl` A scratch token list variable for expanding material.

9938 `\tl_new:N \l_fp_tmp_tl`
(End definition for \l_fp_tmp_tl. This function is documented on page ??.)

`\l_fp_trig_octant_int` To track which octant the trigonometric input is in.

9939 `\int_new:N \l_fp_trig_octant_int`
(End definition for \l_fp_trig_octant_int. This function is documented on page ??.)

`\l_fp_trig_sign_int` Used for the calculation of trigonometric values.

`\l_fp_trig_decimal_int` 9940 `\int_new:N \l_fp_trig_sign_int`
9941 `\int_new:N \l_fp_trig_decimal_int`
9942 `\int_new:N \l_fp_trig_extended_int`
(End definition for \l_fp_trig_sign_int. This function is documented on page ??.)

201.3 Parsing numbers

`\fp_read:N` Reading a stored value is made easier as the format is designed to match the delimited function. This is always used to read the first value (register a).

```
\fp_read_aux:w
9943 \cs_new_protected_nopar:Npn \fp_read:N #1
9944 { \exp_after:wN \fp_read_aux:w #1 \q_stop }
9945 \cs_new_protected_nopar:Npn \fp_read_aux:w #1#2 . #3 e #4 \q_stop
9946 {
9947   \if:w #1 -
9948     \l_fp_input_a_sign_int \c_minus_one
9949   \else:
9950     \l_fp_input_a_sign_int \c_one
9951   \fi:
9952   \l_fp_input_a_integer_int #2 \scan_stop:
9953   \l_fp_input_a_decimal_int #3 \scan_stop:
9954   \l_fp_input_a_exponent_int #4 \scan_stop:
9955 }
```

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

`\fp_split:Nn` The aim here is to use as much of TeX's mechanism as possible to pick up the numerical input without any mistakes. In particular, negative numbers have to be filtered out first in case the integer part is 0 (in which case TeX 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.

```
\fp_split_aux_i:w
\fp_split_aux_ii:w
\fp_split_aux_iii:w
\fp_split_decimal:w
\fp_split_decimal_aux:w
9956 \cs_new_protected_nopar:Npn \fp_split:Nn #1#2
9957 {
9958   \tl_set:Nx \l_fp_tmp_tl {#2}
9959   \tl_set_rescan:Nno \l_fp_tmp_tl { \char_set_catcode_ignore:n { 32 } }
9960   { \l_fp_tmp_tl }
9961   \l_fp_split_sign_int \c_one
9962   \fp_split_sign:
9963   \use:c { l_fp_input_ #1 _sign_int } \l_fp_split_sign_int
9964   \exp_after:wN \fp_split_exponent:w \l_fp_tmp_tl e e \q_stop #1
9965 }
9966 \cs_new_protected_nopar:Npn \fp_split_sign:
9967 {
9968   \if_int_compare:w \pdfTeX_strcmp:D
9969   { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { - }
9970   = \c_zero
9971   \tl_set:Nx \l_fp_tmp_tl
9972   {
9973     \exp_after:wN
9974     \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
9975   }
9976   \l_fp_split_sign_int -\l_fp_split_sign_int
9977   \exp_after:wN \fp_split_sign:
9978   \else:
```



```

9979 \if_int_compare:w \pdfTeX_strcmp:D
9980 { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { + }
9981 = \c_zero
9982 \tl_set:Nx \l_fp_tmp_tl
9983 {
9984 \exp_after:wN
9985 \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
9986 }
9987 \exp_after:wN \exp_after:wN \exp_after:wN \fp_split_sign:
9988 \fi:
9989 \fi:
9990 }
9991 \cs_new_protected_nopar:Npn \fp_split_exponent:w #1 e #2 e #3 \q_stop #4
9992 {
9993 \use:c { l_fp_input_ #4 _exponent_int }
9994 \int_eval:w 0 #2 \scan_stop:
9995 \tex_afterassignment:D \fp_split_aux_i:w
9996 \use:c { l_fp_input_ #4 _integer_int }
9997 \int_eval:w 0 #1 . . \q_stop #4
9998 }
9999 \cs_new_protected_nopar:Npn \fp_split_aux_i:w #1 . #2 . #3 \q_stop
10000 { \fp_split_aux_ii:w #2 000000000 \q_stop }
10001 \cs_new_protected_nopar:Npn \fp_split_aux_ii:w #1#2#3#4#5#6#7#8#9
10002 { \fp_split_aux_iii:w {#1#2#3#4#5#6#7#8#9} }
10003 \cs_new_protected_nopar:Npn \fp_split_aux_iii:w #1#2 \q_stop
10004 {
10005 \l_fp_tmp_int 1 #1 \scan_stop:
10006 \exp_after:wN \fp_split_decimal:w
10007 \int_use:N \l_fp_tmp_int 000000000 \q_stop
10008 }
10009 \cs_new_protected_nopar:Npn \fp_split_decimal:w #1#2#3#4#5#6#7#8#9
10010 { \fp_split_decimal_aux:w {#2#3#4#5#6#7#8#9} }
10011 \cs_new_protected_nopar:Npn \fp_split_decimal_aux:w #1#2#3 \q_stop #4
10012 {
10013 \use:c { l_fp_input_ #4 _decimal_int } #1#2 \scan_stop:
10014 \if_int_compare:w
10015 \int_eval:w
10016 \use:c { l_fp_input_ #4 _integer_int } +
10017 \use:c { l_fp_input_ #4 _decimal_int }
10018 \scan_stop:
10019 = \c_zero
10020 \use:c { l_fp_input_ #4 _sign_int } \c_one
10021 \fi:
10022 \if_int_compare:w
10023 \use:c { l_fp_input_ #4 _integer_int } < \c_one_thousand_million
10024 \else:
10025 \exp_after:wN \fp_overflow_msg:
10026 \fi:
10027 }

```

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

`\fp_standardise:NNNN` The idea here is to shift the input into a known exponent range. This is done using \TeX tokens where possible, as this is faster than arithmetic.

```

\fp_standardise_aux:NNNN
\fp_standardise_aux:w
\fp_standardise_aux:w
10028 \cs_new_protected_nopar:Npn \fp_standardise:NNNN #1#2#3#4
10029 {
10030   \if_int_compare:w
10031     \int_eval:w #2 + #3 = \c_zero
10032     #1 \c_one
10033     #4 \c_zero
10034     \exp_after:wN \use_none:nnnn
10035   \else:
10036     \exp_after:wN \fp_standardise_aux:NNNN
10037   \fi:
10038   #1#2#3#4
10039 }
10040 \cs_new_protected_nopar:Npn \fp_standardise_aux:NNNN #1#2#3#4
10041 {
10042   \cs_set_protected_nopar:Npn \fp_standardise_aux:
10043     {
10044       \if_int_compare:w #2 = \c_zero
10045       \tex_advance:D #3 \c_one_thousand_million
10046       \exp_after:wN \fp_standardise_aux:w
10047       \int_use:N #3 \q_stop
10048       \exp_after:wN \fp_standardise_aux:
10049     \fi:
10050   }
10051   \cs_set_protected_nopar:Npn
10052     \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9 \q_stop
10053   {
10054     #2 ##2 \scan_stop:
10055     #3 ##3##4##5##6##7##8##9 0 \scan_stop:
10056     \tex_advance:D #4 \c_minus_one
10057   }
10058   \fp_standardise_aux:
10059   \cs_set_protected_nopar:Npn \fp_standardise_aux:
10060     {
10061       \if_int_compare:w #2 > \c_nine
10062       \tex_advance:D #2 \c_one_thousand_million
10063       \exp_after:wN \use_i:nn \exp_after:wN
10064       \fp_standardise_aux:w \int_use:N #2
10065       \exp_after:wN \fp_standardise_aux:
10066     \fi:
10067   }
10068   \cs_set_protected_nopar:Npn
10069     \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9
10070   {
10071     #2 ##1##2##3##4##5##6##7##8 \scan_stop:
10072     \tex_advance:D #3 \c_one_thousand_million
10073     \tex_divide:D #3 \c_ten
10074     \tl_set:Nx \l_fp_tmp_tl

```

```

10075         {
10076             ##9
10077             \exp_after:wN \use_none:n \int_use:N #3
10078         }
10079         #3 \l_fp_tmp_tl \scan_stop:
10080         \tex_advance:D #4 \c_one
10081     }
10082     \fp_standardise_aux:
10083     \if_int_compare:w #4 < \c_one_hundred
10084     \if_int_compare:w #4 > -\c_one_hundred
10085     \else:
10086         #1 \c_one
10087         #2 \c_zero
10088         #3 \c_zero
10089         #4 \c_zero
10090     \fi:
10091     \else:
10092     \exp_after:wN \fp_overflow_msg:
10093     \fi:
10094 }
10095 \cs_new_protected_nopar:Npn \fp_standardise_aux: { }
10096 \cs_new_protected_nopar:Npn \fp_standardise_aux:w { }

```

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

201.4 Internal utilities

\fp_level_input_exponents: The routines here are similar to those used to standardise the exponent. However, the aim here is different: the two exponents need to end up the same.

```

\fp_level_input_exponents_a:NNNNNNNNN
\fp_level_input_exponents_b:NNNNNNNNN

```

```

10097 \cs_new_protected_nopar:Npn \fp_level_input_exponents:
10098 {
10099     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10100     \exp_after:wN \fp_level_input_exponents_a:
10101     \else:
10102     \exp_after:wN \fp_level_input_exponents_b:
10103     \fi:
10104 }
10105 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:
10106 {
10107     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10108     \tex_advance:D \l_fp_input_b_integer_int \c_one_thousand_million
10109     \exp_after:wN \use_i:nn \exp_after:wN
10110     \fp_level_input_exponents_a:NNNNNNNNN
10111     \int_use:N \l_fp_input_b_integer_int
10112     \exp_after:wN \fp_level_input_exponents_a:
10113     \fi:
10114 }
10115 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:NNNNNNNNN
10116 #1#2#3#4#5#6#7#8#9
10117 {

```

```

10118 \l_fp_input_b_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10119 \tex_advance:D \l_fp_input_b_decimal_int \c_one_thousand_million
10120 \tex_divide:D \l_fp_input_b_decimal_int \c_ten
10121 \tl_set:Nx \l_fp_tmp_tl
10122 {
10123     #9
10124     \exp_after:wN \use_none:n
10125     \int_use:N \l_fp_input_b_decimal_int
10126 }
10127 \l_fp_input_b_decimal_int \l_fp_tmp_tl \scan_stop:
10128 \tex_advance:D \l_fp_input_b_exponent_int \c_one
10129 }
10130 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:
10131 {
10132     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
10133     \tex_advance:D \l_fp_input_a_integer_int \c_one_thousand_million
10134     \exp_after:wN \use_i:nn \exp_after:wN
10135     \fp_level_input_exponents_b:NNNNNNNNN
10136     \int_use:N \l_fp_input_a_integer_int
10137     \exp_after:wN \fp_level_input_exponents_b:
10138     \fi:
10139 }
10140 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:NNNNNNNNN
10141 #1#2#3#4#5#6#7#8#9
10142 {
10143     \l_fp_input_a_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10144     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10145     \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10146     \tl_set:Nx \l_fp_tmp_tl
10147     {
10148         #9
10149         \exp_after:wN \use_none:n
10150         \int_use:N \l_fp_input_a_decimal_int
10151     }
10152     \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
10153     \tex_advance:D \l_fp_input_a_exponent_int \c_one
10154 }

```

(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.

```

10155 \cs_new_protected_nopar:Npn \fp_tmp:w #1#2 { }

```

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

201.5 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 10156 \cs_new_protected_nopar:Npn \fp_new:N #1
          10157 {
          10158   \tl_new:N #1
          10159   \tl_gset_eq:NN #1 \c_zero_fp
          10160 }
          10161 \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 10162 \cs_new_protected_nopar:Npn \fp_const:Nn #1#2
            10163 {
            10164   \fp_new:N #1
            10165   \fp_gset:Nn #1 {#2}
            10166 }
            10167 \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 10168 \cs_new_protected_nopar:Npn \fp_zero:N #1
\fp_gzero:N 10169 { \tl_set_eq:NN #1 \c_zero_fp }
\fp_gzero:c 10170 \cs_new_protected_nopar:Npn \fp_gzero:N #1
            10171 { \tl_gset_eq:NN #1 \c_zero_fp }
            10172 \cs_generate_variant:Nn \fp_zero:N { c }
            10173 \cs_generate_variant:Nn \fp_gzero:N { c }
            (End definition for \fp_zero:N and \fp_zero:c. These functions are documented on page ??.)

```

`\fp_set:Nn` To trap any input errors, a very simple version of the parser is run here. This will pick up any invalid characters at this stage, saving issues later. The splitting approach is the same as the more advanced function later.

```

\fp_gset:cn 10174 \cs_new_protected_nopar:Npn \fp_set:Nn { \fp_set_aux:NNn \tl_set:Nn }
\fp_set_aux:NNn 10175 \cs_new_protected_nopar:Npn \fp_gset:Nn { \fp_set_aux:NNn \tl_gset:Nn }
              10176 \cs_new_protected_nopar:Npn \fp_set_aux:NNn #1#2#3
              10177 {
              10178   \group_begin:
              10179   \fp_split:Nn a {#3}
              10180   \fp_standardise:NNNN
              10181   \l_fp_input_a_sign_int
              10182   \l_fp_input_a_integer_int
              10183   \l_fp_input_a_decimal_int
              10184   \l_fp_input_a_exponent_int
              10185   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
              10186   \cs_set_protected_nopar:Npx \fp_tmp:w
              10187   {
              10188     \group_end:

```

```

10189         #1 \exp_not:N #2
10190         {
10191             \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10192             -
10193             \else:
10194             +
10195             \fi:
10196             \int_use:N \l_fp_input_a_integer_int
10197             .
10198             \exp_after:wN \use_none:n
10199             \int_use:N \l_fp_input_a_decimal_int
10200             e
10201             \int_use:N \l_fp_input_a_exponent_int
10202         }
10203     }
10204     \fp_tmp:w
10205 }
10206 \cs_generate_variant:Nn \fp_set:Nn { c }
10207 \cs_generate_variant:Nn \fp_gset:Nn { c }

```

(End definition for \fp_set:Nn and \fp_set:cn. These functions are documented on page ??.)

Here, dimensions are converted to fixed-points *via* a temporary variable. This ensures that they always convert as points. The code is then essentially the same as for \fp_set:Nn, but with the dimension passed so that it will be striped of the pt on the way through. The passage through a skip is used to remove any rubber part.

```

\fp_set_from_dim:Nn
\fp_set_from_dim:cn
\fp_gset_from_dim:Nn
\fp_gset_from_dim:cn
\fp_set_from_dim_aux:NNn
\fp_set_from_dim_aux:w
  \l_fp_tmp_dim
  \l_fp_tmp_skip
10208 \cs_new_protected_nopar:Npn \fp_set_from_dim:Nn
10209 { \fp_set_from_dim_aux:NNn \tl_set:Nx }
10210 \cs_new_protected_nopar:Npn \fp_gset_from_dim:Nn
10211 { \fp_set_from_dim_aux:NNn \tl_gset:Nx }
10212 \cs_new_protected_nopar:Npn \fp_set_from_dim_aux:NNn #1#2#3
10213 {
10214     \group_begin:
10215     \l_fp_tmp_skip \etex_glueexpr:D #3 \scan_stop:
10216     \l_fp_tmp_dim \l_fp_tmp_skip
10217     \fp_split:Nn a
10218     {
10219         \exp_after:wN \fp_set_from_dim_aux:w
10220         \dim_use:N \l_fp_tmp_dim
10221     }
10222     \fp_standardise:NNNN
10223     \l_fp_input_a_sign_int
10224     \l_fp_input_a_integer_int
10225     \l_fp_input_a_decimal_int
10226     \l_fp_input_a_exponent_int
10227     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10228     \cs_set_protected_nopar:Npx \fp_tmp:w
10229     {
10230         \group_end:
10231         #1 \exp_not:N #2

```

```

10232         {
10233             \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10234             -
10235             \else:
10236             +
10237             \fi:
10238             \int_use:N \l_fp_input_a_integer_int
10239             .
10240             \exp_after:wN \use_none:n
10241             \int_use:N \l_fp_input_a_decimal_int
10242             e
10243             \int_use:N \l_fp_input_a_exponent_int
10244         }
10245     }
10246     \fp_tmp:w
10247 }
10248 \cs_set_protected_nopar:Npx \fp_set_from_dim_aux:w
10249 {
10250     \cs_set_nopar:Npn \exp_not:N \fp_set_from_dim_aux:w
10251     ##1 \tl_to_str:n { pt } {##1}
10252 }
10253 \fp_set_from_dim_aux:w
10254 \cs_generate_variant:Nn \fp_set_from_dim:Nn { c }
10255 \cs_generate_variant:Nn \fp_gset_from_dim:Nn { c }
10256 \dim_new:N \l_fp_tmp_dim
10257 \skip_new:N \l_fp_tmp_skip

```

(End definition for `\fp_set_from_dim:Nn` and `\fp_set_from_dim:cn`. These functions are documented on page ??.)

`\fp_set_eq:NN` Pretty simple, really.

```

\fp_set_eq:cN 10258 \cs_new_eq:NN \fp_set_eq:NN \tl_set_eq:NN
\fp_set_eq:Nc 10259 \cs_new_eq:NN \fp_set_eq:cN \tl_set_eq:cN
\fp_set_eq:cc 10260 \cs_new_eq:NN \fp_set_eq:Nc \tl_set_eq:Nc
\fp_gset_eq:NN 10261 \cs_new_eq:NN \fp_set_eq:cc \tl_set_eq:cc
\fp_gset_eq:cN 10262 \cs_new_eq:NN \fp_gset_eq:NN \tl_gset_eq:NN
\fp_gset_eq:Nc 10263 \cs_new_eq:NN \fp_gset_eq:cN \tl_gset_eq:cN
\fp_gset_eq:Nc 10264 \cs_new_eq:NN \fp_gset_eq:Nc \tl_gset_eq:Nc
\fp_gset_eq:cc 10265 \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 10266 \cs_new_eq:NN \fp_show:N \tl_show:N
10267 \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 T_EX to use as a number in an expandable manner. The first step is to deal with the sign, then work out how big the input is.

```

\fp_use:c
\fp_use_aux:w
\fp_use_none:w 10268 \cs_new_nopar:Npn \fp_use:N #1
\fp_use_small:w
\fp_use_large:w

```

```

\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

```

```

10269 { \exp_after:wN \fp_use_aux:w #1 \q_stop }
10270 \cs_generate_variant:Nn \fp_use:N { c }
10271 \cs_new_nopar:Npn \fp_use_aux:w #1#2 e #3 \q_stop
10272 {
10273   \if:w #1 -
10274     -
10275   \fi:
10276   \if_int_compare:w #3 > \c_zero
10277     \exp_after:wN \fp_use_large:w
10278   \else:
10279     \if_int_compare:w #3 < \c_zero
10280       \exp_after:wN \exp_after:wN \exp_after:wN
10281       \fp_use_small:w
10282     \else:
10283       \exp_after:wN \exp_after:wN \exp_after:wN \fp_use_none:w
10284     \fi:
10285   \fi:
10286   #2 e #3 \q_stop
10287 }

```

When the exponent is zero, the input is simply returned as output.

```

10288 \cs_new_nopar: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.

```

10289 \cs_new_nopar:Npn \fp_use_small:w #1 . #2 e #3 \q_stop
10290 {
10291   0 .
10292   \prg_replicate:nn { -#3 - 1 } { 0 }
10293   #1#2
10294 }

```

Life is more complex for large numbers. The decimal point needs to be shuffled, with potentially some zero-filling for very large values.

```

10295 \cs_new_nopar:Npn \fp_use_large:w #1 . #2 e #3 \q_stop
10296 {
10297   \if_int_compare:w #3 < \c_ten
10298     \exp_after:wN \fp_use_large_aux_i:w
10299   \else:
10300     \exp_after:wN \fp_use_large_aux_ii:w
10301   \fi:
10302   #1#2 e #3 \q_stop
10303 }
10304 \cs_new_nopar:Npn \fp_use_large_aux_i:w #1#2 e #3 \q_stop
10305 {
10306   #1
10307   \use:c { fp_use_large_aux_ #3 :w } #2 \q_stop
10308 }
10309 \cs_new_nopar:cpn { fp_use_large_aux_1:w } #1#2 \q_stop { #1 . #2 }
10310 \cs_new_nopar:cpn { fp_use_large_aux_2:w } #1#2#3 \q_stop
10311 { #1#2 . #3 }

```



```

10312 \cs_new_nopar:cpn { fp_use_large_aux_3:w } #1#2#3#4 \q_stop
10313 { #1#2#3 . #4 }
10314 \cs_new_nopar:cpn { fp_use_large_aux_4:w } #1#2#3#4#5 \q_stop
10315 { #1#2#3#4 . #5 }
10316 \cs_new_nopar:cpn { fp_use_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10317 { #1#2#3#4#5 . #6 }
10318 \cs_new_nopar:cpn { fp_use_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10319 { #1#2#3#4#5#6 . #7 }
10320 \cs_new_nopar:cpn { fp_use_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10321 { #1#2#3#4#6#7 . #8 }
10322 \cs_new_nopar:cpn { fp_use_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10323 { #1#2#3#4#5#6#7#8 . #9 }
10324 \cs_new_nopar:cpn { fp_use_large_aux_9:w } #1 \q_stop { #1 . }
10325 \cs_new_nopar:Npn \fp_use_large_aux_ii:w #1 e #2 \q_stop
10326 {
10327     #1
10328     \prg_replicate:nn { #2 - 9 } { 0 }
10329     .
10330 }

```

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

201.6 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 10331 \cs_new_nopar:Npn \fp_to_dim:N #1 { \fp_use:N #1 pt }
10332 \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 10333 \cs_new_nopar:Npn \fp_to_int:N #1
\fp_to_int_aux:w 10334 { \exp_after:wN \fp_to_int_aux:w #1 \q_stop }
\fp_to_int_none:w 10335 \cs_generate_variant:Nn \fp_to_int:N { c }
\fp_to_int_small:w 10336 \cs_new_nopar:Npn \fp_to_int_aux:w #1#2 e #3 \q_stop
\fp_to_int_large:w 10337 {
\fp_to_int_large_aux_i:w 10338     \if:w #1 -
\fp_to_int_large_aux_1:w 10339     -
\fp_to_int_large_aux_2:w 10340     \fi:
\fp_to_int_large_aux_3:w 10341     \if_int_compare:w #3 < \c_zero
\fp_to_int_large_aux_4:w 10342     \exp_after:wN \fp_to_int_small:w
\fp_to_int_large_aux_5:w 10343     \else:
\fp_to_int_large_aux_6:w 10344     \exp_after:wN \fp_to_int_large:w
\fp_to_int_large_aux_7:w 10345     \fi:
\fp_to_int_large_aux_8:w 10346     #2 e #3 \q_stop
\fp_to_int_large_aux_i:w 10347 }
\fp_to_int_large_aux:nnn
\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.

```

10348 \cs_new_nopar:Npn \fp_to_int_small:w #1 . #2 e #3 \q_stop
10349 {
10350   \if_int_compare:w #3 > \c_one
10351   \else:
10352     \if_int_compare:w #1 < \c_five
10353     0
10354   \else:
10355     1
10356   \fi:
10357 \fi:
10358 }

```

For large numbers, the idea is to split off the part for rounding, do the rounding and fill if needed.

```

10359 \cs_new_nopar:Npn \fp_to_int_large:w #1 . #2 e #3 \q_stop
10360 {
10361   \if_int_compare:w #3 < \c_ten
10362     \exp_after:wN \fp_to_int_large_aux_i:w
10363   \else:
10364     \exp_after:wN \fp_to_int_large_aux_ii:w
10365   \fi:
10366   #1#2 e #3 \q_stop
10367 }
10368 \cs_new_nopar:Npn \fp_to_int_large_aux_i:w #1#2 e #3 \q_stop
10369 { \use:c { fp_to_int_large_aux_#3 :w } #2 \q_stop {#1} }
10370 \cs_new_nopar:cpn { fp_to_int_large_aux_1:w } #1#2 \q_stop
10371 { \fp_to_int_large_aux:nnn { #2 0 } {#1} }
10372 \cs_new_nopar:cpn { fp_to_int_large_aux_2:w } #1#2#3 \q_stop
10373 { \fp_to_int_large_aux:nnn { #3 00 } {#1#2} }
10374 \cs_new_nopar:cpn { fp_to_int_large_aux_3:w } #1#2#3#4 \q_stop
10375 { \fp_to_int_large_aux:nnn { #4 000 } {#1#2#3} }
10376 \cs_new_nopar:cpn { fp_to_int_large_aux_4:w } #1#2#3#4#5 \q_stop
10377 { \fp_to_int_large_aux:nnn { #5 0000 } {#1#2#3#4} }
10378 \cs_new_nopar:cpn { fp_to_int_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10379 { \fp_to_int_large_aux:nnn { #6 00000 } {#1#2#3#4#5} }
10380 \cs_new_nopar:cpn { fp_to_int_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10381 { \fp_to_int_large_aux:nnn { #7 000000 } {#1#2#3#4#5#6} }
10382 \cs_new_nopar:cpn { fp_to_int_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10383 { \fp_to_int_large_aux:nnn { #8 0000000 } {#1#2#3#4#5#6#7} }
10384 \cs_new_nopar:cpn { fp_to_int_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10385 { \fp_to_int_large_aux:nnn { #9 00000000 } {#1#2#3#4#5#6#7#8} }
10386 \cs_new_nopar:cpn { fp_to_int_large_aux_9:w } #1 \q_stop {#1}
10387 \cs_new_nopar:Npn \fp_to_int_large_aux:nnn #1#2#3
10388 {
10389   \if_int_compare:w #1 < \c_five_hundred_million
10390   #3#2
10391   \else:
10392     \int_value:w \int_eval:w #3#2 + 1 \int_eval_end:

```

```

10393     \fi:
10394   }
10395   \cs_new_nopar:Npn \fp_to_int_large_aux_ii:w #1 e #2 \q_stop
10396   {
10397     #1
10398     \prg_replicate:nn { #2 - 9 } { 0 }
10399   }

```

(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:NNNNNNNN
\fp_to_tl_small_zeros:NNNNNNNN
\fp_use_iix_ix:NNNNNNNNN
\fp_use_ix:NNNNNNNNN
\fp_use_i_to_vii:NNNNNNNNN
\fp_use_i_to_iix:NNNNNNNNN

```

Converting to integers in an expandable manner is very similar to simply using floating point variables, particularly in the lead-off.

```

10400   \cs_new_nopar:Npn \fp_to_tl:N #1
10401   { \exp_after:wN \fp_to_tl_aux:w #1 \q_stop }
10402   \cs_generate_variant:Nn \fp_to_tl:N { c }
10403   \cs_new_nopar:Npn \fp_to_tl_aux:w #1#2 e #3 \q_stop
10404   {
10405     \if:w #1 -
10406     -
10407     \fi:
10408     \if_int_compare:w #3 < \c_zero
10409     \exp_after:wN \fp_to_tl_small:w
10410     \else:
10411     \exp_after:wN \fp_to_tl_large:w
10412     \fi:
10413     #2 e #3 \q_stop
10414   }

```

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.

```

10415   \cs_new_nopar:Npn \fp_to_tl_large:w #1 e #2 \q_stop
10416   {
10417     \if_int_compare:w #2 < \c_ten
10418     \exp_after:wN \fp_to_tl_large_aux_i:w
10419     \else:
10420     \exp_after:wN \fp_to_tl_large_aux_ii:w
10421     \fi:
10422     #1 e #2 \q_stop
10423   }
10424   \cs_new_nopar:Npn \fp_to_tl_large_aux_i:w #1 e #2 \q_stop
10425   { \use:c { fp_to_tl_large_#2 :w } #1 \q_stop }
10426   \cs_new_nopar:Npn \fp_to_tl_large_aux_ii:w #1 . #2 e #3 \q_stop
10427   {
10428     #1
10429     \fp_to_tl_large_zeros:NNNNNNNN #2
10430     e #3
10431   }
10432   \cs_new_nopar:cpn { fp_to_tl_large_0:w } #1 . #2 \q_stop
10433   {

```

```

10434     #1
10435     \fp_to_tl_large_zeros:NNNNNNNNN #2
10436   }
10437   \cs_new_nopar:cpn { fp_to_tl_large_1:w } #1 . #2#3 \q_stop
10438   {
10439     #1#2
10440     \fp_to_tl_large_zeros:NNNNNNNNN #3 0
10441   }
10442   \cs_new_nopar:cpn { fp_to_tl_large_2:w } #1 . #2#3#4 \q_stop
10443   {
10444     #1#2#3
10445     \fp_to_tl_large_zeros:NNNNNNNNN #4 00
10446   }
10447   \cs_new_nopar:cpn { fp_to_tl_large_3:w } #1 . #2#3#4#5 \q_stop
10448   {
10449     #1#2#3#4
10450     \fp_to_tl_large_zeros:NNNNNNNNN #5 000
10451   }
10452   \cs_new_nopar:cpn { fp_to_tl_large_4:w } #1 . #2#3#4#5#6 \q_stop
10453   {
10454     #1#2#3#4#5
10455     \fp_to_tl_large_zeros:NNNNNNNNN #6 0000
10456   }
10457   \cs_new_nopar:cpn { fp_to_tl_large_5:w } #1 . #2#3#4#5#6#7 \q_stop
10458   {
10459     #1#2#3#4#5#6
10460     \fp_to_tl_large_zeros:NNNNNNNNN #7 00000
10461   }
10462   \cs_new_nopar:cpn { fp_to_tl_large_6:w } #1 . #2#3#4#5#6#7#8 \q_stop
10463   {
10464     #1#2#3#4#5#6#7
10465     \fp_to_tl_large_zeros:NNNNNNNNN #8 000000
10466   }
10467   \cs_new_nopar:cpn { fp_to_tl_large_7:w } #1 . #2#3#4#5#6#7#8#9 \q_stop
10468   {
10469     #1#2#3#4#5#6#7#8
10470     \fp_to_tl_large_zeros:NNNNNNNNN #9 0000000
10471   }
10472   \cs_new_nopar:cpn { fp_to_tl_large_8:w } #1 .
10473   {
10474     #1
10475     \use:c { fp_to_tl_large_8_aux:w }
10476   }
10477   \cs_new_nopar:cpn { fp_to_tl_large_8_aux:w } #1#2#3#4#5#6#7#8#9 \q_stop
10478   {
10479     #1#2#3#4#5#6#7#8
10480     \fp_to_tl_large_zeros:NNNNNNNNN #9 00000000
10481   }
10482   \cs_new_nopar: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.

```

10483 \cs_new_nopar:Npn \fp_to_tl_small:w #1 e #2 \q_stop
10484 {
10485   \if_int_compare:w #2 = \c_minus_one
10486     \exp_after:wN \fp_to_tl_small_one:w
10487   \else:
10488     \if_int_compare:w #2 = -\c_two
10489       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_two:w
10490     \else:
10491       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_aux:w
10492     \fi:
10493   \fi:
10494   #1 e #2 \q_stop
10495 }
10496 \cs_new_nopar:Npn \fp_to_tl_small_one:w #1 . #2 e #3 \q_stop
10497 {
10498   \if_int_compare:w \fp_use_ix:NNNNNNNN #2 > \c_four
10499     \if_int_compare:w
10500       \int_eval:w #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10501       < \c_one_thousand_million
10502     0.
10503     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10504     \int_value:w \int_eval:w
10505     #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10506     \int_eval_end:
10507   \else:
10508     1
10509   \fi:
10510 \else:
10511   0. #1
10512   \fp_to_tl_small_zeros:NNNNNNNN #2
10513 \fi:
10514 }
10515 \cs_new_nopar:Npn \fp_to_tl_small_two:w #1 . #2 e #3 \q_stop
10516 {
10517   \if_int_compare:w \fp_use_iix_ix:NNNNNNNN #2 > \c_forty_four
10518     \if_int_compare:w
10519       \int_eval:w #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10520       < \c_one_thousand_million
10521     0.0
10522     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10523     \int_value:w \int_eval:w
10524     #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10525     \int_eval_end:
10526   \else:
10527     0.1
10528   \fi:

```

```

10529     \else:
10530         0.0
10531         #1
10532         \fp_to_tl_small_zeros:NNNNNNNN #2
10533     \fi:
10534 }
10535 \cs_new_nopar:Npn \fp_to_tl_small_aux:w #1 . #2 e #3 \q_stop
10536 {
10537     #1
10538     \fp_to_tl_large_zeros:NNNNNNNN #2
10539     e #3
10540 }

```

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.

```

10541 \cs_new_nopar:Npn \fp_to_tl_large_zeros:NNNNNNNN #1#2#3#4#5#6#7#8#9
10542 {
10543     \if_int_compare:w #9 = \c_zero
10544     \if_int_compare:w #8 = \c_zero
10545     \if_int_compare:w #7 = \c_zero
10546     \if_int_compare:w #6 = \c_zero
10547     \if_int_compare:w #5 = \c_zero
10548     \if_int_compare:w #4 = \c_zero
10549     \if_int_compare:w #3 = \c_zero
10550     \if_int_compare:w #2 = \c_zero
10551     \if_int_compare:w #1 = \c_zero
10552     \else:
10553         . #1
10554     \fi:
10555     \else:
10556         . #1#2
10557     \fi:
10558     \else:
10559         . #1#2#3
10560     \fi:
10561     \else:
10562         . #1#2#3#4
10563     \fi:
10564     \else:
10565         . #1#2#3#4#5
10566     \fi:
10567     \else:
10568         . #1#2#3#4#5#6
10569     \fi:
10570     \else:
10571         . #1#2#3#4#5#6#7
10572     \fi:
10573     \else:
10574         . #1#2#3#4#5#6#7#8
10575     \fi:

```

```

10576     \else:
10577         . #1#2#3#4#5#6#7#8#9
10578     \fi:
10579 }
10580 \cs_new_nopar:Npn \fp_to_tl_small_zeros:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10581 {
10582     \if_int_compare:w #9 = \c_zero
10583     \if_int_compare:w #8 = \c_zero
10584     \if_int_compare:w #7 = \c_zero
10585     \if_int_compare:w #6 = \c_zero
10586     \if_int_compare:w #5 = \c_zero
10587     \if_int_compare:w #4 = \c_zero
10588     \if_int_compare:w #3 = \c_zero
10589     \if_int_compare:w #2 = \c_zero
10590     \if_int_compare:w #1 = \c_zero
10591     \else:
10592         #1
10593     \fi:
10594     \else:
10595         #1#2
10596     \fi:
10597     \else:
10598         #1#2#3
10599     \fi:
10600     \else:
10601         #1#2#3#4
10602     \fi:
10603     \else:
10604         #1#2#3#4#5
10605     \fi:
10606     \else:
10607         #1#2#3#4#5#6
10608     \fi:
10609     \else:
10610         #1#2#3#4#5#6#7
10611     \fi:
10612     \else:
10613         #1#2#3#4#5#6#7#8
10614     \fi:
10615     \else:
10616         #1#2#3#4#5#6#7#8#9
10617     \fi:
10618 }

```

Some quick “return a few” functions.

```

10619 \cs_new_nopar:Npn \fp_use_iix_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#8#9}
10620 \cs_new_nopar:Npn \fp_use_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#9}
10621 \cs_new_nopar:Npn \fp_use_i_to_vii:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10622     {#1#2#3#4#5#6#7}
10623 \cs_new_nopar:Npn \fp_use_i_to_iix:NNNNNNNNN #1#2#3#4#5#6#7#8#9

```

```
10624 {#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 ??.)

201.7 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
10625 \cs_new_protected_nopar:Npn \fp_round_figures:Nn
10626 { \fp_round_figures_aux:NNn \tl_set:Nn }
10627 \cs_generate_variant:Nn \fp_round_figures:Nn { c }
10628 \cs_new_protected_nopar:Npn \fp_ground_figures:Nn
10629 { \fp_round_figures_aux:NNn \tl_gset:Nn }
10630 \cs_generate_variant:Nn \fp_ground_figures:Nn { c }
10631 \cs_new_protected_nopar:Npn \fp_round_figures_aux:NNn #1#2#3
10632 {
10633   \group_begin:
10634   \fp_read:N #2
10635   \int_set:Nn \l_fp_round_target_int { #3 - 1 }
10636   \if_int_compare:w \l_fp_round_target_int < \c_ten
10637     \exp_after:wN \fp_round:
10638   \fi:
10639   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10640   \cs_set_protected_nopar:Npx \fp_tmp:w
10641   {
10642     \group_end:
10643     #1 \exp_not:N #2
10644     {
10645       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10646         -
10647       \else:
10648         +
10649       \fi:
10650       \int_use:N \l_fp_input_a_integer_int
10651       .
10652       \exp_after:wN \use_none:n
10653       \int_use:N \l_fp_input_a_decimal_int
10654       e
10655       \int_use:N \l_fp_input_a_exponent_int
10656     }
10657   }
10658   \fp_tmp:w
10659 }
```

(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`

```

10660 \cs_new_protected_nopar:Npn \fp_round_places:Nn
10661 { \fp_round_places_aux:NNn \tl_set:Nn }
10662 \cs_generate_variant:Nn \fp_round_places:Nn { c }
10663 \cs_new_protected_nopar:Npn \fp_ground_places:Nn
10664 { \fp_round_places_aux:NNn \tl_gset:Nn }
10665 \cs_generate_variant:Nn \fp_ground_places:Nn { c }
10666 \cs_new_protected_nopar:Npn \fp_round_places_aux:NNn #1#2#3
10667 {
10668   \group_begin:
10669   \fp_read:N #2
10670   \int_set:Nn \l_fp_round_target_int
10671     { #3 + \l_fp_input_a_exponent_int }
10672   \if_int_compare:w \l_fp_round_target_int < \c_ten
10673     \exp_after:wN \fp_round:
10674   \fi:
10675   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10676   \cs_set_protected_nopar:Npx \fp_tmp:w
10677   {
10678     \group_end:
10679     #1 \exp_not:N #2
10680     {
10681       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10682         -
10683       \else:
10684         +
10685       \fi:
10686       \int_use:N \l_fp_input_a_integer_int
10687       .
10688       \exp_after:wN \use_none:n
10689       \int_use:N \l_fp_input_a_decimal_int
10690       e
10691       \int_use:N \l_fp_input_a_exponent_int
10692     }
10693   }
10694   \fp_tmp:w
10695 }

```

(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.

```

10696 \cs_new_protected_nopar:Npn \fp_round:
10697 {
10698   \bool_set_false:N \l_fp_round_carry_bool
10699   \l_fp_round_position_int \c_eight

```

```

10700 \tl_clear:N \l_fp_round_decimal_tl
10701 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10702 \exp_after:wN \use_i:nn \exp_after:wN
10703 \fp_round_aux:NNNNNNNN \int_use:N \l_fp_input_a_decimal_int
10704 }
10705 \cs_new_protected_nopar:Npn \fp_round_aux:NNNNNNNN #1#2#3#4#5#6#7#8#9
10706 {
10707 \fp_round_loop:N #9#8#7#6#5#4#3#2#1
10708 \bool_if:NT \l_fp_round_carry_bool
10709 { \tex_advance:D \l_fp_input_a_integer_int \c_one }
10710 \l_fp_input_a_decimal_int \l_fp_round_decimal_tl \scan_stop:
10711 \if_int_compare:w \l_fp_input_a_integer_int < \c_ten
10712 \else:
10713 \l_fp_input_a_integer_int \c_one
10714 \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10715 \tex_advance:D \l_fp_input_a_exponent_int \c_one
10716 \fi:
10717 }
10718 \cs_new_protected_nopar:Npn \fp_round_loop:N #1
10719 {
10720 \if_int_compare:w \l_fp_round_position_int < \l_fp_round_target_int
10721 \bool_if:NTF \l_fp_round_carry_bool
10722 { \l_fp_tmp_int \int_eval:w #1 + \c_one \scan_stop: }
10723 { \l_fp_tmp_int \int_eval:w #1 \scan_stop: }
10724 \if_int_compare:w \l_fp_tmp_int = \c_ten
10725 \l_fp_tmp_int \c_zero
10726 \else:
10727 \bool_set_false:N \l_fp_round_carry_bool
10728 \fi:
10729 \tl_set:Nx \l_fp_round_decimal_tl
10730 { \int_use:N \l_fp_tmp_int \l_fp_round_decimal_tl }
10731 \else:
10732 \tl_set:Nx \l_fp_round_decimal_tl { 0 \l_fp_round_decimal_tl }
10733 \if_int_compare:w \l_fp_round_position_int = \l_fp_round_target_int
10734 \if_int_compare:w #1 > \c_four
10735 \bool_set_true:N \l_fp_round_carry_bool
10736 \fi:
10737 \fi:
10738 \fi:
10739 \tex_advance:D \l_fp_round_position_int \c_minus_one
10740 \if_int_compare:w \l_fp_round_position_int > \c_minus_one
10741 \exp_after:wN \fp_round_loop:N
10742 \fi:
10743 }

```

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

201.8 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

```

10744 \cs_new_protected_nopar:Npn \fp_abs:N { \fp_abs_aux:NN \tl_set:Nn }
10745 \cs_new_protected_nopar:Npn \fp_gabs:N { \fp_abs_aux:NN \tl_gset:Nn }
10746 \cs_generate_variant:Nn \fp_abs:N { c }
10747 \cs_generate_variant:Nn \fp_gabs:N { c }
10748 \cs_new_protected_nopar:Npn \fp_abs_aux:NN #1#2
10749 {
10750   \group_begin:
10751   \fp_read:N #2
10752   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10753   \cs_set_protected_nopar:Npx \fp_tmp:w
10754   {
10755     \group_end:
10756     #1 \exp_not:N #2
10757     {
10758       +
10759       \int_use:N \l_fp_input_a_integer_int
10760       .
10761       \exp_after:wN \use_none:n
10762       \int_use:N \l_fp_input_a_decimal_int
10763       e
10764       \int_use:N \l_fp_input_a_exponent_int
10765     }
10766   }
10767   \fp_tmp:w
10768 }

```

(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 10769 \cs_new_protected_nopar:Npn \fp_neg:N { \fp_neg_aux:NN \tl_set:Nn }
\fp_gneg:N 10770 \cs_new_protected_nopar:Npn \fp_gneg:N { \fp_neg_aux:NN \tl_gset:Nn }
\fp_gneg:c 10771 \cs_generate_variant:Nn \fp_neg:N { c }
\fp_neg:NN 10772 \cs_generate_variant:Nn \fp_gneg:N { c }
10773 \cs_new_protected_nopar:Npn \fp_neg_aux:NN #1#2
10774 {
10775   \group_begin:
10776   \fp_read:N #2
10777   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10778   \tl_set:Nx \l_fp_tmp_tl
10779   {
10780     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10781     +
10782     \else:
10783     -
10784     \fi:
10785     \int_use:N \l_fp_input_a_integer_int
10786     .
10787     \exp_after:wN \use_none:n
10788     \int_use:N \l_fp_input_a_decimal_int
10789     e
10790     \int_use:N \l_fp_input_a_exponent_int

```

```

10791     }
10792     \exp_after:wN \group_end: \exp_after:wN
10793     #1 \exp_after:wN #2 \exp_after:wN { \l_fp_tmp_tl }
10794 }

```

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

201.9 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`
`\fp_gadd:cn`
`\fp_add_aux:NNn`
`\fp_add_core:`
`\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.

```

10799 \cs_new_protected_nopar:Npn \fp_add_aux:NNn #1#2#3
10800 {
10801   \group_begin:
10802   \fp_read:N #2
10803   \fp_split:Nn b {#3}
10804   \fp_standardise:NNNN
10805   \l_fp_input_b_sign_int
10806   \l_fp_input_b_integer_int
10807   \l_fp_input_b_decimal_int
10808   \l_fp_input_b_exponent_int
10809   \fp_add_core:
10810   \fp_tmp:w #1#2
10811 }
10812 \cs_new_protected_nopar:Npn \fp_add_core:
10813 {
10814   \fp_level_input_exponents:
10815   \if_int_compare:w
10816     \int_eval:w
10817       \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
10818       > \c_zero
10819     \exp_after:wN \fp_add_sum:
10820   \else:
10821     \exp_after:wN \fp_add_difference:
10822   \fi:
10823   \l_fp_output_exponent_int \l_fp_input_a_exponent_int
10824   \fp_standardise:NNNN
10825   \l_fp_output_sign_int
10826   \l_fp_output_integer_int
10827   \l_fp_output_decimal_int
10828   \l_fp_output_exponent_int
10829   \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2

```

```

10830 {
10831   \group_end:
10832   ##1 ##2
10833   {
10834     \if_int_compare:w \l_fp_output_sign_int < \c_zero
10835     -
10836     \else:
10837     +
10838     \fi:
10839     \int_use:N \l_fp_output_integer_int
10840     .
10841     \exp_after:wN \use_none:n
10842     \int_value:w \int_eval:w
10843     \l_fp_output_decimal_int + \c_one_thousand_million
10844     e
10845     \int_use:N \l_fp_output_exponent_int
10846   }
10847 }
10848 }

```

Finding the sum of two numbers is trivially easy.

```

10849 \cs_new_protected_nopar:Npn \fp_add_sum:
10850 {
10851   \l_fp_output_sign_int \l_fp_input_a_sign_int
10852   \l_fp_output_integer_int
10853   \int_eval:w
10854   \l_fp_input_a_integer_int + \l_fp_input_b_integer_int
10855   \scan_stop:
10856   \l_fp_output_decimal_int
10857   \int_eval:w
10858   \l_fp_input_a_decimal_int + \l_fp_input_b_decimal_int
10859   \scan_stop:
10860   \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
10861   \else:
10862     \tex_advance:D \l_fp_output_integer_int \c_one
10863     \tex_advance:D \l_fp_output_decimal_int -\c_one_thousand_million
10864   \fi:
10865 }

```

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.

```

10866 \cs_new_protected_nopar:Npn \fp_add_difference:
10867 {
10868   \l_fp_output_integer_int
10869   \int_eval:w
10870   \l_fp_input_a_integer_int - \l_fp_input_b_integer_int
10871   \scan_stop:
10872   \l_fp_output_decimal_int

```

```

10873     \int_eval:w
10874     \l_fp_input_a_decimal_int - \l_fp_input_b_decimal_int
10875     \scan_stop:
10876     \if_int_compare:w \l_fp_output_decimal_int < \c_zero
10877     \tex_advance:D \l_fp_output_integer_int \c_minus_one
10878     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
10879     \fi:
10880     \if_int_compare:w \l_fp_output_integer_int < \c_zero
10881     \l_fp_output_sign_int \l_fp_input_b_sign_int
10882     \if_int_compare:w \l_fp_output_decimal_int = \c_zero
10883     \l_fp_output_integer_int -\l_fp_output_integer_int
10884     \else:
10885     \l_fp_output_decimal_int
10886     \int_eval:w
10887     \c_one_thousand_million - \l_fp_output_decimal_int
10888     \scan_stop:
10889     \l_fp_output_integer_int
10890     \int_eval:w
10891     - \l_fp_output_integer_int - \c_one
10892     \scan_stop:
10893     \fi:
10894     \else:
10895     \l_fp_output_sign_int \l_fp_input_a_sign_int
10896     \fi:
10897 }

```

(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 10898 \cs_new_protected_nopar:Npn \fp_sub:Nn { \fp_sub_aux:NNn \tl_set:Nn }
10899 \cs_new_protected_nopar:Npn \fp_gsub:Nn { \fp_sub_aux:NNn \tl_gset:Nn }
10900 \cs_generate_variant:Nn \fp_sub:Nn { c }
10901 \cs_generate_variant:Nn \fp_gsub:Nn { c }
10902 \cs_new_protected_nopar:Npn \fp_sub_aux:NNn #1#2#3
10903 {
10904   \group_begin:
10905   \fp_read:N #2
10906   \fp_split:Nn b {#3}
10907   \fp_standardise:NNNN
10908   \l_fp_input_b_sign_int
10909   \l_fp_input_b_integer_int
10910   \l_fp_input_b_decimal_int
10911   \l_fp_input_b_exponent_int
10912   \tex_multiply:D \l_fp_input_b_sign_int \c_minus_one
10913   \fp_add_core:
10914   \fp_tmp:w #1#2
10915 }

```

(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	<p>The pattern is much the same for multiplication.</p> <pre> 10916 \cs_new_protected_nopar:Npn \fp_mul:Nn { \fp_mul_aux:NNn \tl_set:Nn } 10917 \cs_new_protected_nopar:Npn \fp_gmul:Nn { \fp_mul_aux:NNn \tl_gset:Nn } 10918 \cs_generate_variant:Nn \fp_mul:Nn { c } 10919 \cs_generate_variant:Nn \fp_gmul:Nn { c } </pre> <p>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</p> <pre> 10920 \cs_new_protected_nopar:Npn \fp_mul_aux:NNn #1#2#3 10921 { 10922 \group_begin: 10923 \fp_read:N #2 10924 \fp_split:Nn b {#3} 10925 \fp_standardise:NNNN 10926 \l_fp_input_b_sign_int 10927 \l_fp_input_b_integer_int 10928 \l_fp_input_b_decimal_int 10929 \l_fp_input_b_exponent_int 10930 \fp_mul_internal: 10931 \l_fp_output_exponent_int 10932 \int_eval:w 10933 \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int 10934 \scan_stop: 10935 \fp_standardise:NNNN 10936 \l_fp_output_sign_int 10937 \l_fp_output_integer_int 10938 \l_fp_output_decimal_int 10939 \l_fp_output_exponent_int 10940 \cs_set_protected_nopar:Npx \fp_tmp:w 10941 { 10942 \group_end: 10943 #1 \exp_not:N #2 10944 { 10945 \if_int_compare:w 10946 \int_eval:w 10947 \l_fp_input_a_sign_int * \l_fp_input_b_sign_int 10948 < \c_zero 10949 \if_int_compare:w 10950 \int_eval:w 10951 \l_fp_output_integer_int + \l_fp_output_decimal_int 10952 = \c_zero 10953 + 10954 \else: 10955 - 10956 \fi: 10957 \else: 10958 + 10959 \fi: </pre>
---	--

```

10960         \int_use:N \l_fp_output_integer_int
10961         .
10962         \exp_after:wN \use_none:n
10963         \int_value:w \int_eval:w
10964         \l_fp_output_decimal_int + \c_one_thousand_million
10965         e
10966         \int_use:N \l_fp_output_exponent_int
10967     }
10968 }
10969 \fp_tmp:w
10970 }

```

Done separately so that the internal use is a bit easier.

```

10971 \cs_new_protected_nopar:Npn \fp_mul_internal:
10972 {
10973     \fp_mul_split:NNNN \l_fp_input_a_decimal_int
10974     \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
10975     \fp_mul_split:NNNN \l_fp_input_b_decimal_int
10976     \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
10977     \l_fp_mul_output_int \c_zero
10978     \tl_clear:N \l_fp_mul_output_tl
10979     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_iii_int
10980     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_ii_int
10981     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_i_int
10982     \tex_divide:D \l_fp_mul_output_int \c_one_thousand
10983     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_iii_int
10984     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_ii_int
10985     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_i_int
10986     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_input_b_integer_int
10987     \fp_mul_end_level:
10988     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_ii_int
10989     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_i_int
10990     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_input_b_integer_int
10991     \fp_mul_end_level:
10992     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_i_int
10993     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_input_b_integer_int
10994     \fp_mul_end_level:
10995     \l_fp_output_decimal_int 0 \l_fp_mul_output_tl \scan_stop:
10996     \tl_clear:N \l_fp_mul_output_tl
10997     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_input_b_integer_int
10998     \fp_mul_end_level:
10999     \l_fp_output_integer_int 0 \l_fp_mul_output_tl \scan_stop:
11000 }

```

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.

```

11001 \cs_new_protected_nopar:Npn \fp_mul_split:NNNN #1#2#3#4
11002 {

```



```

11003 \tex_advance:D #1 \c_one_thousand_million
11004 \cs_set_protected_nopar:Npn \fp_mul_split_aux:w
11005   ##1##2##3##4##5##6##7##8##9 \q_stop {
11006     #2 ##2##3##4 \scan_stop:
11007     #3 ##5##6##7 \scan_stop:
11008     #4 ##8##9 \scan_stop:
11009   }
11010 \exp_after:wN \fp_mul_split_aux:w \int_use:N #1 \q_stop
11011 \tex_advance:D #1 -\c_one_thousand_million
11012 }
11013 \cs_new_protected_nopar:Npn \fp_mul_product:NN #1#2
11014 {
11015   \l_fp_mul_output_int
11016   \int_eval:w \l_fp_mul_output_int + #1 * #2 \scan_stop:
11017 }

```

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.

```

11018 \cs_new_protected_nopar:Npn \fp_mul_end_level:
11019 {
11020   \tex_advance:D \l_fp_mul_output_int \c_one_thousand_million
11021   \exp_after:wN \use_i:nn \exp_after:wN
11022   \fp_mul_end_level:NNNNNNNN \int_use:N \l_fp_mul_output_int
11023 }
11024 \cs_new_protected_nopar:Npn \fp_mul_end_level:NNNNNNNN #1#2#3#4#5#6#7#8#9
11025 {
11026   \tl_set:Nx \l_fp_mul_output_tl { #7#8#9 \l_fp_mul_output_tl }
11027   \l_fp_mul_output_int #1#2#3#4#5#6 \scan_stop:
11028 }

```

(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 11029 \cs_new_protected_nopar:Npn \fp_div:Nn { \fp_div_aux:NNn \tl_set:Nn }
\fp_gdiv:Nn 11030 \cs_new_protected_nopar:Npn \fp_gdiv:Nn { \fp_div_aux:NNn \tl_gset:Nn }
\fp_gdiv:cn 11031 \cs_generate_variant:Nn \fp_div:Nn { c }
\fp_div_aux:NNn 11032 \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: 11033 \cs_new_protected_nopar:Npn \fp_div_aux:NNn #1#2#3
\fp_div_store: 11034 {
\fp_div_store_integer: 11035   \group_begin:
\fp_div_store_decimal: 11036     \fp_read:N #2
11037     \fp_split:Nn b {#3}
11038     \fp_standardise:NNNN
11039     \l_fp_input_b_sign_int
11040     \l_fp_input_b_integer_int
11041     \l_fp_input_b_decimal_int

```

```

11042         \l_fp_input_b_exponent_int
11043     \if_int_compare:w
11044         \int_eval:w
11045         \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
11046         = \c_zero
11047     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
11048     {
11049         \group_end:
11050         #1 \exp_not:N #2 { \c_undefined_fp }
11051     }
11052 \else:
11053     \if_int_compare:w
11054         \int_eval:w
11055         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11056         = \c_zero
11057     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
11058     {
11059         \group_end:
11060         #1 \exp_not:N #2 { \c_zero_fp }
11061     }
11062 \else:
11063     \exp_after:wN \exp_after:wN \exp_after:wN \fp_div_internal:
11064 \fi:
11065 \fi:
11066 \fp_tmp:w #1#2
11067 }

```

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.

```

11068 \cs_new_protected_nopar:Npn \fp_div_internal: {
11069     \l_fp_output_integer_int \c_zero
11070     \l_fp_output_decimal_int \c_zero
11071     \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
11072     \l_fp_div_offset_int \c_one_hundred_million
11073     \fp_div_loop:
11074     \l_fp_output_exponent_int
11075     \int_eval:w
11076     \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
11077     \scan_stop:
11078     \fp_standardise:NNNN
11079     \l_fp_output_sign_int
11080     \l_fp_output_integer_int
11081     \l_fp_output_decimal_int
11082     \l_fp_output_exponent_int
11083     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
11084     {
11085         \group_end:

```

```

11086     ##1 ##2
11087     {
11088         \if_int_compare:w
11089             \int_eval:w
11090             \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11091             < \c_zero
11092         \if_int_compare:w
11093             \int_eval:w
11094             \l_fp_output_integer_int + \l_fp_output_decimal_int
11095             = \c_zero
11096         +
11097         \else:
11098             -
11099         \fi:
11100     \else:
11101         +
11102         \fi:
11103         \int_use:N \l_fp_output_integer_int
11104         .
11105         \exp_after:wN \use_none:n
11106         \int_value:w \int_eval:w
11107         \l_fp_output_decimal_int + \c_one_thousand_million
11108         \int_eval_end:
11109     e
11110     \int_use:N \l_fp_output_exponent_int
11111 }
11112 }
11113 }

```

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.

```

11114 \cs_new_protected_nopar:Npn \fp_div_loop:
11115 {
11116     \l_fp_count_int \c_zero
11117     \fp_div_divide:
11118     \fp_div_store:
11119     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11120     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11121     \exp_after:wN \fp_div_loop_step:w
11122     \int_use:N \l_fp_input_a_decimal_int \q_stop
11123     \if_int_compare:w
11124         \int_eval:w \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11125         > \c_zero
11126     \if_int_compare:w \l_fp_div_offset_int > \c_zero
11127         \exp_after:wN \exp_after:wN \exp_after:wN
11128         \fp_div_loop:
11129     \fi:
11130 \fi:
11131 }

```

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.

```

11132 \cs_new_protected_nopar:Npn \fp_div_divide:
11133 {
11134   \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
11135     \exp_after:wN \fp_div_divide_aux:
11136   \else:
11137     \if_int_compare:w \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
11138     \else:
11139       \if_int_compare:w
11140         \l_fp_input_a_decimal_int < \l_fp_input_b_decimal_int
11141       \else:
11142         \exp_after:wN \exp_after:wN \exp_after:wN
11143         \exp_after:wN \exp_after:wN \exp_after:wN
11144         \exp_after:wN \fp_div_divide_aux:
11145       \fi:
11146     \fi:
11147   \fi:
11148 }
11149 \cs_new_protected_nopar:Npn \fp_div_divide_aux:
11150 {
11151   \tex_advance:D \l_fp_count_int \c_one
11152   \tex_advance:D \l_fp_input_a_integer_int -\l_fp_input_b_integer_int
11153   \tex_advance:D \l_fp_input_a_decimal_int -\l_fp_input_b_decimal_int
11154   \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11155     \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11156     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11157   \fi:
11158   \fp_div_divide:
11159 }

```

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.

```

11160 \cs_new_protected_nopar:Npn \fp_div_store: { }
11161 \cs_new_protected_nopar:Npn \fp_div_store_integer:
11162 {
11163   \l_fp_output_integer_int \l_fp_count_int
11164   \cs_set_eq:NN \fp_div_store: \fp_div_store_decimal:
11165 }
11166 \cs_new_protected_nopar:Npn \fp_div_store_decimal:
11167 {
11168   \l_fp_output_decimal_int
11169   \int_eval:w
11170     \l_fp_output_decimal_int +
11171     \l_fp_count_int * \l_fp_div_offset_int
11172   \int_eval_end:
11173   \tex_divide:D \l_fp_div_offset_int \c_ten
11174 }

```

```

11175 \cs_new_protected_nopar:Npn \fp_div_loop_step:w #1#2#3#4#5#6#7#8#9 \q_stop
11176 {
11177   \l_fp_input_a_integer_int
11178   \int_eval:w #2 + \l_fp_input_a_integer_int \int_eval_end:
11179   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11180 }

```

(End definition for `\fp_div:Nn` and `\fp_div:cn`. These functions are documented on page ??.)

201.10 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.

```

11181 \cs_new_protected_nopar:Npn \fp_add:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11182 {
11183   #7 \int_eval:w #1 + #4 \int_eval_end:
11184   #8 \int_eval:w #2 + #5 \int_eval_end:
11185   #9 \int_eval:w #3 + #6 \int_eval_end:
11186   \if_int_compare:w #9 < \c_one_thousand_million
11187   \else:
11188     \tex_advance:D #8 \c_one
11189     \tex_advance:D #9 -\c_one_thousand_million
11190   \fi:
11191   \if_int_compare:w #8 < \c_one_thousand_million
11192   \else:
11193     \tex_advance:D #7 \c_one

```

```

11194     \tex_advance:D #8 -\c_one_thousand_million
11195     \fi:
11196   }
      (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.

```

11197 \cs_new_protected_nopar:Npn \fp_sub:NNNNNNNN #1#2#3#4#5#6#7#8#9
11198 {
11199   #7 \int_eval:w #1 - #4 \int_eval_end:
11200   #8 \int_eval:w #2 - #5 \int_eval_end:
11201   #9 \int_eval:w #3 - #6 \int_eval_end:
11202   \if_int_compare:w #9 < \c_zero
11203     \tex_advance:D #8 \c_minus_one
11204     \tex_advance:D #9 \c_one_thousand_million
11205   \fi:
11206   \if_int_compare:w #8 < \c_zero
11207     \tex_advance:D #7 \c_minus_one
11208     \tex_advance:D #8 \c_one_thousand_million
11209   \fi:
11210   \if_int_compare:w #7 < \c_zero
11211     \if_int_compare:w \int_eval:w #8 + #9 = \c_zero
11212       #7 -#7
11213   \else:
11214     \tex_advance:D #7 \c_one
11215     #8 \int_eval:w \c_one_thousand_million - #8 \int_eval_end:
11216     #9 \int_eval:w \c_one_thousand_million - #9 \int_eval_end:
11217   \fi:
11218   \fi:
11219 }
      (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.

```

11220 \cs_new_protected_nopar:Npn \fp_mul:NNNNNN #1#2#3#4#5#6
11221 {
11222   \fp_mul_split:NNNN #1
11223     \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11224   \fp_mul_split:NNNN #2
11225     \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11226   \fp_mul_split:NNNN #3
11227     \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11228   \fp_mul_split:NNNN #4
11229     \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11230   \l_fp_mul_output_int \c_zero
11231   \tl_clear:N \l_fp_mul_output_tl
11232   \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_vi_int
11233   \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_v_int
11234   \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_iv_int

```

```

11235 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int
11236 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11237 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11238 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11239 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11240 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11241 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11242 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11243 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11244 \fp_mul_end_level:
11245 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11246 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11247 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11248 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11249 \fp_mul_end_level:
11250 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11251 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11252 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11253 \fp_mul_end_level:
11254 #6 0 \l_fp_mul_output_tl \scan_stop:
11255 \tl_clear:N \l_fp_mul_output_tl
11256 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11257 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11258 \fp_mul_end_level:
11259 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11260 \fp_mul_end_level:
11261 \fp_mul_end_level:
11262 #5 0 \l_fp_mul_output_tl \scan_stop:
11263 }

```

(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.

```

11264 \cs_new_protected_nopar:Npn \fp_mul:NNNNNNNN #1#2#3#4#5#6#7#8#9
11265 {
11266   \fp_mul_split:NNNN #2
11267   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11268   \fp_mul_split:NNNN #3
11269   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11270   \fp_mul_split:NNNN #5
11271   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11272   \fp_mul_split:NNNN #6
11273   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11274   \l_fp_mul_output_int \c_zero
11275   \tl_clear:N \l_fp_mul_output_tl
11276   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_vi_int
11277   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_v_int
11278   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iv_int
11279   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int

```

```

11280 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11281 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11282 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11283 \fp_mul_product:NN #1 \l_fp_mul_b_vi_int
11284 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11285 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11286 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11287 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11288 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11289 \fp_mul_product:NN \l_fp_mul_a_vi_int #4
11290 \fp_mul_end_level:
11291 \fp_mul_product:NN #1 \l_fp_mul_b_v_int
11292 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11293 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11294 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11295 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11296 \fp_mul_product:NN \l_fp_mul_a_v_int #4
11297 \fp_mul_end_level:
11298 \fp_mul_product:NN #1 \l_fp_mul_b_iv_int
11299 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11300 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11301 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11302 \fp_mul_product:NN \l_fp_mul_a_iv_int #4
11303 \fp_mul_end_level:
11304 #9 0 \l_fp_mul_output_tl \scan_stop:
11305 \tl_clear:N \l_fp_mul_output_tl
11306 \fp_mul_product:NN #1 \l_fp_mul_b_iii_int
11307 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11308 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11309 \fp_mul_product:NN \l_fp_mul_a_iii_int #4
11310 \fp_mul_end_level:
11311 \fp_mul_product:NN #1 \l_fp_mul_b_ii_int
11312 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11313 \fp_mul_product:NN \l_fp_mul_a_ii_int #4
11314 \fp_mul_end_level:
11315 \fp_mul_product:NN #1 \l_fp_mul_b_i_int
11316 \fp_mul_product:NN \l_fp_mul_a_i_int #4
11317 \fp_mul_end_level:
11318 #8 0 \l_fp_mul_output_tl \scan_stop:
11319 \tl_clear:N \l_fp_mul_output_tl
11320 \fp_mul_product:NN #1 #4
11321 \fp_mul_end_level:
11322 #7 0 \l_fp_mul_output_tl \scan_stop:
11323 }

```

(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.

```

11324 \cs_new_protected_nopar:Npn \fp_div_integer:NNNNN #1#2#3#4#5
11325 {
11326   \l_fp_tmp_int #1
11327   \tex_divide:D \l_fp_tmp_int #3
11328   \l_fp_tmp_int \int_eval:w #1 - \l_fp_tmp_int * #3 \int_eval_end:
11329   #4 #1
11330   \tex_divide:D #4 #3
11331   #5 #2
11332   \tex_divide:D #5 #3
11333   \tex_multiply:D \l_fp_tmp_int \c_one_thousand
11334   \tex_divide:D \l_fp_tmp_int #3
11335   #5 \int_eval:w #5 + \l_fp_tmp_int * \c_one_million \int_eval_end:
11336   \if_int_compare:w #5 > \c_one_thousand_million
11337     \tex_advance:D #4 \c_one
11338     \tex_advance:D #5 -\c_one_thousand_million
11339   \fi:
11340 }

```

(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
11341 \cs_new_protected_nopar:Npn \fp_extended_normalise:
11342 {
11343   \fp_extended_normalise_aux_i:
11344   \fp_extended_normalise_aux_ii:
11345 }
11346 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:
11347 {
11348   \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
11349     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11350     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11351     \exp_after:wN \fp_extended_normalise_aux_i:w
11352     \int_use:N \l_fp_input_a_decimal_int \q_stop
11353     \exp_after:wN \fp_extended_normalise_aux_i:
11354   \fi:
11355 }
11356 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:w
11357 #1#2#3#4#5#6#7#8#9 \q_stop
11358 {
11359   \l_fp_input_a_integer_int
11360   \int_eval:w \l_fp_input_a_integer_int + #2 \scan_stop:
11361   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11362   \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11363   \exp_after:wN \fp_extended_normalise_aux_ii:w
11364   \int_use:N \l_fp_input_a_extended_int \q_stop
11365 }

```

```

11366 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:w
11367 #1#2#3#4#5#6#7#8#9 \q_stop
11368 {
11369   \l_fp_input_a_decimal_int
11370   \int_eval:w \l_fp_input_a_decimal_int + #2 \scan_stop:
11371   \l_fp_input_a_extended_int #3#4#5#6#7#8#9 0 \scan_stop:
11372   \tex_advance:D \l_fp_input_a_exponent_int \c_minus_one
11373 }
11374 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:
11375 {
11376   \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
11377   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11378   \exp_after:wN \use_i:nn \exp_after:wN
11379   \fp_extended_normalise_ii_aux:NNNNNNNNN
11380   \int_use:N \l_fp_input_a_decimal_int
11381   \exp_after:wN \fp_extended_normalise_aux_ii:
11382   \fi:
11383 }
11384 \cs_new_protected_nopar:Npn \fp_extended_normalise_ii_aux:NNNNNNNNN
11385 #1#2#3#4#5#6#7#8#9
11386 {
11387   \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11388   \l_fp_input_a_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
11389   \else:
11390     \tl_set:Nx \l_fp_tmp_tl
11391     {
11392       \int_use:N \l_fp_input_a_integer_int
11393       #1#2#3#4#5#6#7#8
11394     }
11395     \l_fp_input_a_integer_int \c_zero
11396     \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
11397   \fi:
11398   \tex_divide:D \l_fp_input_a_extended_int \c_ten
11399   \tl_set:Nx \l_fp_tmp_tl
11400   {
11401     #9
11402     \int_use:N \l_fp_input_a_extended_int
11403   }
11404   \l_fp_input_a_extended_int \l_fp_tmp_tl \scan_stop:
11405   \tex_advance:D \l_fp_input_a_exponent_int \c_one
11406 }

```

(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
11407 \cs_new_protected_nopar:Npn \fp_extended_normalise_output:
11408 {
11409   \if_int_compare:w \l_fp_output_integer_int > \c_nine

```

```

11410 \tex_advance:D \l_fp_output_integer_int \c_one_thousand_million
11411 \exp_after:wN \use_i:nn \exp_after:wN
11412 \fp_extended_normalise_output_aux_i:NNNNNNNNN
11413 \int_use:N \l_fp_output_integer_int
11414 \exp_after:wN \fp_extended_normalise_output:
11415 \fi:
11416 }
11417 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux_i:NNNNNNNNN
11418 #1#2#3#4#5#6#7#8#9
11419 {
11420 \l_fp_output_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
11421 \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11422 \tl_set:Nx \l_fp_tmp_tl
11423 {
11424 #9
11425 \exp_after:wN \use_none:n
11426 \int_use:N \l_fp_output_decimal_int
11427 }
11428 \exp_after:wN \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11429 \l_fp_tmp_tl
11430 }
11431 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11432 #1#2#3#4#5#6#7#8#9
11433 {
11434 \l_fp_output_decimal_int #1#2#3#4#5#6#7#8#9 \scan_stop:
11435 \fp_extended_normalise_output_aux:N
11436 }
11437 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux:N #1
11438 {
11439 \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11440 \tex_divide:D \l_fp_output_extended_int \c_ten
11441 \tl_set:Nx \l_fp_tmp_tl
11442 {
11443 #1
11444 \exp_after:wN \use_none:n
11445 \int_use:N \l_fp_output_extended_int
11446 }
11447 \l_fp_output_extended_int \l_fp_tmp_tl \scan_stop:
11448 \tex_advance:D \l_fp_output_exponent_int \c_one
11449 }

```

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

201.11 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 $-\pi < x \leq \pi$.

```

11450 \cs_new_protected_nopar:Npn \fp_trig_normalise:
11451 {

```

```

11452 \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11453 \l_fp_input_a_extended_int \c_zero
11454 \fp_extended_normalise:
11455 \fp_trig_normalise_aux:
11456 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11457 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11458 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11459 \fi:
11460 \exp_after:wN \fp_trig_octant:
11461 \else:
11462 \l_fp_input_a_sign_int \c_one
11463 \l_fp_output_integer_int \c_zero
11464 \l_fp_output_decimal_int \c_zero
11465 \l_fp_output_exponent_int \c_zero
11466 \exp_after:wN \fp_trig_overflow_msg:
11467 \fi:
11468 }
11469 \cs_new_protected_nopar:Npn \fp_trig_normalise_aux:
11470 {
11471 \if_int_compare:w \l_fp_input_a_integer_int > \c_three
11472 \fp_trig_sub:NNN
11473 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11474 \exp_after:wN \fp_trig_normalise_aux:
11475 \else:
11476 \if_int_compare:w \l_fp_input_a_integer_int > \c_two
11477 \if_int_compare:w \l_fp_input_a_decimal_int > \c_fp_pi_decimal_int
11478 \fp_trig_sub:NNN
11479 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11480 \exp_after:wN \exp_after:wN \exp_after:wN
11481 \exp_after:wN \exp_after:wN \exp_after:wN
11482 \exp_after:wN \fp_trig_normalise_aux:
11483 \fi:
11484 \fi:
11485 \fi:
11486 }

```

Here, there may be a sign change but there will never be any variation in the input. So a dedicated function can be used.

```

11487 \cs_new_protected_nopar:Npn \fp_trig_sub:NNN #1#2#3
11488 {
11489 \l_fp_input_a_integer_int
11490 \int_eval:w \l_fp_input_a_integer_int - #1 \int_eval_end:
11491 \l_fp_input_a_decimal_int
11492 \int_eval:w \l_fp_input_a_decimal_int - #2 \int_eval_end:
11493 \l_fp_input_a_extended_int
11494 \int_eval:w \l_fp_input_a_extended_int - #3 \int_eval_end:
11495 \if_int_compare:w \l_fp_input_a_extended_int < \c_zero
11496 \tex_advance:D \l_fp_input_a_decimal_int \c_minus_one
11497 \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11498 \fi:

```

```

11499 \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11500 \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11501 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11502 \fi:
11503 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11504 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11505 \if_int_compare:w
11506 \int_eval:w
11507 \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
11508 = \c_zero
11509 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11510 \else:
11511 \l_fp_input_a_integer_int
11512 \int_eval:w
11513 - \l_fp_input_a_integer_int - \c_one
11514 \int_eval_end:
11515 \l_fp_input_a_decimal_int
11516 \int_eval:w
11517 \c_one_thousand_million - \l_fp_input_a_decimal_int
11518 \int_eval_end:
11519 \l_fp_input_a_extended_int
11520 \int_eval:w
11521 \c_one_thousand_million - \l_fp_input_a_extended_int
11522 \int_eval_end:
11523 \fi:
11524 \fi:
11525 }

```

(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!

```

11526 \cs_new_protected_nopar:Npn \fp_trig_octant:
11527 {
11528 \l_fp_trig_octant_int \c_one
11529 \fp_trig_octant_aux_i:
11530 \if_int_compare:w \l_fp_input_a_decimal_int < \c_ten
11531 \l_fp_input_a_decimal_int \c_zero
11532 \l_fp_input_a_extended_int \c_zero
11533 \fi:
11534 \if_int_odd:w \l_fp_trig_octant_int
11535 \else:
11536 \fp_sub:NNNNNNNNN
11537 \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11538 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11539 \l_fp_input_a_extended_int
11540 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int

```

```

11541         \l_fp_input_a_extended_int
11542     \fi:
11543 }
11544 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_i:
11545 {
11546     \if_int_compare:w \l_fp_trig_octant_int > \c_four
11547         \l_fp_trig_octant_int \c_four
11548         \l_fp_input_a_decimal_int \c_fp_pi_by_four_decimal_int
11549         \l_fp_input_a_extended_int \c_fp_pi_by_four_extended_int
11550     \else:
11551         \exp_after:wN \fp_trig_octant_aux_ii:
11552     \fi:
11553 }
11554 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_ii:
11555 {
11556     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
11557         \fp_sub:NNNNNNNNN
11558         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11559         \l_fp_input_a_extended_int
11560         \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11561         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11562         \l_fp_input_a_extended_int
11563         \tex_advance:D \l_fp_trig_octant_int \c_one
11564         \exp_after:wN \fp_trig_octant_aux_i:
11565     \else:
11566         \if_int_compare:w
11567             \l_fp_input_a_decimal_int > \c_fp_pi_by_four_decimal_int
11568         \fp_sub:NNNNNNNNN
11569             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11570             \l_fp_input_a_extended_int
11571             \c_zero \c_fp_pi_by_four_decimal_int
11572             \c_fp_pi_by_four_extended_int
11573             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11574             \l_fp_input_a_extended_int
11575             \tex_advance:D \l_fp_trig_octant_int \c_one
11576             \exp_after:wN \exp_after:wN \exp_after:wN
11577             \fp_trig_octant_aux_i:
11578         \fi:
11579     \fi:
11580 }

```

(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> 11581 \cs_new_protected_nopar:Npn \fp_sin:Nn { \fp_sin_aux:NNn \tl_set:Nn } 11582 \cs_new_protected_nopar:Npn \fp_gsin:Nn { \fp_sin_aux:NNn \tl_gset:Nn } 11583 \cs_generate_variant:Nn \fp_sin:Nn { c } 11584 \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} .

```

11585 \cs_new_protected_nopar:Npn \fp_sin_aux:NNn #1#2#3
11586 {
11587   \group_begin:
11588     \fp_split:Nn a {#3}
11589     \fp_standardise:NNNN
11590     \l_fp_input_a_sign_int
11591     \l_fp_input_a_integer_int
11592     \l_fp_input_a_decimal_int
11593     \l_fp_input_a_exponent_int
11594     \tl_set:Nx \l_fp_arg_tl
11595     {
11596       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11597       -
11598       \else:
11599       +
11600       \fi:
11601       \int_use:N \l_fp_input_a_integer_int
11602       .
11603       \exp_after:wN \use_none:n
11604       \int_value:w \int_eval:w
11605       \l_fp_input_a_decimal_int + \c_one_thousand_million
11606       e
11607       \int_use:N \l_fp_input_a_exponent_int
11608     }
11609     \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
11610     \cs_set_protected_nopar:Npx \fp_tmp:w
11611     {
11612       \group_end:
11613       #1 \exp_not:N #2 { \l_fp_arg_tl }
11614     }
11615     \else:
11616     \if_cs_exist:w
11617     c_fp_sin ( \l_fp_arg_tl ) _fp
11618     \cs_end:
11619     \else:
11620     \exp_after:wN \exp_after:wN \exp_after:wN
11621     \fp_sin_aux_i:
11622     \fi:
11623     \cs_set_protected_nopar:Npx \fp_tmp:w
11624     {
11625       \group_end:
11626       #1 \exp_not:N #2
11627       { \use:c { c_fp_sin ( \l_fp_arg_tl ) _fp } }
11628     }
11629     \fi:
11630     \fp_tmp:w

```

```
11631 }
```

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.

```
11632 \cs_new_protected_nopar:Npn \fp_sin_aux_i:
11633 {
11634   \fp_trig_normalise:
11635   \fp_sin_aux_ii:
11636   \if_int_compare:w \l_fp_output_integer_int = \c_one
11637     \l_fp_output_exponent_int \c_zero
11638   \else:
11639     \l_fp_output_integer_int \l_fp_output_decimal_int
11640     \l_fp_output_decimal_int \l_fp_output_extended_int
11641     \l_fp_output_exponent_int -\c_nine
11642   \fi:
11643   \fp_standardise:NNNN
11644     \l_fp_input_a_sign_int
11645     \l_fp_output_integer_int
11646     \l_fp_output_decimal_int
11647     \l_fp_output_exponent_int
11648   \tl_new:c { c_fp_sin ( \l_fp_arg_tl ) _fp }
11649   \tl_gset:cx { c_fp_sin ( \l_fp_arg_tl ) _fp }
11650   {
11651     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11652       +
11653     \else:
11654       -
11655     \fi:
11656     \int_use:N \l_fp_output_integer_int
11657     .
11658     \exp_after:wN \use_none:n
11659     \int_value:w \int_eval:w
11660       \l_fp_output_decimal_int + \c_one_thousand_million
11661     e
11662     \int_use:N \l_fp_output_exponent_int
11663   }
11664 }
11665 \cs_new_protected_nopar:Npn \fp_sin_aux_ii:
11666 {
11667   \if_case:w \l_fp_trig_octant_int
11668   \or:
11669     \exp_after:wN \fp_trig_calc_sin:
11670   \or:
11671     \exp_after:wN \fp_trig_calc_cos:
11672   \or:
11673     \exp_after:wN \fp_trig_calc_cos:
11674   \or:
11675     \exp_after:wN \fp_trig_calc_sin:
11676   \fi:
```



```

11677 }
      (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 11678 \cs_new_protected_nopar:Npn \fp_cos:Nn { \fp_cos_aux:NNn \tl_set:Nn }
\fp_gcos:Nn 11679 \cs_new_protected_nopar:Npn \fp_gcos:Nn { \fp_cos_aux:NNn \tl_gset:Nn }
\fp_gcos:cn 11680 \cs_generate_variant:Nn \fp_cos:Nn { c }
\fp_cos_aux:NNn 11681 \cs_generate_variant:Nn \fp_gcos:Nn { c }
\fp_cos_aux_i: 11682 \cs_new_protected_nopar:Npn \fp_cos_aux:NNn #1#2#3
\fp_cos_aux_ii: 11683 {
11684   \group_begin:
11685   \fp_split:Nn a {#3}
11686   \fp_standardise:NNNN
11687   \l_fp_input_a_sign_int
11688   \l_fp_input_a_integer_int
11689   \l_fp_input_a_decimal_int
11690   \l_fp_input_a_exponent_int
11691   \tl_set:Nx \l_fp_arg_tl
11692   {
11693     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11694     -
11695     \else:
11696     +
11697     \fi:
11698     \int_use:N \l_fp_input_a_integer_int
11699     .
11700     \exp_after:wN \use_none:n
11701     \int_value:w \int_eval:w
11702     \l_fp_input_a_decimal_int + \c_one_thousand_million
11703     e
11704     \int_use:N \l_fp_input_a_exponent_int
11705   }
11706   \if_cs_exist:w c_fp_cos ( \l_fp_arg_tl ) _fp \cs_end:
11707   \else:
11708     \exp_after:wN \fp_cos_aux_i:
11709   \fi:
11710   \cs_set_protected_nopar:Npx \fp_tmp:w
11711   {
11712     \group_end:
11713     #1 \exp_not:N #2
11714     { \use:c { c_fp_cos ( \l_fp_arg_tl ) _fp } }
11715   }
11716   \fp_tmp:w
11717 }

```

Almost the same as for sine: just a bit of correction for the sign of the output.

```

11718 \cs_new_protected_nopar:Npn \fp_cos_aux_i:
11719 {
11720   \fp_trig_normalise:
11721   \fp_cos_aux_ii:

```

```

11722 \if_int_compare:w \l_fp_output_integer_int = \c_one
11723 \l_fp_output_exponent_int \c_zero
11724 \else:
11725 \l_fp_output_integer_int \l_fp_output_decimal_int
11726 \l_fp_output_decimal_int \l_fp_output_extended_int
11727 \l_fp_output_exponent_int -\c_nine
11728 \fi:
11729 \fp_standardise:NNNN
11730 \l_fp_input_a_sign_int
11731 \l_fp_output_integer_int
11732 \l_fp_output_decimal_int
11733 \l_fp_output_exponent_int
11734 \tl_new:c { c_fp_cos ( \l_fp_arg_tl ) _fp }
11735 \tl_gset:cx { c_fp_cos ( \l_fp_arg_tl ) _fp }
11736 {
11737 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11738 +
11739 \else:
11740 -
11741 \fi:
11742 \int_use:N \l_fp_output_integer_int
11743 .
11744 \exp_after:wN \use_none:n
11745 \int_value:w \int_eval:w
11746 \l_fp_output_decimal_int + \c_one_thousand_million
11747 e
11748 \int_use:N \l_fp_output_exponent_int
11749 }
11750 }
11751 \cs_new_protected_nopar:Npn \fp_cos_aux_ii:
11752 {
11753 \if_case:w \l_fp_trig_octant_int
11754 \or:
11755 \exp_after:wN \fp_trig_calc_cos:
11756 \or:
11757 \exp_after:wN \fp_trig_calc_sin:
11758 \or:
11759 \exp_after:wN \fp_trig_calc_sin:
11760 \or:
11761 \exp_after:wN \fp_trig_calc_cos:
11762 \fi:
11763 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11764 \if_int_compare:w \l_fp_trig_octant_int > \c_two
11765 \l_fp_input_a_sign_int \c_minus_one
11766 \fi:
11767 \else:
11768 \if_int_compare:w \l_fp_trig_octant_int > \c_two
11769 \else:
11770 \l_fp_input_a_sign_int \c_one
11771 \fi:

```

```

11772     \fi:
11773   }
      (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:
11774 \cs_new_protected_nopar:Npn \fp_trig_calc_cos:
11775 {
11776   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11777     \l_fp_output_integer_int \c_one
11778     \l_fp_output_decimal_int \c_zero
11779   \else:
11780     \l_fp_trig_sign_int \c_minus_one
11781     \fp_mul:NNNNNN
11782       \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11783       \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11784       \l_fp_trig_decimal_int \l_fp_trig_extended_int
11785     \fp_div_integer:NNNNN
11786       \l_fp_trig_decimal_int \l_fp_trig_extended_int
11787       \c_two
11788       \l_fp_trig_decimal_int \l_fp_trig_extended_int
11789     \l_fp_count_int \c_three
11790     \if_int_compare:w \l_fp_trig_extended_int = \c_zero
11791       \if_int_compare:w \l_fp_trig_decimal_int = \c_zero
11792         \l_fp_output_integer_int \c_one
11793         \l_fp_output_decimal_int \c_zero
11794         \l_fp_output_extended_int \c_zero
11795       \else:
11796         \l_fp_output_integer_int \c_zero
11797         \l_fp_output_decimal_int \c_one_thousand_million
11798         \l_fp_output_extended_int \c_zero
11799     \fi:
11800   \else:
11801     \l_fp_output_integer_int \c_zero
11802     \l_fp_output_decimal_int 999999999 \scan_stop:
11803     \l_fp_output_extended_int \c_one_thousand_million
11804   \fi:
11805   \tex_advance:D \l_fp_output_extended_int -\l_fp_trig_extended_int
11806   \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
11807   \exp_after:wN \fp_trig_calc_Taylor:
11808 \fi:
11809 }
11810 \cs_new_protected_nopar:Npn \fp_trig_calc_sin:
11811 {
11812   \l_fp_output_integer_int \c_zero
11813   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11814     \l_fp_output_decimal_int \c_zero
11815   \else:
11816     \l_fp_output_decimal_int \l_fp_input_a_decimal_int
11817     \l_fp_output_extended_int \l_fp_input_a_extended_int
11818     \l_fp_trig_sign_int \c_one

```

```

11819     \l_fp_trig_decimal_int \l_fp_input_a_decimal_int
11820     \l_fp_trig_extended_int \l_fp_input_a_extended_int
11821     \l_fp_count_int \c_two
11822     \exp_after:wN \fp_trig_calc_Taylor:
11823   \fi:
11824 }

```

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.

```

11825 \cs_new_protected_nopar:Npn \fp_trig_calc_Taylor:
11826 {
11827   \l_fp_trig_sign_int -\l_fp_trig_sign_int
11828   \fp_mul:NNNNNN
11829     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11830     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11831     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11832   \fp_mul:NNNNNN
11833     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11834     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11835     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11836   \fp_div_integer:NNNNN
11837     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11838     \l_fp_count_int
11839     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11840   \tex_advance:D \l_fp_count_int \c_one
11841   \fp_div_integer:NNNNN
11842     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11843     \l_fp_count_int
11844     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11845   \tex_advance:D \l_fp_count_int \c_one
11846   \if_int_compare:w \l_fp_trig_decimal_int > \c_zero
11847     \if_int_compare:w \l_fp_trig_sign_int > \c_zero
11848       \tex_advance:D \l_fp_output_decimal_int \l_fp_trig_decimal_int
11849       \tex_advance:D \l_fp_output_extended_int
11850         \l_fp_trig_extended_int
11851     \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
11852     \else:
11853       \tex_advance:D \l_fp_output_decimal_int \c_one
11854       \tex_advance:D \l_fp_output_extended_int
11855         -\c_one_thousand_million
11856     \fi:
11857   \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
11858   \else:
11859     \tex_advance:D \l_fp_output_integer_int \c_one
11860     \tex_advance:D \l_fp_output_decimal_int
11861       -\c_one_thousand_million
11862   \fi:
11863   \else:
11864     \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
11865     \tex_advance:D \l_fp_output_extended_int

```

```

11866         -\l_fp_input_a_extended_int
11867         \if_int_compare:w \l_fp_output_extended_int < \c_zero
11868             \tex_advance:D \l_fp_output_decimal_int \c_minus_one
11869             \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11870         \fi:
11871         \if_int_compare:w \l_fp_output_decimal_int < \c_zero
11872             \tex_advance:D \l_fp_output_integer_int \c_minus_one
11873             \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11874         \fi:
11875     \fi:
11876     \exp_after:wN \fp_trig_calc_Taylor:
11877 \fi:
11878 }

```

(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:
11879 \cs_new_protected_nopar:Npn \fp_tan:Nn { \fp_tan_aux:NNn \tl_set:Nn }
11880 \cs_new_protected_nopar:Npn \fp_gtan:Nn { \fp_tan_aux:NNn \tl_gset:Nn }
11881 \cs_generate_variant:Nn \fp_tan:Nn { c }
11882 \cs_generate_variant:Nn \fp_gtan:Nn { c }
11883 \cs_new_protected_nopar:Npn \fp_tan_aux:NNn #1#2#3
11884 {
11885     \group_begin:
11886     \fp_split:Nn a {#3}
11887     \fp_standardise:NNNN
11888     \l_fp_input_a_sign_int
11889     \l_fp_input_a_integer_int
11890     \l_fp_input_a_decimal_int
11891     \l_fp_input_a_exponent_int
11892     \tl_set:Nx \l_fp_arg_tl
11893     {
11894         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11895             -
11896         \else:
11897             +
11898         \fi:
11899         \int_use:N \l_fp_input_a_integer_int
11900         .
11901         \exp_after:wN \use_none:n
11902         \int_value:w \int_eval:w
11903         \l_fp_input_a_decimal_int + \c_one_thousand_million
11904         e
11905         \int_use:N \l_fp_input_a_exponent_int
11906     }
11907     \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
11908         \cs_set_protected_nopar:Npx \fp_tmp:w

```

```

11909     {
11910         \group_end:
11911         #1 \exp_not:N #2 { \l_fp_arg_tl }
11912     }
11913 \else:
11914     \if_cs_exist:w
11915         c_fp_tan ( \l_fp_arg_tl ) _fp
11916     \cs_end:
11917 \else:
11918     \exp_after:wN \exp_after:wN \exp_after:wN
11919         \fp_tan_aux_i:
11920 \fi:
11921 \cs_set_protected_nopar:Npx \fp_tmp:w
11922     {
11923         \group_end:
11924         #1 \exp_not:N #2
11925         { \use:c { c_fp_tan ( \l_fp_arg_tl ) _fp } }
11926     }
11927 \fi:
11928 \fp_tmp:w
11929 }

```

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.

```

11930 \cs_new_protected_nopar:Npn \fp_tan_aux_i:
11931 {
11932     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11933         \exp_after:wN \fp_tan_aux_ii:
11934     \else:
11935         \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11936         \c_zero_fp
11937         \exp_after:wN \fp_trig_overflow_msg:
11938     \fi:
11939 }
11940 \cs_new_protected_nopar:Npn \fp_tan_aux_ii:
11941 {
11942     \fp_trig_normalise:
11943     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11944         \if_int_compare:w \l_fp_trig_octant_int > \c_two
11945             \l_fp_output_sign_int \c_minus_one
11946         \else:
11947             \l_fp_output_sign_int \c_one
11948         \fi:
11949     \else:
11950         \if_int_compare:w \l_fp_trig_octant_int > \c_two
11951             \l_fp_output_sign_int \c_one
11952         \else:
11953             \l_fp_output_sign_int \c_minus_one

```

```

11954     \fi:
11955 \fi:
11956 \fp_cos_aux_ii:
11957 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11958   \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11959     \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11960     \c_undefined_fp
11961   \else:
11962     \exp_after:wN \exp_after:wN \exp_after:wN
11963     \fp_tan_aux_iii:
11964   \fi:
11965 \else:
11966   \exp_after:wN \fp_tan_aux_iii:
11967 \fi:
11968 }

```

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.

```

11969 \cs_new_protected_nopar:Npn \fp_tan_aux_iii:
11970 {
11971   \l_fp_input_b_integer_int \l_fp_output_decimal_int
11972   \l_fp_input_b_decimal_int \l_fp_output_extended_int
11973   \l_fp_input_b_exponent_int -\c_nine
11974   \fp_standardise:NNNN
11975   \l_fp_input_b_sign_int
11976   \l_fp_input_b_integer_int
11977   \l_fp_input_b_decimal_int
11978   \l_fp_input_b_exponent_int
11979   \fp_sin_aux_ii:
11980   \l_fp_input_a_integer_int \l_fp_output_decimal_int
11981   \l_fp_input_a_decimal_int \l_fp_output_extended_int
11982   \l_fp_input_a_exponent_int -\c_nine
11983   \fp_standardise:NNNN
11984   \l_fp_input_a_sign_int
11985   \l_fp_input_a_integer_int
11986   \l_fp_input_a_decimal_int
11987   \l_fp_input_a_exponent_int
11988   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11989     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11990       \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11991       \c_zero_fp
11992     \else:
11993       \exp_after:wN \exp_after:wN \exp_after:wN \fp_tan_aux_iv:
11994     \fi:
11995   \else:
11996     \exp_after:wN \fp_tan_aux_iv:
11997   \fi:
11998 }
11999 \cs_new_protected_nopar:Npn \fp_tan_aux_iv:
12000 {

```

```

12001 \l_fp_output_integer_int \c_zero
12002 \l_fp_output_decimal_int \c_zero
12003 \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
12004 \l_fp_div_offset_int \c_one_hundred_million
12005 \fp_div_loop:
12006 \l_fp_output_exponent_int
12007 \int_eval:w
12008 \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
12009 \int_eval_end:
12010 \fp_standardise:NNNN
12011 \l_fp_output_sign_int
12012 \l_fp_output_integer_int
12013 \l_fp_output_decimal_int
12014 \l_fp_output_exponent_int
12015 \tl_new:c { c_fp_tan ( \l_fp_arg_tl ) _fp }
12016 \tl_gset:cx { c_fp_tan ( \l_fp_arg_tl ) _fp }
12017 {
12018 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12019 +
12020 \else:
12021 -
12022 \fi:
12023 \int_use:N \l_fp_output_integer_int
12024 .
12025 \exp_after:wN \use_none:n
12026 \int_value:w \int_eval:w
12027 \l_fp_output_decimal_int + \c_one_thousand_million
12028 e
12029 \int_use:N \l_fp_output_exponent_int
12030 }
12031 }

```

(End definition for `\fp_tan:Nn` and `\fp_tan:cn`. These functions are documented on page ??.)

201.12 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>	12032	<code>\tl_const:cn { c_fp_exp_1_tl }</code>	{ { 2 } { 718281828 } { 459045235 } { 0 } }
<code>\c_fp_exp_3_tl</code>	12033	<code>\tl_const:cn { c_fp_exp_2_tl }</code>	{ { 7 } { 389056098 } { 930650227 } { 0 } }
<code>\c_fp_exp_4_tl</code>	12034	<code>\tl_const:cn { c_fp_exp_3_tl }</code>	{ { 2 } { 008553692 } { 318766774 } { 1 } }
<code>\c_fp_exp_5_tl</code>	12035	<code>\tl_const:cn { c_fp_exp_4_tl }</code>	{ { 5 } { 459815003 } { 314423908 } { 1 } }
<code>\c_fp_exp_6_tl</code>	12036	<code>\tl_const:cn { c_fp_exp_5_tl }</code>	{ { 1 } { 484131591 } { 025766034 } { 2 } }
<code>\c_fp_exp_7_tl</code>	12037	<code>\tl_const:cn { c_fp_exp_6_tl }</code>	{ { 4 } { 034287934 } { 927351226 } { 2 } }
<code>\c_fp_exp_8_tl</code>	12038	<code>\tl_const:cn { c_fp_exp_7_tl }</code>	{ { 1 } { 096633158 } { 428458599 } { 3 } }
<code>\c_fp_exp_9_tl</code>	12039	<code>\tl_const:cn { c_fp_exp_8_tl }</code>	{ { 2 } { 980957987 } { 041728275 } { 3 } }
<code>\c_fp_exp_10_tl</code>	12040	<code>\tl_const:cn { c_fp_exp_9_tl }</code>	{ { 8 } { 103083927 } { 575384008 } { 3 } }
<code>\c_fp_exp_20_tl</code>	12041	<code>\tl_const:cn { c_fp_exp_10_tl }</code>	{ { 2 } { 202646579 } { 480671652 } { 4 } }
<code>\c_fp_exp_30_tl</code>	12042	<code>\tl_const:cn { c_fp_exp_20_tl }</code>	{ { 4 } { 851651954 } { 097902280 } { 8 } }
<code>\c_fp_exp_40_tl</code>	12043	<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>			


```

12044 \tl_const:cn { c_fp_exp_40_tl } { { 2 } { 353852668 } { 370199854 } { 17 } }
12045 \tl_const:cn { c_fp_exp_50_tl } { { 5 } { 184705528 } { 587072464 } { 21 } }
12046 \tl_const:cn { c_fp_exp_60_tl } { { 1 } { 142007389 } { 815684284 } { 26 } }
12047 \tl_const:cn { c_fp_exp_70_tl } { { 2 } { 515438670 } { 919167006 } { 30 } }
12048 \tl_const:cn { c_fp_exp_80_tl } { { 5 } { 540622384 } { 393510053 } { 34 } }
12049 \tl_const:cn { c_fp_exp_90_tl } { { 1 } { 220403294 } { 317840802 } { 39 } }
12050 \tl_const:cn { c_fp_exp_100_tl } { { 2 } { 688117141 } { 816135448 } { 43 } }
12051 \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 12052 \tl_const:cn { c_fp_exp_-1_tl } { { 3 } { 678794411 } { 71442322 } { -1 } }
\c_fp_exp_-3_tl 12053 \tl_const:cn { c_fp_exp_-2_tl } { { 1 } { 353352832 } { 366132692 } { -1 } }
\c_fp_exp_-4_tl 12054 \tl_const:cn { c_fp_exp_-3_tl } { { 4 } { 978706836 } { 786394298 } { -2 } }
\c_fp_exp_-5_tl 12055 \tl_const:cn { c_fp_exp_-4_tl } { { 1 } { 831563888 } { 873418029 } { -2 } }
\c_fp_exp_-6_tl 12056 \tl_const:cn { c_fp_exp_-5_tl } { { 6 } { 737946999 } { 085467097 } { -3 } }
\c_fp_exp_-7_tl 12057 \tl_const:cn { c_fp_exp_-6_tl } { { 2 } { 478752176 } { 666358423 } { -3 } }
\c_fp_exp_-8_tl 12058 \tl_const:cn { c_fp_exp_-7_tl } { { 9 } { 118819655 } { 545162080 } { -4 } }
\c_fp_exp_-9_tl 12059 \tl_const:cn { c_fp_exp_-8_tl } { { 3 } { 354626279 } { 025118388 } { -4 } }
\c_fp_exp_-10_tl 12060 \tl_const:cn { c_fp_exp_-9_tl } { { 1 } { 234098040 } { 866795495 } { -4 } }
\c_fp_exp_-20_tl 12061 \tl_const:cn { c_fp_exp_-10_tl } { { 4 } { 539992976 } { 248451536 } { -5 } }
\c_fp_exp_-30_tl 12062 \tl_const:cn { c_fp_exp_-20_tl } { { 2 } { 061153622 } { 438557828 } { -9 } }
\c_fp_exp_-40_tl 12063 \tl_const:cn { c_fp_exp_-30_tl } { { 9 } { 357622968 } { 840174605 } { -14 } }
\c_fp_exp_-50_tl 12064 \tl_const:cn { c_fp_exp_-40_tl } { { 4 } { 248354255 } { 291588995 } { -18 } }
\c_fp_exp_-60_tl 12065 \tl_const:cn { c_fp_exp_-50_tl } { { 1 } { 928749847 } { 963917783 } { -22 } }
\c_fp_exp_-70_tl 12066 \tl_const:cn { c_fp_exp_-60_tl } { { 8 } { 756510762 } { 696520338 } { -27 } }
\c_fp_exp_-80_tl 12067 \tl_const:cn { c_fp_exp_-70_tl } { { 3 } { 975449735 } { 908646808 } { -31 } }
\c_fp_exp_-90_tl 12068 \tl_const:cn { c_fp_exp_-80_tl } { { 1 } { 804851387 } { 845415172 } { -35 } }
\c_fp_exp_-100_tl 12069 \tl_const:cn { c_fp_exp_-90_tl } { { 8 } { 194012623 } { 990515430 } { -40 } }
\c_fp_exp_-200_tl 12070 \tl_const:cn { c_fp_exp_-100_tl } { { 3 } { 720075976 } { 020835963 } { -44 } }
12071 \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 12072 \cs_new_protected_nopar:Npn \fp_exp:Nn { \fp_exp_aux:NNn \tl_set:Nn }
\fp_exp_internal: 12073 \cs_new_protected_nopar:Npn \fp_gexp:Nn { \fp_exp_aux:NNn \tl_gset:Nn }
\fp_exp_aux: 12074 \cs_generate_variant:Nn \fp_exp:Nn { c }
\fp_exp_integer: 12075 \cs_generate_variant:Nn \fp_gexp:Nn { c }
\fp_exp_integer_tens: 12076 \cs_new_protected_nopar:Npn \fp_exp_aux:NNn #1#2#3
\fp_exp_integer_units: 12077 {
\fp_exp_integer_const:n 12078 \group_begin:
\fp_exp_integer_const:nnnn 12079 \fp_split:Nn a {#3}
\fp_exp_decimal: 12080 \fp_standardise:NNNN
\fp_exp_Taylor: 12081 \l_fp_input_a_sign_int
\fp_exp_const:Nx 12082 \l_fp_input_a_integer_int
\fp_exp_const:cx 12083 \l_fp_input_a_decimal_int

```

```

12084         \l_fp_input_a_exponent_int
12085     \l_fp_input_a_extended_int \c_zero
12086     \tl_set:Nx \l_fp_arg_tl
12087     {
12088         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12089         -
12090         \else:
12091         +
12092         \fi:
12093         \int_use:N \l_fp_input_a_integer_int
12094         .
12095         \exp_after:wN \use_none:n
12096         \int_value:w \int_eval:w
12097         \l_fp_input_a_decimal_int + \c_one_thousand_million
12098         e
12099         \int_use:N \l_fp_input_a_exponent_int
12100     }
12101     \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp \cs_end:
12102     \else:
12103         \exp_after:wN \fp_exp_internal:
12104     \fi:
12105     \cs_set_protected_nopar:Npx \fp_tmp:w
12106     {
12107         \group_end:
12108         #1 \exp_not:N #2
12109         {
12110             \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp
12111             \cs_end:
12112             \use:c { c_fp_exp ( \l_fp_arg_tl ) _fp }
12113         \else:
12114             \c_zero_fp
12115         \fi:
12116         }
12117     }
12118     \fp_tmp:w
12119 }

```

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

```

12120 \cs_new_protected_nopar:Npn \fp_exp_internal:
12121 {
12122     \if_int_compare:w \l_fp_input_a_exponent_int < \c_three
12123     \fp_extended_normalise:
12124     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12125     \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12126     \exp_after:wN \exp_after:wN \exp_after:wN
12127     \exp_after:wN \exp_after:wN \exp_after:wN
12128     \exp_after:wN \fp_exp_aux:
12129     \else:

```

```

12130         \exp_after:wN \exp_after:wN \exp_after:wN
12131         \exp_after:wN \exp_after:wN \exp_after:wN
12132         \exp_after:wN \fp_exp_overflow_msg:
12133     \fi:
12134 \else:
12135     \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12136     \exp_after:wN \exp_after:wN \exp_after:wN
12137     \exp_after:wN \exp_after:wN \exp_after:wN
12138     \exp_after:wN \fp_exp_aux:
12139 \else:
12140     \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12141     { \c_zero_fp }
12142 \fi:
12143 \fi:
12144 \else:
12145     \exp_after:wN \fp_exp_overflow_msg:
12146 \fi:
12147 }

```

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.

```

12148 \cs_new_protected_nopar:Npn \fp_exp_aux:
12149 {
12150     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12151     \exp_after:wN \fp_exp_integer:
12152 \else:
12153     \l_fp_output_integer_int \c_one
12154     \l_fp_output_decimal_int \c_zero
12155     \l_fp_output_extended_int \c_zero
12156     \l_fp_output_exponent_int \c_zero
12157     \exp_after:wN \fp_exp_decimal:
12158 \fi:
12159 }

```

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).

```

12160 \cs_new_protected_nopar:Npn \fp_exp_integer:
12161 {
12162     \if_int_compare:w \l_fp_input_a_integer_int < \c_one_hundred
12163     \l_fp_exp_integer_int \c_one

```

```

12164     \l_fp_exp_decimal_int  \c_zero
12165     \l_fp_exp_extended_int \c_zero
12166     \l_fp_exp_exponent_int \c_zero
12167     \exp_after:wN \fp_exp_integer_tens:
12168 \else:
12169     \tl_set:Nx \l_fp_tmp_tl
12170     {
12171         \exp_after:wN \use_i:nnn
12172         \int_use:N \l_fp_input_a_integer_int
12173     }
12174     \l_fp_input_a_integer_int
12175     \int_eval:w
12176     \l_fp_input_a_integer_int - \l_fp_tmp_tl 00
12177     \int_eval_end:
12178     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12179     \if_int_compare:w \l_fp_output_integer_int > 200 \scan_stop:
12180         \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12181         { \c_zero_fp }
12182     \else:
12183         \fp_exp_integer_const:n { - \l_fp_tmp_tl 00 }
12184         \exp_after:wN \exp_after:wN \exp_after:wN
12185         \exp_after:wN \exp_after:wN \exp_after:wN
12186         \exp_after:wN \fp_exp_integer_tens:
12187     \fi:
12188 \else:
12189     \fp_exp_integer_const:n { \l_fp_tmp_tl 00 }
12190     \exp_after:wN \exp_after:wN \exp_after:wN
12191     \exp_after:wN \fp_exp_integer_tens:
12192 \fi:
12193 \fi:
12194 }

```

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.

```

12195 \cs_new_protected_nopar:Npn \fp_exp_integer_tens:
12196 {
12197     \l_fp_output_integer_int  \l_fp_exp_integer_int
12198     \l_fp_output_decimal_int  \l_fp_exp_decimal_int
12199     \l_fp_output_extended_int \l_fp_exp_extended_int
12200     \l_fp_output_exponent_int \l_fp_exp_exponent_int
12201     \if_int_compare:w \l_fp_input_a_integer_int > \c_nine
12202         \tl_set:Nx \l_fp_tmp_tl
12203         {
12204             \exp_after:wN \use_i:nn
12205             \int_use:N \l_fp_input_a_integer_int
12206         }
12207     \l_fp_input_a_integer_int
12208     \int_eval:w
12209     \l_fp_input_a_integer_int - \l_fp_tmp_tl 0

```

```

12210         \int_eval_end:
12211         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12212             \fp_exp_integer_const:n { \l_fp_tmp_tl 0 }
12213         \else:
12214             \fp_exp_integer_const:n { - \l_fp_tmp_tl 0 }
12215         \fi:
12216         \fp_mul:NNNNNNNNN
12217             \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12218             \l_fp_output_integer_int \l_fp_output_decimal_int
12219             \l_fp_output_extended_int
12220             \l_fp_output_integer_int \l_fp_output_decimal_int
12221             \l_fp_output_extended_int
12222         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12223         \fp_extended_normalise_output:
12224     \fi:
12225     \fp_exp_integer_units:
12226 }
12227 \cs_new_protected_nopar:Npn \fp_exp_integer_units:
12228 {
12229     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12230     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12231         \fp_exp_integer_const:n { \int_use:N \l_fp_input_a_integer_int }
12232     \else:
12233         \fp_exp_integer_const:n
12234             { - \int_use:N \l_fp_input_a_integer_int }
12235     \fi:
12236     \fp_mul:NNNNNNNNN
12237         \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12238         \l_fp_output_integer_int \l_fp_output_decimal_int
12239         \l_fp_output_extended_int
12240         \l_fp_output_integer_int \l_fp_output_decimal_int
12241         \l_fp_output_extended_int
12242     \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12243     \fp_extended_normalise_output:
12244     \fi:
12245     \fp_exp_decimal:
12246 }

```

Recovery of the stored constant values into the separate registers is done with a simple expansion then assignment.

```

12247 \cs_new_protected_nopar:Npn \fp_exp_integer_const:n #1
12248 {
12249     \exp_after:wN \exp_after:wN \exp_after:wN
12250     \fp_exp_integer_const:nnnn
12251     \cs:w c_fp_exp_ #1 _tl \cs_end:
12252 }
12253 \cs_new_protected_nopar:Npn \fp_exp_integer_const:nnnn #1#2#3#4
12254 {
12255     \l_fp_exp_integer_int #1 \scan_stop:
12256     \l_fp_exp_decimal_int #2 \scan_stop:

```

```

12257     \l_fp_exp_extended_int #3 \scan_stop:
12258     \l_fp_exp_exponent_int #4 \scan_stop:
12259 }

```

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.

```

12260 \cs_new_protected_nopar:Npn \fp_exp_decimal:
12261 {
12262   \if_int_compare:w \l_fp_input_a_decimal_int > \c_zero
12263     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12264       \l_fp_exp_integer_int \c_one
12265       \l_fp_exp_decimal_int \l_fp_input_a_decimal_int
12266       \l_fp_exp_extended_int \l_fp_input_a_extended_int
12267     \else:
12268       \l_fp_exp_integer_int \c_zero
12269       \if_int_compare:w \l_fp_exp_extended_int = \c_zero
12270         \l_fp_exp_decimal_int
12271         \int_eval:w
12272           \c_one_thousand_million - \l_fp_input_a_decimal_int
12273         \int_eval_end:
12274         \l_fp_exp_extended_int \c_zero
12275       \else:
12276         \l_fp_exp_decimal_int
12277         \int_eval:w
12278           999999999 - \l_fp_input_a_decimal_int
12279         \scan_stop:
12280         \l_fp_exp_extended_int
12281         \int_eval:w
12282           \c_one_thousand_million - \l_fp_input_a_extended_int
12283         \int_eval_end:
12284       \fi:
12285     \fi:
12286     \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12287     \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12288     \l_fp_input_b_extended_int \l_fp_input_a_extended_int
12289     \l_fp_count_int \c_one
12290     \fp_exp_Taylor:
12291     \fp_mul:NNNNNNNNN
12292       \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12293       \l_fp_output_integer_int \l_fp_output_decimal_int
12294       \l_fp_output_extended_int
12295       \l_fp_output_integer_int \l_fp_output_decimal_int
12296       \l_fp_output_extended_int
12297     \fi:
12298     \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12299   \else:
12300     \tex_advance:D \l_fp_output_decimal_int \c_one
12301     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million

```

```

12302     \else:
12303         \l_fp_output_decimal_int \c_zero
12304         \tex_advance:D \l_fp_output_integer_int \c_one
12305     \fi:
12306 \fi:
12307 \fp_standardise:NNNN
12308     \l_fp_output_sign_int
12309     \l_fp_output_integer_int
12310     \l_fp_output_decimal_int
12311     \l_fp_output_exponent_int
12312 \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12313 {
12314     +
12315     \int_use:N \l_fp_output_integer_int
12316     .
12317     \exp_after:wN \use_none:n
12318     \int_value:w \int_eval:w
12319     \l_fp_output_decimal_int + \c_one_thousand_million
12320     e
12321     \int_use:N \l_fp_output_exponent_int
12322 }
12323 }

```

The Taylor series for $\exp(x)$ is

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

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`.

```

12324 \cs_new_protected_nopar:Npn \fp_exp_Taylor:
12325 {
12326     \tex_advance:D \l_fp_count_int \c_one
12327     \tex_multiply:D \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12328     \fp_mul:NNNNNN
12329     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12330     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12331     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12332     \fp_div_integer:NNNNN
12333     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12334     \l_fp_count_int
12335     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12336     \if_int_compare:w
12337         \int_eval:w
12338         \l_fp_input_b_decimal_int + \l_fp_input_b_extended_int
12339         > \c_zero
12340     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12341         \tex_advance:D \l_fp_exp_decimal_int \l_fp_input_b_decimal_int

```

```

12342 \tex_advance:D \l_fp_exp_extended_int
12343 \l_fp_input_b_extended_int
12344 \if_int_compare:w \l_fp_exp_extended_int < \c_one_thousand_million
12345 \else:
12346 \tex_advance:D \l_fp_exp_decimal_int \c_one
12347 \tex_advance:D \l_fp_exp_extended_int
12348 -\c_one_thousand_million
12349 \fi:
12350 \if_int_compare:w \l_fp_exp_decimal_int < \c_one_thousand_million
12351 \else:
12352 \tex_advance:D \l_fp_exp_integer_int \c_one
12353 \tex_advance:D \l_fp_exp_decimal_int
12354 -\c_one_thousand_million
12355 \fi:
12356 \else:
12357 \tex_advance:D \l_fp_exp_decimal_int -\l_fp_input_b_decimal_int
12358 \tex_advance:D \l_fp_exp_extended_int
12359 -\l_fp_input_a_extended_int
12360 \if_int_compare:w \l_fp_exp_extended_int < \c_zero
12361 \tex_advance:D \l_fp_exp_decimal_int \c_minus_one
12362 \tex_advance:D \l_fp_exp_extended_int \c_one_thousand_million
12363 \fi:
12364 \if_int_compare:w \l_fp_exp_decimal_int < \c_zero
12365 \tex_advance:D \l_fp_exp_integer_int \c_minus_one
12366 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12367 \fi:
12368 \fi:
12369 \exp_after:wN \fp_exp_Taylor:
12370 \fi:
12371 }

```

This is set up as a function so that the power code can redirect the effect.

```

12372 \cs_new_protected_nopar:Npn \fp_exp_const:Nx #1#2
12373 {
12374 \tl_new:N #1
12375 \tl_gset:Nx #1 {#2}
12376 }
12377 \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.

```

12378 \tl_const:cn { c_fp_ln_10_1_tl } { { 2 } { 302585092 } { 994045684 } { 0 } }
12379 \tl_const:cn { c_fp_ln_10_2_tl } { { 4 } { 605170185 } { 988091368 } { 0 } }
12380 \tl_const:cn { c_fp_ln_10_3_tl } { { 6 } { 907755278 } { 982137052 } { 0 } }
12381 \tl_const:cn { c_fp_ln_10_4_tl } { { 9 } { 210340371 } { 976182736 } { 0 } }
12382 \tl_const:cn { c_fp_ln_10_5_tl } { { 1 } { 151292546 } { 497022842 } { 1 } }
12383 \tl_const:cn { c_fp_ln_10_6_tl } { { 1 } { 381551055 } { 796427410 } { 1 } }
12384 \tl_const:cn { c_fp_ln_10_7_tl } { { 1 } { 611809565 } { 095831979 } { 1 } }
12385 \tl_const:cn { c_fp_ln_10_8_tl } { { 1 } { 842068074 } { 395226547 } { 1 } }
12386 \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 12387 \tl_const:cn { c_fp_ln_2_1_t1 } { { 0 } { 693147180 } { 559945309 } { 0 } }
\c_fp_ln_2_3_t1 12388 \tl_const:cn { c_fp_ln_2_2_t1 } { { 1 } { 386294361 } { 119890618 } { 0 } }
12389 \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 12390 \cs_new_protected_nopar:Npn \fp_ln:Nn { \fp_ln_aux:NNn \tl_set:Nn }
\fp_ln_aux:NNn 12391 \cs_new_protected_nopar:Npn \fp_gln:Nn { \fp_ln_aux:NNn \tl_gset:Nn }
\fp_ln_aux: 12392 \cs_generate_variant:Nn \fp_ln:Nn { c }
\fp_ln_exponent: 12393 \cs_generate_variant:Nn \fp_gln:Nn { c }
\fp_ln_internal: 12394 \cs_new_protected_nopar:Npn \fp_ln_aux:NNn #1#2#3
\fp_ln_exponent_tens: 12395 {
\fp_ln_exponent_units: 12396 \group_begin:
\fp_ln_normalise: 12397 \fp_split:Nn a {#3}
12398 \fp_standardise:NNNN
12399 \l_fp_input_a_sign_int
12400 \l_fp_input_a_integer_int
\fp_ln_mantissa: 12401 \l_fp_input_a_decimal_int
\fp_ln_mantissa_aux: 12402 \l_fp_input_a_exponent_int
\fp_ln_mantissa_divide_two: 12403 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
\fp_ln_integer_const:nn 12404 \if_int_compare:w
\fp_ln_Taylor: 12405 \int_eval:w
\fp_ln_fixed: 12406 \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
\fp_ln_fixed_aux:NNNNNNNNN 12407 > \c_zero
\fp_ln_Taylor_aux: 12408 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_aux:
12409 \else:
12410 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12411 {
12412 \group_end:
12413 ##1 \exp_not:N ##2 { \c_zero_fp }
12414 }
12415 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_error_msg:
12416 \fi:
12417 \else:
12418 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12419 {
12420 \group_end:
12421 ##1 \exp_not:N ##2 { \c_zero_fp }
12422 }
12423 \exp_after:wN \fp_ln_error_msg:
12424 \fi:
12425 \fp_tmp:w #1 #2
12426 }

```

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.

```

12427 \cs_new_protected_nopar:Npn \fp_ln_aux:
12428 {
12429   \tl_set:Nx \l_fp_arg_tl
12430   {
12431     +
12432     \int_use:N \l_fp_input_a_integer_int
12433     .
12434     \exp_after:wN \use_none:n
12435     \int_value:w \int_eval:w
12436     \l_fp_input_a_decimal_int + \c_one_thousand_million
12437     e
12438     \int_use:N \l_fp_input_a_exponent_int
12439   }
12440   \if_cs_exist:w c_fp_ln ( \l_fp_arg_tl ) _fp \cs_end:
12441   \else:
12442     \exp_after:wN \fp_ln_exponent:
12443     \fi:
12444     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12445     {
12446       \group_end:
12447       ##1 \exp_not:N ##2
12448       { \use:c { c_fp_ln ( \l_fp_arg_tl ) _fp } }
12449     }
12450   }

```

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.

```

12451 \cs_new_protected_nopar:Npn \fp_ln_exponent:
12452 {
12453   \fp_ln_internal:
12454   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12455   \else:
12456     \tex_advance:D \l_fp_output_decimal_int \c_one

```

```

12457 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12458 \else:
12459 \l_fp_output_decimal_int \c_zero
12460 \tex_advance:D \l_fp_output_integer_int \c_one
12461 \fi:
12462 \fi:
12463 \fp_standardise:NNNN
12464 \l_fp_output_sign_int
12465 \l_fp_output_integer_int
12466 \l_fp_output_decimal_int
12467 \l_fp_output_exponent_int
12468 \tl_const:cx { c_fp_ln ( \l_fp_arg_tl ) _fp }
12469 {
12470 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12471 +
12472 \else:
12473 -
12474 \fi:
12475 \int_use:N \l_fp_output_integer_int
12476 .
12477 \exp_after:wN \use_none:n
12478 \int_value:w \int_eval:w
12479 \l_fp_output_decimal_int + \c_one_thousand_million
12480 \scan_stop:
12481 e
12482 \int_use:N \l_fp_output_exponent_int
12483 }
12484 }
12485 \cs_new_protected_nopar:Npn \fp_ln_internal:
12486 {
12487 \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
12488 \l_fp_input_a_exponent_int -\l_fp_input_a_exponent_int
12489 \l_fp_output_sign_int \c_minus_one
12490 \else:
12491 \l_fp_output_sign_int \c_one
12492 \fi:
12493 \if_int_compare:w \l_fp_input_a_exponent_int > \c_nine
12494 \exp_after:wN \fp_ln_exponent_tens:NN
12495 \int_use:N \l_fp_input_a_exponent_int
12496 \else:
12497 \l_fp_output_integer_int \c_zero
12498 \l_fp_output_decimal_int \c_zero
12499 \l_fp_output_extended_int \c_zero
12500 \l_fp_output_exponent_int \c_zero
12501 \fi:
12502 \fp_ln_exponent_units:
12503 }
12504 \cs_new_protected_nopar:Npn \fp_ln_exponent_tens:NN #1 #2
12505 {
12506 \l_fp_input_a_exponent_int #2 \scan_stop:

```

```

12507 \fp_ln_const:nn { 10 } { #1 }
12508 \tex_advance:D \l_fp_exp_exponent_int \c_one
12509 \l_fp_output_integer_int \l_fp_exp_integer_int
12510 \l_fp_output_decimal_int \l_fp_exp_decimal_int
12511 \l_fp_output_extended_int \l_fp_exp_extended_int
12512 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12513 }

```

Next the smaller powers of ten, which will need to be combined with the above: always an additive process.

```

12514 \cs_new_protected_nopar:Npn \fp_ln_exponent_units:
12515 {
12516 \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
12517 \fp_ln_const:nn { 10 } { \int_use:N \l_fp_input_a_exponent_int }
12518 \fp_ln_normalise:
12519 \fp_add:NNNNNNNNN
12520 \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12521 \l_fp_output_integer_int \l_fp_output_decimal_int
12522 \l_fp_output_extended_int
12523 \l_fp_output_integer_int \l_fp_output_decimal_int
12524 \l_fp_output_extended_int
12525 \fi:
12526 \fp_ln_mantissa:
12527 }

```

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.

```

12528 \cs_new_protected_nopar:Npn \fp_ln_normalise:
12529 {
12530 \if_int_compare:w \l_fp_exp_exponent_int < \l_fp_output_exponent_int
12531 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12532 \exp_after:wN \use_i:nn \exp_after:wN
12533 \fp_ln_normalise_aux:NNNNNNNNN
12534 \int_use:N \l_fp_exp_decimal_int
12535 \exp_after:wN \fp_ln_normalise:
12536 \else:
12537 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12538 \fi:
12539 }
12540 \cs_new_protected_nopar:Npn \fp_ln_normalise_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12541 {
12542 \if_int_compare:w \l_fp_exp_integer_int = \c_zero
12543 \l_fp_exp_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12544 \else:
12545 \tl_set:Nx \l_fp_tmp_tl
12546 {
12547 \int_use:N \l_fp_exp_integer_int
12548 #1#2#3#4#5#6#7#8

```

```

12549     }
12550     \l_fp_exp_integer_int \c_zero
12551     \l_fp_exp_decimal_int \l_fp_tmp_tl \scan_stop:
12552 \fi:
12553 \tex_divide:D \l_fp_exp_extended_int \c_ten
12554 \tl_set:Nx \l_fp_tmp_tl
12555 {
12556     #9
12557     \int_use:N \l_fp_exp_extended_int
12558 }
12559 \l_fp_exp_extended_int \l_fp_tmp_tl \scan_stop:
12560 \tex_advance:D \l_fp_exp_exponent_int \c_one
12561 }

```

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*.

```

12562 \cs_new_protected_nopar:Npn \fp_ln_mantissa:
12563 {
12564     \l_fp_count_int \c_zero
12565     \l_fp_input_a_extended_int \c_zero
12566     \fp_ln_mantissa_aux:
12567     \if_int_compare:w \l_fp_count_int > \c_zero
12568         \fp_ln_const:nn { 2 } { \int_use:N \l_fp_count_int }
12569         \fp_ln_normalise:
12570         \if_int_compare:w \l_fp_output_sign_int > \c_zero
12571             \exp_after:wN \fp_add:NNNNNNNNN
12572         \else:
12573             \exp_after:wN \fp_sub:NNNNNNNNN
12574         \fi:
12575         \l_fp_output_integer_int \l_fp_output_decimal_int
12576         \l_fp_output_extended_int
12577         \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12578         \l_fp_output_integer_int \l_fp_output_decimal_int
12579         \l_fp_output_extended_int
12580     \fi:
12581     \if_int_compare:w
12582         \int_eval:w
12583         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int > \c_one
12584         \exp_after:wN \fp_ln_Taylor:
12585     \fi:
12586 }
12587 \cs_new_protected_nopar:Npn \fp_ln_mantissa_aux:
12588 {
12589     \if_int_compare:w \l_fp_input_a_integer_int > \c_one
12590         \tex_advance:D \l_fp_count_int \c_one
12591         \fp_ln_mantissa_divide_two:
12592         \exp_after:wN \fp_ln_mantissa_aux:
12593     \fi:
12594 }

```

A fast one-shot division by two.

```

12595 \cs_new_protected_nopar:Npn \fp_ln_mantissa_divide_two:
12596 {
12597   \if_int_odd:w \l_fp_input_a_decimal_int
12598     \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
12599   \fi:
12600   \if_int_odd:w \l_fp_input_a_integer_int
12601     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
12602   \fi:
12603   \tex_divide:D \l_fp_input_a_integer_int \c_two
12604   \tex_divide:D \l_fp_input_a_decimal_int \c_two
12605   \tex_divide:D \l_fp_input_a_extended_int \c_two
12606 }

```

Recovering constants makes use of the same auxiliary code as for exponents.

```

12607 \cs_new_protected_nopar:Npn \fp_ln_const:nn #1#2
12608 {
12609   \exp_after:wN \exp_after:wN \exp_after:wN
12610   \fp_exp_integer_const:nnnn
12611   \cs:w c_fp_ln_ #1 _ #2 _t1 \cs_end:
12612 }

```

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”.

```

12613 \cs_new_protected_nopar:Npn \fp_ln_Taylor:
12614 {

```

```

12615 \group_begin:
12616   \l_fp_input_a_integer_int \c_zero
12617   \l_fp_input_a_exponent_int \c_zero
12618   \l_fp_input_b_integer_int \c_two
12619   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12620   \l_fp_input_b_exponent_int \c_zero
12621   \fp_div_internal:
12622   \fp_ln_fixed:
12623   \l_fp_input_a_integer_int \l_fp_output_integer_int
12624   \l_fp_input_a_decimal_int \l_fp_output_decimal_int
12625   \l_fp_input_a_extended_int \c_zero
12626   \l_fp_input_a_exponent_int \l_fp_output_exponent_int
12627   \l_fp_output_decimal_int \c_zero %^^A Bug?
12628   \l_fp_output_decimal_int \l_fp_input_a_decimal_int
12629   \l_fp_output_extended_int \l_fp_input_a_extended_int
12630   \fp_mul:NNNNNN
12631     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12632     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12633     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12634   \l_fp_count_int \c_one
12635   \fp_ln_Taylor_aux:
12636   \cs_set_protected_nopar:Npx \fp_tmp:w
12637   {
12638     \group_end:
12639     \l_fp_exp_integer_int \c_zero
12640     \exp_not:N \l_fp_exp_decimal_int
12641       \int_use:N \l_fp_output_decimal_int \scan_stop:
12642     \exp_not:N \l_fp_exp_extended_int
12643       \int_use:N \l_fp_output_extended_int \scan_stop:
12644     \exp_not:N \l_fp_exp_exponent_int
12645       \int_use:N \l_fp_output_exponent_int \scan_stop:
12646   }
12647   \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.

```

12648   \tex_advance:D \l_fp_exp_decimal_int \l_fp_exp_decimal_int
12649   \if_int_compare:w \l_fp_exp_extended_int < \c_five_hundred_million
12650   \else:
12651     \tex_advance:D \l_fp_exp_extended_int -\c_five_hundred_million
12652     \tex_advance:D \l_fp_exp_decimal_int \c_one
12653   \fi:
12654   \tex_advance:D \l_fp_exp_extended_int \l_fp_exp_extended_int
12655   \fp_ln_normalise:
12656   \if_int_compare:w \l_fp_output_sign_int > \c_zero
12657     \exp_after:wN \fp_add:NNNNNNNNN
12658   \else:
12659     \exp_after:wN \fp_sub:NNNNNNNNN
12660   \fi:
12661   \l_fp_output_integer_int \l_fp_output_decimal_int

```

```

12662     \l_fp_output_extended_int
12663     \c_zero \l_fp_exp_decimal_int \l_fp_exp_extended_int
12664     \l_fp_output_integer_int \l_fp_output_decimal_int
12665     \l_fp_output_extended_int
12666 }

```

The usual shifts to move to fixed-point working. This is done using the `output` registers as this saves a reassignment here.

```

12667 \cs_new_protected_nopar:Npn \fp_ln_fixed:
12668 {
12669     \if_int_compare:w \l_fp_output_exponent_int < \c_zero
12670     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
12671     \exp_after:wN \use_i:nn \exp_after:wN
12672     \fp_ln_fixed_aux:NNNNNNNNN
12673     \int_use:N \l_fp_output_decimal_int
12674     \exp_after:wN \fp_ln_fixed:
12675     \fi:
12676 }
12677 \cs_new_protected_nopar:Npn \fp_ln_fixed_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12678 {
12679     \if_int_compare:w \l_fp_output_integer_int = \c_zero
12680     \l_fp_output_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12681     \else:
12682     \tl_set:Nx \l_fp_tmp_tl
12683     {
12684         \int_use:N \l_fp_output_integer_int
12685         #1#2#3#4#5#6#7#8
12686     }
12687     \l_fp_output_integer_int \c_zero
12688     \l_fp_output_decimal_int \l_fp_tmp_tl \scan_stop:
12689     \fi:
12690     \tex_advance:D \l_fp_output_exponent_int \c_one
12691 }

```

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.

```

12692 \cs_new_protected_nopar:Npn \fp_ln_Taylor_aux:
12693 {
12694     \tex_advance:D \l_fp_count_int \c_two
12695     \fp_mul:NNNNNN
12696     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12697     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12698     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12699     \if_int_compare:w
12700     \int_eval:w
12701     \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12702     > \c_zero
12703     \fp_div_integer:NNNNN
12704     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int

```



```

12705 \l_fp_count_int
12706 \l_fp_exp_decimal_int \l_fp_exp_extended_int
12707 \tex_advance:D \l_fp_output_decimal_int \l_fp_exp_decimal_int
12708 \tex_advance:D \l_fp_output_extended_int \l_fp_exp_extended_int
12709 \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
12710 \else:
12711 \tex_advance:D \l_fp_output_decimal_int \c_one
12712 \tex_advance:D \l_fp_output_extended_int
12713 -\c_one_thousand_million
12714 \fi:
12715 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12716 \else:
12717 \tex_advance:D \l_fp_output_integer_int \c_one
12718 \tex_advance:D \l_fp_output_decimal_int
12719 -\c_one_thousand_million
12720 \fi:
12721 \exp_after:wN \fp_ln_Taylor_aux:
12722 \fi:
12723 }

```

(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.

```

12724 \cs_new_protected_nopar:Npn \fp_pow:Nn { \fp_pow_aux:NNn \tl_set:Nn }
12725 \cs_new_protected_nopar:Npn \fp_gpow:Nn { \fp_pow_aux:NNn \tl_gset:Nn }
12726 \cs_generate_variant:Nn \fp_pow:Nn { c }
12727 \cs_generate_variant:Nn \fp_gpow:Nn { c }
12728 \cs_new_protected_nopar:Npn \fp_pow_aux:NNn #1#2#3
12729 {
12730 \group_begin:
12731 \fp_read:N #2
12732 \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12733 \l_fp_input_b_integer_int \l_fp_input_a_integer_int
12734 \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12735 \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
12736 \fp_split:Nn a {#3}
12737 \fp_standardise:NNNN
12738 \l_fp_input_a_sign_int
12739 \l_fp_input_a_integer_int
12740 \l_fp_input_a_decimal_int
12741 \l_fp_input_a_exponent_int
12742 \if_int_compare:w
12743 \int_eval:w
12744 \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
12745 = \c_zero
12746 \if_int_compare:w
12747 \int_eval:w

```

```

12748         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12749         = \c_zero
12750         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12751         {
12752             \group_end:
12753             ##1 ##2 { \c_undefined_fp }
12754         }
12755     \else:
12756         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12757         {
12758             \group_end:
12759             ##1 ##2 { \c_zero_fp }
12760         }
12761     \fi:
12762 \else:
12763     \if_int_compare:w
12764         \int_eval:w
12765         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12766         = \c_zero
12767         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12768         {
12769             \group_end:
12770             ##1 ##2 { \c_one_fp }
12771         }
12772     \else:
12773         \exp_after:wN \exp_after:wN \exp_after:wN
12774         \fp_pow_aux_i:
12775     \fi:
12776     \fi:
12777     \fp_tmp:w #1 #2
12778 }

```

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.

```

12779 \cs_new_protected_nopar:Npn \fp_pow_aux_i:
12780 {
12781     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12782         \tl_set:Nn \l_fp_sign_tl { + }
12783         \exp_after:wN \fp_pow_aux_ii:
12784     \else:
12785         \l_fp_input_a_extended_int \c_zero
12786         \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12787             \group_begin:
12788             \fp_extended_normalise:
12789             \if_int_compare:w
12790                 \int_eval:w
12791                 \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12792                 = \c_zero
12793             \group_end:

```

```

12794         \tl_set:Nn \l_fp_sign_tl { - }
12795         \exp_after:wN \exp_after:wN \exp_after:wN
12796         \exp_after:wN \exp_after:wN \exp_after:wN
12797         \exp_after:wN \fp_pow_aux_ii:
12798     \else:
12799         \group_end:
12800         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12801         {
12802             \group_end:
12803             ##1 ##2 { \c_undefined_fp }
12804         }
12805     \fi:
12806 \else:
12807     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12808     {
12809         \group_end:
12810         ##1 ##2 { \c_undefined_fp }
12811     }
12812 \fi:
12813 \fi:
12814 }

```

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.

```

12815 \cs_new_protected_nopar:Npn \fp_pow_aux_ii:
12816 {
12817     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12818         \exp_after:wN \fp_pow_aux_iv:
12819     \else:
12820         \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12821             \group_begin:
12822             \l_fp_input_a_extended_int \c_zero
12823             \fp_extended_normalise:
12824             \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12825                 \if_int_compare:w \l_fp_input_a_integer_int > \c_ten
12826                     \group_end:
12827                     \exp_after:wN \exp_after:wN \exp_after:wN
12828                     \exp_after:wN \exp_after:wN \exp_after:wN
12829                     \exp_after:wN \exp_after:wN \exp_after:wN
12830                     \exp_after:wN \exp_after:wN \exp_after:wN
12831                     \exp_after:wN \exp_after:wN \exp_after:wN
12832                     \fp_pow_aux_iv:
12833                 \else:
12834                     \group_end:
12835                     \exp_after:wN \exp_after:wN \exp_after:wN
12836                     \exp_after:wN \exp_after:wN \exp_after:wN
12837                     \exp_after:wN \exp_after:wN \exp_after:wN

```

```

12838         \exp_after:wN \exp_after:wN \exp_after:wN
12839         \exp_after:wN \exp_after:wN \exp_after:wN
12840         \exp_after:wN \fp_pow_aux_iii:
12841     \fi:
12842 \else:
12843     \group_end:
12844     \exp_after:wN \exp_after:wN \exp_after:wN
12845     \exp_after:wN \exp_after:wN \exp_after:wN
12846     \exp_after:wN \fp_pow_aux_iv:
12847 \fi:
12848 \else:
12849     \exp_after:wN \exp_after:wN \exp_after:wN
12850     \fp_pow_aux_iv:
12851 \fi:
12852 \fi:
12853 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12854 {
12855     \group_end:
12856     ##1 ##2
12857     {
12858         \l_fp_sign_tl
12859         \int_use:N \l_fp_output_integer_int
12860         .
12861         \exp_after:wN \use_none:n
12862         \int_value:w \int_eval:w
12863         \l_fp_output_decimal_int + \c_one_thousand_million
12864         e
12865         \int_use:N \l_fp_output_exponent_int
12866     }
12867 }
12868 }

```

For the small negative integer powers, the calculation is done for the positive power and the reciprocal is then taken.

```

12869 \cs_new_protected_nopar:Npn \fp_pow_aux_iii:
12870 {
12871     \l_fp_input_a_sign_int \c_one
12872     \fp_pow_aux_iv:
12873     \l_fp_input_a_integer_int \c_one
12874     \l_fp_input_a_decimal_int \c_zero
12875     \l_fp_input_a_exponent_int \c_zero
12876     \l_fp_input_b_integer_int \l_fp_output_integer_int
12877     \l_fp_input_b_decimal_int \l_fp_output_decimal_int
12878     \l_fp_input_b_exponent_int \l_fp_output_exponent_int
12879     \fp_div_internal:
12880 }

```

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.

```

12881 \cs_new_protected_nopar:Npn \fp_pow_aux_iv:
12882 {
12883   \group_begin:
12884     \l_fp_input_a_integer_int \l_fp_input_b_integer_int
12885     \l_fp_input_a_decimal_int \l_fp_input_b_decimal_int
12886     \l_fp_input_a_exponent_int \l_fp_input_b_exponent_int
12887     \fp_ln_internal:
12888     \cs_set_protected_nopar:Npx \fp_tmp:w
12889     {
12890       \group_end:
12891       \exp_not:N \l_fp_input_b_sign_int
12892       \int_use:N \l_fp_output_sign_int \scan_stop:
12893       \exp_not:N \l_fp_input_b_integer_int
12894       \int_use:N \l_fp_output_integer_int \scan_stop:
12895       \exp_not:N \l_fp_input_b_decimal_int
12896       \int_use:N \l_fp_output_decimal_int \scan_stop:
12897       \exp_not:N \l_fp_input_b_extended_int
12898       \int_use:N \l_fp_output_extended_int \scan_stop:
12899       \exp_not:N \l_fp_input_b_exponent_int
12900       \int_use:N \l_fp_output_exponent_int \scan_stop:
12901     }
12902     \fp_tmp:w
12903     \l_fp_input_a_extended_int \c_zero
12904     \fp_mul:NNNNNNNNN
12905     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
12906     \l_fp_input_a_extended_int
12907     \l_fp_input_b_integer_int \l_fp_input_b_decimal_int
12908     \l_fp_input_b_extended_int
12909     \l_fp_output_integer_int \l_fp_output_decimal_int
12910     \l_fp_output_extended_int
12911     \l_fp_output_exponent_int
12912     \int_eval:w
12913     \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
12914     \scan_stop:
12915     \fp_extended_normalise_output:
12916     \tex_multiply:D \l_fp_input_a_sign_int \l_fp_input_b_sign_int
12917     \l_fp_input_a_integer_int \l_fp_output_integer_int
12918     \l_fp_input_a_decimal_int \l_fp_output_decimal_int
12919     \l_fp_input_a_extended_int \l_fp_output_extended_int
12920     \l_fp_input_a_exponent_int \l_fp_output_exponent_int
12921     \l_fp_output_integer_int \c_zero
12922     \l_fp_output_decimal_int \c_zero
12923     \l_fp_output_extended_int \c_zero
12924     \l_fp_output_exponent_int \c_zero
12925     \cs_set_eq:NN \fp_exp_const:Nx \use_none:nn

```

```

12926     \fp_exp_internal:
12927 }
      (End definition for \fp_pow:Nn and \fp_pow:cn. These functions are documented on page ??.)

```

201.13 Tests for special values

\fp_if_undefined:N Testing for an undefined value is easy.

```

12928 \prg_new_conditional:Npnn \fp_if_undefined:N #1 { p , T , F , TF }
12929 {
12930     \if_meaning:w #1 \c_undefined_fp
12931     \prg_return_true:
12932 }else:
12933     \prg_return_false:
12934 \fi:
12935 }
      (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.

```

12936 \prg_new_conditional:Npnn \fp_if_zero:N #1 { p , T , F , TF }
12937 {
12938     \if_meaning:w #1 \c_zero_fp
12939     \prg_return_true:
12940 }else:
12941     \prg_return_false:
12942 \fi:
12943 }
      (End definition for \fp_if_zero:N. This function is documented on page 168.)

```

201.14 Floating-point conditionals

\fp_compare:nNn \fp_compare:NNN \fp_compare_aux:N 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=: 12944 \prg_new_protected_conditional:Npnn \fp_compare:nNn #1#2#3 { T , F , TF }
\fp_compare<: 12945 {
\fp_compare<_aux: 12946     \group_begin:
\fp_compare_absolute_a>b: 12947     \fp_split:Nn a {#1}
\fp_compare_absolute_a<b: 12948     \fp_standardise:NNNN
\fp_compare_>: 12949     \l_fp_input_a_sign_int
12950     \l_fp_input_a_integer_int
12951     \l_fp_input_a_decimal_int
12952     \l_fp_input_a_exponent_int
12953     \fp_split:Nn b {#3}
12954     \fp_standardise:NNNN
12955     \l_fp_input_b_sign_int
12956     \l_fp_input_b_integer_int
12957     \l_fp_input_b_decimal_int
12958     \l_fp_input_b_exponent_int

```

```

12959     \fp_compare_aux:N #2
12960   }
12961 \prg_new_protected_conditional:Npnn \fp_compare:NNN #1#2#3 { T , F , TF }
12962 {
12963   \group_begin:
12964     \fp_read:N #3
12965     \l_fp_input_b_sign_int     \l_fp_input_a_sign_int
12966     \l_fp_input_b_integer_int  \l_fp_input_a_integer_int
12967     \l_fp_input_b_decimal_int  \l_fp_input_a_decimal_int
12968     \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
12969     \fp_read:N #1
12970     \fp_compare_aux:N #2
12971   }
12972 \cs_new_protected_nopar:Npn \fp_compare_aux:N #1
12973 {
12974   \cs_if_exist:cTF { fp_compare_#1: }
12975   { \use:c { fp_compare_#1: } }
12976   {
12977     \group_end:
12978     \prg_return_false:
12979   }
12980 }

```

For equality, the test is pretty easy as things are either equal or they are not.

```

12981 \cs_new_protected_nopar:cpn { fp_compare_=: }
12982 {
12983   \if_int_compare:w \l_fp_input_a_sign_int = \l_fp_input_b_sign_int
12984   \if_int_compare:w \l_fp_input_a_integer_int = \l_fp_input_b_integer_int
12985   \if_int_compare:w \l_fp_input_a_decimal_int = \l_fp_input_b_decimal_int
12986   \if_int_compare:w
12987     \l_fp_input_a_exponent_int = \l_fp_input_b_exponent_int
12988   \group_end:
12989   \prg_return_true:
12990   \else:
12991     \group_end:
12992     \prg_return_false:
12993   \fi:
12994   \else:
12995     \group_end:
12996     \prg_return_false:
12997   \fi:
12998   \else:
12999     \group_end:
13000     \prg_return_false:
13001   \fi:
13002   \else:
13003     \group_end:
13004     \prg_return_false:
13005   \fi:
13006 }

```

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.

```

13007 \cs_new_protected_nopar:cpn { fp_compare_>: }
13008 {
13009   \if_int_compare:w \int_eval:w
13010     \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
13011     = \c_zero
13012   \if_int_compare:w \int_eval:w
13013     \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13014     = \c_zero
13015   \group_end:
13016   \prg_return_false:
13017   \else:
13018     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
13019     \group_end:
13020     \prg_return_false:
13021     \else:
13022     \group_end:
13023     \prg_return_true:
13024   \fi:
13025   \fi:
13026   \else:
13027     \if_int_compare:w \int_eval:w
13028       \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13029       = \c_zero
13030     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13031     \group_end:
13032     \prg_return_true:
13033     \else:
13034     \group_end:
13035     \prg_return_false:
13036   \fi:
13037   \else:
13038     \use:c { fp_compare_>_aux: }
13039   \fi:
13040   \fi:
13041 }

```

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.

```

13042 \cs_new_protected_nopar:cpn { fp_compare_>_aux: }
13043 {
13044   \if_int_compare:w \l_fp_input_a_sign_int > \l_fp_input_b_sign_int
13045   \group_end:
13046   \prg_return_true:
13047   \else:
13048     \if_int_compare:w \l_fp_input_a_sign_int < \l_fp_input_b_sign_int
13049     \group_end:
13050     \prg_return_false:
13051   \else:

```



```

13052         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13053         \use:c { fp_compare_absolute_a>b: }
13054     \else:
13055         \use:c { fp_compare_absolute_a<b: }
13056     \fi:
13057 \fi:
13058 \fi:
13059 }

```

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.

```

13060 \cs_new_protected_nopar:cpn { fp_compare_absolute_a>b: }
13061 {
13062     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
13063     \group_end:
13064     \prg_return_true:
13065 \else:
13066     \if_int_compare:w \l_fp_input_a_exponent_int < \l_fp_input_b_exponent_int
13067     \group_end:
13068     \prg_return_false:
13069 \else:
13070     \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
13071     \group_end:
13072     \prg_return_true:
13073 \else:
13074     \if_int_compare:w
13075         \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
13076     \group_end:
13077     \prg_return_false:
13078 \else:
13079     \if_int_compare:w
13080         \l_fp_input_a_decimal_int > \l_fp_input_b_decimal_int
13081     \group_end:
13082     \prg_return_true:
13083 \else:
13084     \group_end:
13085     \prg_return_false:
13086 \fi:
13087 \fi:
13088 \fi:
13089 \fi:
13090 \fi:
13091 }
13092 \cs_new_protected_nopar:cpn { fp_compare_absolute_a<b: }
13093 {
13094     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
13095     \group_end:
13096     \prg_return_true:
13097 \else:

```

```

13098 \if_int_compare:w \l_fp_input_b_exponent_int < \l_fp_input_a_exponent_int
13099 \group_end:
13100 \prg_return_false:
13101 \else:
13102 \if_int_compare:w \l_fp_input_b_integer_int > \l_fp_input_a_integer_int
13103 \group_end:
13104 \prg_return_true:
13105 \else:
13106 \if_int_compare:w
13107 \l_fp_input_b_integer_int < \l_fp_input_a_integer_int
13108 \group_end:
13109 \prg_return_false:
13110 \else:
13111 \if_int_compare:w
13112 \l_fp_input_b_decimal_int > \l_fp_input_a_decimal_int
13113 \group_end:
13114 \prg_return_true:
13115 \else:
13116 \group_end:
13117 \prg_return_false:
13118 \fi:
13119 \fi:
13120 \fi:
13121 \fi:
13122 \fi:
13123 }

```

This is just a case of reversing the two input values and then running the tests already defined.

```

13124 \cs_new_protected_nopar:cpn { fp_compare_<: }
13125 {
13126 \tl_set:Nx \l_fp_tmp_tl
13127 {
13128 \int_set:Nn \exp_not:N \l_fp_input_a_sign_int
13129 { \int_use:N \l_fp_input_b_sign_int }
13130 \int_set:Nn \exp_not:N \l_fp_input_a_integer_int
13131 { \int_use:N \l_fp_input_b_integer_int }
13132 \int_set:Nn \exp_not:N \l_fp_input_a_decimal_int
13133 { \int_use:N \l_fp_input_b_decimal_int }
13134 \int_set:Nn \exp_not:N \l_fp_input_a_exponent_int
13135 { \int_use:N \l_fp_input_b_exponent_int }
13136 \int_set:Nn \exp_not:N \l_fp_input_b_sign_int
13137 { \int_use:N \l_fp_input_a_sign_int }
13138 \int_set:Nn \exp_not:N \l_fp_input_b_integer_int
13139 { \int_use:N \l_fp_input_a_integer_int }
13140 \int_set:Nn \exp_not:N \l_fp_input_b_decimal_int
13141 { \int_use:N \l_fp_input_a_decimal_int }
13142 \int_set:Nn \exp_not:N \l_fp_input_b_exponent_int
13143 { \int_use:N \l_fp_input_a_exponent_int }
13144 }

```

```

13145 \l_fp_tmp_tl
13146 \use:c { fp_compare_>: }
13147 }

```

(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
13148 \prg_new_protected_conditional:Npnn \fp_compare:n #1 { T , F , TF }
13149 {
13150   \group_begin:
13151   \tl_set:Nx \l_fp_tmp_tl
13152   {
13153     \group_end:
13154     \fp_compare_aux_i:w #1 \exp_not:n { == \q_nil == \q_stop }
13155   }
13156   \l_fp_tmp_tl
13157 }
13158 \cs_new_protected_nopar:Npn \fp_compare_aux_i:w #1 == #2 == #3 \q_stop
13159 {
13160   \quark_if_nil:nTF {#2}
13161   { \fp_compare_aux_ii:w #1 != \q_nil != \q_stop }
13162   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13163 }
13164 \cs_new_protected_nopar:Npn \fp_compare_aux_ii:w #1 != #2 != #3 \q_stop
13165 {
13166   \quark_if_nil:nTF {#2}
13167   { \fp_compare_aux_iii:w #1 <= \q_nil <= \q_stop }
13168   { \fp_compare:nNnTF {#1} = {#2} \prg_return_false: \prg_return_true: }
13169 }
13170 \cs_new_protected_nopar:Npn \fp_compare_aux_iii:w #1 <= #2 <= #3 \q_stop
13171 {
13172   \quark_if_nil:nTF {#2}
13173   { \fp_compare_aux_iv:w #1 >= \q_nil >= \q_stop }
13174   { \fp_compare:nNnTF {#1} > {#2} \prg_return_false: \prg_return_true: }
13175 }
13176 \cs_new_protected_nopar:Npn \fp_compare_aux_iv:w #1 >= #2 >= #3 \q_stop
13177 {
13178   \quark_if_nil:nTF {#2}
13179   { \fp_compare_aux_v:w #1 = \q_nil \q_stop }
13180   { \fp_compare:nNnTF {#1} < {#2} \prg_return_false: \prg_return_true: }
13181 }
13182 \cs_new_protected_nopar:Npn \fp_compare_aux_v:w #1 = #2 = #3 \q_stop
13183 {
13184   \quark_if_nil:nTF {#2}
13185   { \fp_compare_aux_vi:w #1 < \q_nil < \q_stop }
13186   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13187 }
13188 \cs_new_protected_nopar:Npn \fp_compare_aux_vi:w #1 < #2 < #3 \q_stop

```

```

13189 {
13190   \quark_if_nil:nTF {#2}
13191   { \fp_compare_aux_vii:w #1 > \q_nil > \q_stop }
13192   { \fp_compare:nNnTF {#1} < {#2} \prg_return_true: \prg_return_false: }
13193 }
13194 \cs_new_protected_nopar:Npn \fp_compare_aux_vii:w #1 > #2 > #3 \q_stop
13195 {
13196   \quark_if_nil:nTF {#2}
13197   { \prg_return_false: }
13198   { \fp_compare:nNnTF {#1} > {#2} \prg_return_true: \prg_return_false: }
13199 }

```

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

201.15 Messages

\fp_overflow_msg: A generic overflow message, used whenever there is a possible overflow.

```

13200 \msg_kernel_new:nnnn { fpu } { overflow }
13201 { Number~too~big. }
13202 {
13203   The~input~given~is~too~big~for~the~LaTeX~floating~point~unit. \\
13204   Further~errors~may~well~occur!
13205 }
13206 \cs_new_protected_nopar:Npn \fp_overflow_msg:
13207 { \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.

```

13208 \msg_kernel_new:nnnn { fpu } { exponent-overflow }
13209 { Number~too~big~for~exponent~unit. }
13210 {
13211   The~exponent~of~the~input~given~is~too~big~for~the~floating~point~
13212   unit:~the~maximum~input~value~for~an~exponent~is~230.
13213 }
13214 \cs_new_protected_nopar:Npn \fp_exp_overflow_msg:
13215 { \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

```

13216 \msg_kernel_new:nnnn { fpu } { logarithm-input-error }
13217 { Invalid~input~to~ln~function. }
13218 { Logarithms~can~only~be~calculated~for~positive~numbers. }
13219 \cs_new_protected_nopar:Npn \fp_ln_error_msg: {
13220   \msg_kernel_error:nn { fpu } { logarithm-input-error }
13221 }

```

(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.
13222 \msg_kernel_new:nnnn { fpu } { trigonometric-overflow }
13223 { Number~too~big~for~trigonometry~unit. }
13224 {
13225   The~trigonometry~code~can~only~work~with~numbers~smaller~
13226   than~1000000000.
13227 }
13228 \cs_new_protected_nopar:Npn \fp_trig_overflow_msg:
13229 { \msg_kernel_error:nn { fpu } { trigonometric-overflow } }
      (End definition for \fp_trig_overflow_msg:. This function is documented on page ??.)
13230 </initex | package>

```

202 l3luatex implementation

```

13231 <*initex | package>

      Announce and ensure that the required packages are loaded.
13232 <*package>
13233 \ProvidesExplPackage
13234   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
13235 \package_check_loaded_expl:
13236 </package>

\lua_now:n When LuaTeX is in use, this is all a question of primitives with new names. On the
\lua_now:x other hand, for pdfTeX and XeTeX the argument should be removed from the input
\lua_shipout_x:n stream before issuing an error. This is expandable, using \msg_expandable_error:n as
\lua_shipout_x:x for V-type expansion.
\lua_shipout:n
\lua_shipout:x
13237 \luatex_if_engine:TF
13238 {
13239   \cs_new_eq:NN \lua_now:x \luatex_directlua:D
13240   \cs_new_eq:NN \lua_shipout_x:n \luatex_latelua:D
13241 }
13242 {
13243   \cs_new:Npn \lua_now:x #1
13244   {
13245     \msg_expandable_error:n
13246       { LuaTeX~ engine~ not~ in~ use!~ Ignoring~ \lua_now:x. }
13247   }
13248   \cs_new_protected:Npn \lua_shipout_x:n #1
13249   {
13250     \msg_expandable_error:n
13251       { LuaTeX~ engine~ not~ in~ use!~ Ignoring~ \lua_shipout_x:n. }
13252   }
13253 }
13254 \cs_new:Npn \lua_now:n #1
13255 { \lua_now:x { \exp_not:n {#1} } }
13256 \cs_generate_variant:Nn \lua_shipout_x:n { x }
13257 \cs_new_protected:Npn \lua_shipout:n #1
13258 { \lua_shipout_x:n { \exp_not:n {#1} } }

```

```

13259 \cs_generate_variant:Nn \lua_shipout:n { x }
      (End definition for \lua_now:n and \lua_now:x. These functions are documented on page ??.)

```

202.1 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`

```

13260 \int_new:N \g_cctab_allocate_int
13261 \int_set:Nn \g_cctab_allocate_int { -1 }
13262 \int_new:N \g_cctab_stack_int
13263 \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.

```

13264 \cs_new_protected_nopar:Npn \cctab_new:N #1
13265 {
13266   \cs_if_free:NTF #1
13267   {
13268     \int_gadd:Nn \g_cctab_allocate_int { 2 }
13269     \int_compare:nNnTF
13270     { \g_cctab_allocate_int } < { \c_max_register_int + 1 }
13271     {
13272       \tex_global:D \tex_mathchardef:D #1 \g_cctab_allocate_int
13273       \luatex_initcatcodetable:D #1
13274     }
13275     { \msg_kernel_fatal:nxx { alloc } { out-of-registers } { cctab } }
13276   }
13277   {
13278     \msg_kernel_error:nxx { code } { variable-already-defined }
13279     { \token_to_str:N #1 }
13280   }
13281 }
13282 \luatex_if_engine:F
13283 { \cs_set_protected_nopar:Npn \cctab_new:N #1 { \lua_wrong_engine: } }
13284 <*package>
13285 \luatex_if_engine:T
13286 {
13287   \cs_set_protected_nopar:Npn \cctab_new:N #1
13288   {
13289     \newcatcodetable #1
13290     \luatex_initcatcodetable:D #1
13291   }
13292 }
13293 </package>

```

(End definition for `\cctab_new:N`. This function is documented on page 175.)

`\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.

```

\l_cctab_tmp_tl 13294 \cs_new_protected_nopar:Npn \cctab_begin:N #1
13295 {
13296   \seq_gpush:Nx \g_cctab_stack_seq { \tex_the:D \luatex_catcodetable:D }
13297   \luatex_catcodetable:D #1
13298   \int_gadd:Nn \g_cctab_stack_int { 2 }
13299   \int_compare:nNnT { \g_cctab_stack_int } > { 268 435 453 }
13300     { \msg_kernel_error:nn { code } { cctab-stack-full } }
13301   \luatex_savecatcodetable:D \g_cctab_stack_int
13302   \luatex_catcodetable:D \g_cctab_stack_int
13303 }
13304 \cs_new_protected_nopar:Npn \cctab_end:
13305 {
13306   \int_gsub:Nn \g_cctab_stack_int { 2 }
13307   \seq_gpop:NN \g_cctab_stack_seq \l_cctab_tmp_tl
13308   \quark_if_no_value:NT \l_cctab_tmp_tl
13309     { \tl_set:Nn \l_cctab_tmp_tl { 0 } }
13310   \luatex_catcodetable:D \l_cctab_tmp_tl \scan_stop:
13311 }
13312 \luatex_if_engine:F
13313 {
13314   \cs_set_protected_nopar:Npn \cctab_begin:N #1 { \lua_wrong_engine: }
13315   \cs_set_protected_nopar:Npn \cctab_end: { \lua_wrong_engine: }
13316 }
13317 <*package>
13318 \luatex_if_engine:T
13319 {
13320   \cs_set_protected_nopar:Npn \cctab_begin:N #1 { \BeginCatcodeRegime #1 }
13321   \cs_set_protected_nopar:Npn \cctab_end: { \EndCatcodeRegime }
13322 }
13323 </package>
13324 \tl_new:N \l_cctab_tmp_tl

```

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

`\cctab_gset:Nn` Category code tables are always global, so only one version is needed. The set up here is simple, and means that at the point of use there is no need to worry about escaping category codes.

```

13325 \cs_new_protected:Npn \cctab_gset:Nn #1#2
13326 {
13327   \group_begin:
13328     #2
13329     \luatex_savecatcodetable:D #1
13330   \group_end:
13331 }
13332 \luatex_if_engine:F
13333 { \cs_set_protected_nopar:Npn \cctab_gset:Nn #1#2 { \lua_wrong_engine: } }

```

(End definition for `\cctab_gset:Nn`. This function is documented on page 175.)

`\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_string_cctab`

```

13334 \luatex_if_engine:T
13335 {
13336   \cctab_new:N \c_code_cctab
13337   \cctab_gset:Nn \c_code_cctab { }
13338 }
13339 <*package>
13340 \luatex_if_engine:T
13341 {
13342   \cs_new_eq:NN \c_document_cctab \CatcodeTableLaTeX
13343   \cs_new_eq:NN \c_initex_cctab \CatcodeTableIniTeX
13344   \cs_new_eq:NN \c_other_cctab \CatcodeTableOther
13345   \cs_new_eq:NN \c_string_cctab \CatcodeTableString
13346 }
13347 </package>
13348 <*initex>
13349 \luatex_if_engine:T
13350 {
13351   \cctab_new:N \c_document_cctab
13352   \cctab_new:N \c_other_cctab
13353   \cctab_new:N \c_string_cctab
13354   \cctab_gset:Nn \c_document_cctab
13355   {
13356     \char_set_catcode_space:n { 9 }
13357     \char_set_catcode_space:n { 32 }
13358     \char_set_catcode_other:n { 58 }
13359     \char_set_catcode_math_subscript:n { 95 }
13360     \char_set_catcode_active:n { 126 }
13361   }
13362   \cctab_gset:Nn \c_other_cctab
13363   {
13364     \prg_stepwise_inline:nmmm { 0 } { 1 } { 127 }
13365     { \char_set_catcode_other:n {#1} }
13366   }
13367   \cctab_gset:Nn \c_string_cctab
13368   {
13369     \prg_stepwise_inline:nmmm { 0 } { 1 } { 127 }
13370     { \char_set_catcode_other:n {#1} }
13371     \char_set_catcode_space:n { 32 }
13372   }
13373 }
13374 </initex>
13375 </initex | package>

```

(End definition for `\c_code_cctab`. This function is documented on page 176.)

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	
\!	8655
\#	5, 8358
\%	8360
\	2283, 2284, 2297, 2298, 8655, 8656
*	2591, 2593, 2597, 2604, 8315, 8316
\,	9114, 9116
\-	348
\.	8597, 8602
\.bool_gset:N	154, 9456
\.bool_gset_inverse:N	154, 9460
\.bool_set:N	153, 9456
\.bool_set_inverse:N	154, 9460
\.choice:	154, 9464
\.choice_code:n	154, 9472
\.choice_code:x	154, 9472
\.choices:nn	154, 9466
\.clist_gset:N	154, 9476
\.clist_gset:c	154, 9476
\.clist_set:N	154, 9476
\.clist_set:c	154, 9476
\.code:n	155, 9468
\.code:x	155, 9468
\.default:V	155, 9484
\.default:n	155, 9484
\.dim_gset:N	155, 9488
\.dim_gset:c	155, 9488
\.dim_set:N	155, 9488
\.dim_set:c	155, 9488
\.fp_gset:N	155, 9496
\.fp_gset:c	155, 9496
\.fp_set:N	155, 9496
\.fp_set:c	155, 9496
\.generate_choices:n	156, 9504
\.int_gset:N	156, 9506
\.int_gset:c	156, 9506
\.int_set:N	156, 9506
\.int_set:c	156, 9506
\.meta:n	156, 9514
\.meta:x	156, 9514
\.multichoice:	156, 9518
\.multichoice:nn	156
\.multichoices:nn	9518
\.skip_gset:N	156, 9522
\.skip_gset:c	156, 9522
\.skip_set:N	156, 9522
\.skip_set:c	156, 9522
\.tl_gset:N	157, 9530
\.tl_gset:c	157, 9530
\.tl_gset_x:N	157, 9530
\.tl_gset_x:c	157, 9530
\.tl_set:N	157, 9530
\.tl_set:c	157, 9530
\.tl_set_x:N	157, 9530
\.tl_set_x:c	157, 9530
\.value_forbidden:	157, 9546
\.value_required:	157, 9546
\/	347
\:	1068, 2683, 2816
\::	30,
	1537, 1538, 1539, 1539–1542, 1544,
	1546, 1553, 1558, 1564, 1685–1711,
	1713, 1718, 1720, 1725, 1751–1753
\::N	30, 1541,
	1541, 1694, 1700, 1701, 1705, 1706
\::V	30, 1558, 1558, 1691
\::V_unbraced	1712, 1720
\::c	30, 1542, 1542,
	1686, 1692, 1695, 1702, 1709, 1710
\::f	30, 1546, 1546, 1687–1689, 1753
\::f_unbraced	1712, 1713
\::n	30, 1540, 1540,
	1686, 1689–1691, 1696, 1700, 1702,
	1703, 1705, 1707, 1708, 1710, 1751
\::o	30, 1544, 1544, 1687, 1690,
	1692, 1693, 1697, 1698, 1700, 1701,
	1703, 1704, 1706, 1708, 1711, 1752
\::o_unbraced	1712, 1718, 1751–1753
\::v	30, 1558, 1564
\::v_unbraced	1712, 1725
\::x	30, 1553,
	1553, 1685, 1694–1699, 1705–1711
\;	2683, 2815, 2816
\=	9113, 9115
\?	1835, 2723
\@	1068, 1069, 4349, 4350
\@end	766
\@hyph	769

`\@input` 770
`\@italiccorr` 771
`\@underline` 772
`\@addtofilelist` 9837
`\@currname` 9765
`\@filelist` 9862
`\@ifpackageloaded` 216
`\@namedef` 198
`\@nil` 193, 201
`\@popfilename` 163, 181, 183
`\@pushfilename` 163, 164, 179
`\@tempa` 54, 56, 64
`\@tempboxa` 6468
`\` 1835, 2723,
 8273, 8280, 8363, 8373, 8421, 8442,
 8559, 8577, 8579, 8584, 8585, 8622,
 8624, 8630, 8632, 8696, 8772, 8780,
 8927, 8949, 8969, 8977, 8984, 8991,
 9000, 9008, 9009, 9053, 9687, 9702,
 9703, 9709, 9722, 9735, 9741, 13203
`\{` 3, 5356, 5910, 6273, 6275, 8357
`\}` ... 4, 5356, 5910, 6273, 6275, 8359, 8654
`\^` 6, 9, 249, 3078, 3085, 8322, 9058
`_` 2297
`\|` 3893
`\~` 8361

Numbers

`\8` 9115
`\9` 9116
`_` . 346, 1060, 8316, 8364, 8654, 9010, 9059

A

`\A` 2682, 2724, 4349, 4351
`\above` 467
`\abovedisplayshortskip` 480
`\abovedisplayskip` 481
`\abovewithdelims` 468
`\accent` 518
`\adjdemerits` 555
`\advance` 362
`\afterassignment` 372
`\aftergroup` 373
`\alloc_reg:nNN` 8069, 8072
`\alloc_setup_type:nnn` 8067, 8070
`\AtBeginDocument` 9860
`\atop` 469
`\atopwithdelims` 470

B

`\B` 4350, 4352
`\badness` 617
`\baselineskip` 545
`\batchmode` 438
`\begin` 8618
`\BeginCatcodeRegime` 13320
`\begingroup` . 12, 61, 100, 228, 232, 338, 376
`\beginL` 730
`\beginR` 732
`\belowdisplayshortskip` 482
`\belowdisplayskip` 483
`\binoppenalty` 506
`\bool(:w` 1936
`\bool_)_0:w` 1936
`\bool_)_1:w` 1936
`\bool_8_0:w` 1936
`\bool_8_1:w` 1936
`\bool_:w` 1936
`\bool_choose:NN` 1936, 2025, 2033
`\bool_cleanup:N` .. 1936, 2018, 2021, 2023
`\bool_do_until:cn` 2081
`\bool_do_until:Nn` 2081, 2083, 2084, 2086
`\bool_do_until:nn` 2087, 2108, 2111
`\bool_do_while:cn` 2081
`\bool_do_while:Nn` 2081, 2081, 2082, 2085
`\bool_do_while:nn` 2087, 2095, 2098
`\bool_eval_skip_to_end:Nw` 1936,
 2045, 2046, 2048, 2050, 2052, 2066
`\bool_eval_skip_to_end_aux:Nw`
 1936, 2054, 2058
`\bool_eval_skip_to_end_aux_ii:Nw` ...
 1936, 2062, 2064
`\bool_get_next:N` ... 1936, 1947, 1949,
 1969, 1998, 2018, 2020, 2035–2038
`\bool_get_next:NN` 1969, 1977
`\bool_get_not_next:N` ... 1959, 1970, 1997
`\bool_get_not_next:NN` 1970, 1990
`\bool_gset:cn` 1916
`\bool_gset:Nn` 36, 1916, 1918, 1921
`\bool_gset_eq:cc` 1908, 1915
`\bool_gset_eq:cN` 1908, 1914
`\bool_gset_eq:Nc` 1908, 1913
`\bool_gset_eq:NN` 35, 1908, 1912
`\bool_gset_false:c` 1896
`\bool_gset_false:N` .. 35, 1896, 1902, 1907
`\bool_gset_true:c` 1896
`\bool_gset_true:N` .. 35, 1896, 1900, 1906
`\bool_I_0:w` 1936
`\bool_I_1:w` 1936

\bool_if:c	1922	\botmark	453
\bool_if:N	1922, 1922	\botmarks	679
\bool_if:n	1936, 1936	\box	661
\bool_if:NF	294, 1932, 2078, 2084, 3723, 7877, 9649	\box_clear:c	6375
\bool_if:nF	2102, 2111	\box_clear:N	122, 6375, 6375, 6379, 6980, 7036, 7081
\bool_if:NT	1931, 2076, 2082, 3723, 3724, 7243, 9614, 10708	\box_clear_new:c	6381
\bool_if:nT	2089, 2098	\box_clear_new:N	122, 6381, 6381, 6393
\bool_if:NTF	36, 1933, 3709, 8404, 9275, 10721	\box_clip:c	6847
\bool_if:nTF	37, 2948, 3098, 4155, 9589, 9599	\box_clip:N	126, 6847, 6847, 6849
\bool_if_p:N	1930	\box_dp:c	6407, 7101
\bool_if_p:n	1917, 1919, 1938, 1944, 2068, 2071	\box_dp:N	124, 6407, 6408, 6411, 6414, 6590, 6703, 6750, 6771, 6793, 6858, 6859, 6863, 7100, 7148, 7149, 7197, 7199, 7216, 7230, 7392, 7410, 7558, 7947, 7961
\bool_new:c	1894	\box_gclear:c	6375
\bool_new:N	35, 1894, 1894, 1895, 1934, 1935, 6937, 8313, 9223, 9292, 9307, 9931	\box_gclear:N	122, 6375, 6377, 6380
\bool_Not:N	1979, 1999	\box_gclear_new:c	6381
\bool_Not:w	1936, 1974, 1997	\box_gclear_new:N	122, 6381, 6387, 6394
\bool_not_choose:NN	2030, 2034	\box_gset_eq:cc	6395
\bool_not_cleanup:N	2020, 2022, 2028	\box_gset_eq:cN	6395
\bool_not_Not:N	1992, 2008	\box_gset_eq:Nc	6395
\bool_not_Not:w	1987, 1998	\box_gset_eq:NN	122, 6378, 6390, 6395, 6397, 6400
\bool_not_p:n	37, 2068, 2068	\box_gset_eq_clear:cc	6401
\bool_p:w	1936, 2001, 2010	\box_gset_eq_clear:cN	6401
\bool_S_0:w	1936	\box_gset_eq_clear:Nc	6401
\bool_S_1:w	1936	\box_gset_eq_clear:NN	123, 6401, 6403, 6406
\bool_set:cn	1916	\box_gset_to_last:c	6455
\bool_set:Nn	35, 1916, 1916, 1920	\box_gset_to_last:N	127, 6455, 6457, 6460
\bool_set_eq:cc	1908, 1911	\box_ht:c	6407, 7103
\bool_set_eq:cN	1908, 1910	\box_ht:N	124, 6407, 6407, 6410, 6416, 6589, 6702, 6749, 6770, 6792, 6868, 6869, 6873, 7032, 7076, 7102, 7147, 7149, 7193, 7195, 7216, 7223, 7391, 7409, 7555, 7557, 7945, 7960
\bool_set_eq:Nc	1908, 1909	\box_if_empty:c	6449
\bool_set_eq:NN	35, 1908, 1908	\box_if_empty:N	6449, 6449
\bool_set_false:c	1896	\box_if_empty:NF	6453
\bool_set_false:N	35, 309, 1896, 1898, 1905, 7239, 7875, 8406, 9244, 9581, 10698, 10727	\box_if_empty:NT	6452
\bool_set_true:c	1896	\box_if_empty:NTF	126, 6454
\bool_set_true:N	35, 323, 1896, 1896, 1904, 7257, 7272, 7305, 8355, 8445, 9239, 9576, 10735	\box_if_empty_p:N	6451
\bool_until_do:cn	2075	\box_if_horizontal:c	6437
\bool_until_do:Nn	37, 2075, 2077, 2078, 2080	\box_if_horizontal:N	6437, 6437
\bool_until_do:nn	38, 2087, 2100, 2105	\box_if_horizontal:NF	6443
\bool_while_do:cn	2075	\box_if_horizontal:NT	6442
\bool_while_do:Nn	37, 2075, 2075, 2076, 2079	\box_if_horizontal:NTF	126, 6444
\bool_while_do:nn	38, 2087, 2087, 2092	\box_if_horizontal_p:N	6441
\bool_xor_p:nn	37, 2069, 2069	\box_if_vertical:c	6437

- \box_if_vertical:N [6437](#), [6439](#)
 - \box_if_vertical:NF [6447](#)
 - \box_if_vertical:NT [6446](#)
 - \box_if_vertical:NTF [127](#), [6448](#)
 - \box_if_vertical_p:N [6445](#)
 - \box_move_down:nn
..... [123](#), [6426](#), [6432](#), [6863](#), [6890](#), [7538](#)
 - \box_move_left:nn [123](#), [6426](#), [6426](#)
 - \box_move_right:nn [123](#), [6426](#), [6428](#)
 - \box_move_up:nn
..... [123](#), [6426](#), [6430](#), [6873](#), [6898](#), [7430](#), [7942](#)
 - \box_new:c [6367](#)
 - \box_new:N
..... [122](#), [6367](#), [6368](#), [6374](#), [6385](#), [6391](#),
..... [6465](#), [6471](#), [6473](#), [6564](#), [6911](#), [6987](#)
 - \box_resize:cnn [6697](#)
 - \box_resize:Nnn [125](#), [6697](#), [6697](#), [6723](#), [7653](#)
 - \box_resize_aux:Nnn
.. [6697](#), [6717](#), [6719](#), [6724](#), [6760](#), [6780](#)
 - \box_resize_common:N [6742](#), [6821](#), [6823](#), [6823](#)
 - \box_resize_to_ht_plus_dp:cn [6744](#)
 - \box_resize_to_ht_plus_dp:Nn
..... [125](#), [6744](#), [6744](#), [6764](#)
 - \box_resize_to_wd:cn [6744](#)
 - \box_resize_to_wd:Nn [125](#), [6744](#), [6765](#), [6784](#)
 - \box_rotate:Nn [125](#), [6570](#), [6570](#), [7533](#)
 - \box_rotate_aux:N [6570](#), [6581](#), [6583](#), [6587](#)
 - \box_rotate_quadrant_four:
..... [6570](#), [6602](#), [6684](#)
 - \box_rotate_quadrant_one: [6570](#), [6596](#), [6651](#)
 - \box_rotate_quadrant_three:
..... [6570](#), [6601](#), [6673](#)
 - \box_rotate_quadrant_two: [6570](#), [6597](#), [6662](#)
 - \box_rotate_set_sin_cos: [6570](#), [6576](#), [6621](#)
 - \box_rotate_x:nnN
..... [6570](#), [6629](#), [6657](#), [6659](#),
..... [6668](#), [6670](#), [6679](#), [6681](#), [6690](#), [6692](#)
 - \box_rotate_y:nnN
..... [6570](#), [6640](#), [6653](#), [6655](#),
..... [6664](#), [6666](#), [6675](#), [6677](#), [6686](#), [6688](#)
 - \box_scale:cnn [6785](#)
 - \box_scale:Nnn [125](#), [6785](#), [6785](#), [6806](#), [7680](#)
 - \box_scale_aux:Nnn [6785](#), [6800](#), [6802](#), [6807](#)
 - \box_set_dp:cn [6413](#)
 - \box_set_dp:Nn [124](#), [6413](#),
..... [6413](#), [6420](#), [6616](#), [6832](#), [6859](#), [6866](#),
..... [6891](#), [6893](#), [7392](#), [7410](#), [7543](#), [7946](#)
 - \box_set_eq:cc [6395](#)
 - \box_set_eq:cN [6395](#)
 - \box_set_eq:Nc [6395](#)
 - \box_set_eq:NN . [122](#), [6376](#), [6384](#), [6395](#),
..... [6395](#), [6398](#), [6399](#), [7091](#), [7412](#), [7950](#)
 - \box_set_eq_clear:cc [6401](#)
 - \box_set_eq_clear:cN [6401](#)
 - \box_set_eq_clear:Nc [6401](#)
 - \box_set_eq_clear:NN
..... [123](#), [6401](#), [6401](#), [6404](#), [6405](#)
 - \box_set_ht:cn [6413](#)
 - \box_set_ht:Nn [124](#), [6413](#),
..... [6415](#), [6419](#), [6615](#), [6831](#), [6869](#), [6876](#),
..... [6895](#), [6899](#), [7391](#), [7409](#), [7541](#), [7944](#)
 - \box_set_to_last:c [6455](#)
 - \box_set_to_last:N
..... [127](#), [6455](#), [6455](#), [6458](#), [6459](#)
 - \box_set_wd:cn [6413](#)
 - \box_set_wd:Nn
..... [124](#), [6413](#), [6417](#), [6421](#), [6617](#), [6843](#),
..... [6852](#), [6882](#), [7393](#), [7411](#), [7544](#), [7948](#)
 - \box_show:c [6474](#)
 - \box_show:N [127](#), [6474](#), [6474](#), [6475](#)
 - \box_trim:cnnnn [6850](#)
 - \box_trim:Nnnnn ... [126](#), [6850](#), [6850](#), [6879](#)
 - \box_use:c [6422](#)
 - \box_use:N [123](#), [6422](#), [6423](#), [6425](#),
..... [6580](#), [6604](#), [6611](#), [6619](#), [6716](#), [6759](#),
..... [6779](#), [6799](#), [6828](#), [6838](#), [6844](#), [6856](#),
..... [6864](#), [6874](#), [6886](#), [6890](#), [6898](#), [7427](#),
..... [7430](#), [7508](#), [7539](#), [7869](#), [7939](#), [7942](#)
 - \box_use_clear:c [6422](#)
 - \box_use_clear:N .. [123](#), [6422](#), [6422](#), [6424](#)
 - \box_viewport:cnnnn [6880](#)
 - \box_viewport:Nnnnn [126](#), [6880](#), [6880](#), [6902](#)
 - \box_wd:c [6407](#), [7105](#)
 - \box_wd:N [124](#), [6407](#), [6409](#), [6412](#),
..... [6418](#), [6591](#), [6704](#), [6751](#), [6772](#), [6794](#),
..... [6852](#), [7104](#), [7150](#), [7195](#), [7199](#), [7205](#),
..... [7210](#), [7360](#), [7393](#), [7411](#), [7428](#), [7557](#),
..... [7562](#), [7722](#), [7729](#), [7940](#), [7949](#), [7962](#)
 - \boxmaxdepth [623](#)
 - \brokenpenalty [580](#)
- C**
- \C [1841](#), [2724](#)
 - \c_active_char_token [3147](#), [3148](#)
 - \c_alignment_tab_token [3141](#), [3142](#)
 - \c_alignment_token
..... [50](#), [2588](#), [2594](#), [2624](#), [3142](#)
 - \c_catcode_active_tl
..... [50](#), [2603](#), [2605](#), [2662](#), [3148](#)

\c_catcode_letter_token	\c_fp_exp_-6_tl	12052
..... 50 , 2588 , 2600 , 2652 , 3144	\c_fp_exp_-70_tl	12052
\c_catcode_other_space_tl	\c_fp_exp_-7_tl	12052
141 , 8314 , 8317 , 8327 , 8329 , 8331 , 8364	\c_fp_exp_-80_tl	12052
\c_catcode_other_token	\c_fp_exp_-8_tl	12052
..... 50 , 2588 , 2601 , 2657 , 3145	\c_fp_exp_-90_tl	12052
\c_code_cctab ... 175 , 13334 , 13336 , 13337	\c_fp_exp_-9_tl	12052
\c_coffin_corners_prop	\c_fp_exp_100_tl	12032
..... 6915 , 6915-6919 , 6991 , 7119	\c_fp_exp_10_tl	12032
\c_coffin_poles_prop 6920 , 6920 ,	\c_fp_exp_1_tl	12032
6922-6924 , 6926-6931 , 6993 , 7121	\c_fp_exp_200_tl	12032
\c_document_cctab	\c_fp_exp_20_tl	12032
..... 176 , 13334 , 13342 , 13351 , 13354	\c_fp_exp_2_tl	12032
\c_e_fp	\c_fp_exp_30_tl	12032
..... 172 , 9890 , 9890	\c_fp_exp_3_tl	12032
\c_eight	\c_fp_exp_40_tl	12032
..... 67 , 2516 ,	\c_fp_exp_4_tl	12032
2548 , 3785 , 3848 , 3853 , 8046 , 10699	\c_fp_exp_50_tl	12032
\c_eleven 67 , 2522 , 2554 , 3848 , 3856 , 8049	\c_fp_exp_5_tl	12032
\c_empty_box	\c_fp_exp_60_tl	12032
..... 127 , 6376 , 6378 ,	\c_fp_exp_6_tl	12032
6384 , 6390 , 6461 , 6462 , 6465 , 7507	\c_fp_exp_70_tl	12032
\c_empty_coffin .. 7096 , 7096 , 7097 , 7505	\c_fp_exp_7_tl	12032
\c_empty_prop . 120 , 6047 , 6047-6053 , 6166	\c_fp_exp_80_tl	12032
\c_empty_tl	\c_fp_exp_8_tl	12032
..... 94 , 3598 , 4247 , 4263 ,	\c_fp_exp_90_tl	12032
4265 , 4488 , 4827 , 4827 , 5248 , 5362	\c_fp_exp_9_tl	12032
\c_false_bool	\c_fp_ln_10_1_tl	12378
..... 20 ,	\c_fp_ln_10_2_tl	12378
987 , 1016 , 1056 , 1057 , 1085 , 1453 ,	\c_fp_ln_10_3_tl	12378
1455 , 1464 , 1476 , 1894 , 1899 , 1903 ,	\c_fp_ln_10_4_tl	12378
2003 , 2014 , 2039 , 2042 , 2043 , 2045 ,	\c_fp_ln_10_5_tl	12378
2048 , 2072 , 2768 , 3698 , 3703 , 3712	\c_fp_ln_10_6_tl	12378
\c_fifteen 67 , 2530 , 2562 , 3848 , 3859 , 8053	\c_fp_ln_10_7_tl	12378
\c_five	\c_fp_ln_10_8_tl	12378
..... 67 , 2510 , 2542 ,	\c_fp_ln_10_9_tl	12378
3848 , 3852 , 8043 , 10352 , 11609 , 11907	\c_fp_ln_2_1_tl	12387
\c_five_hundred_million . 9873 , 9876 ,	\c_fp_ln_2_2_tl	12387
10389 , 12298 , 12454 , 12649 , 12651	\c_fp_ln_2_3_tl	12387
\c_forty_four	\c_fp_pi_by_four_decimal_int	
..... 9873 , 9873 , 10517 9878 , 9878 , 9879 ,	
\c_four 67 , 2508 , 2540 , 3848 , 3851 , 8042 ,	11537 , 11548 , 11560 , 11567 , 11571	
8449 , 8455 , 10498 , 10734 , 11546 , 11547	\c_fp_pi_by_four_extended_int 9878 ,	
\c_fourteen 67 , 2528 , 2560 , 3848 , 3858 , 8052	9880 , 9881 , 11537 , 11549 , 11560 , 11572	
\c_fp_exp_-100_tl	\c_fp_pi_decimal_int	
..... 12052 9878 , 9882 , 9883 , 11477	
\c_fp_exp_-10_tl	\c_fp_pi_extended_int .. 9878 , 9884 , 9885	
..... 12052	\c_fp_two_pi_decimal_int	
\c_fp_exp_-1_tl 9878 , 9886 , 9887 , 11473 , 11479	
..... 12052		
\c_fp_exp_-200_tl		
..... 12052		
\c_fp_exp_-20_tl		
..... 12052		
\c_fp_exp_-2_tl		
..... 12052		
\c_fp_exp_-30_tl		
..... 12052		
\c_fp_exp_-3_tl		
..... 12052		
\c_fp_exp_-40_tl		
..... 12052		
\c_fp_exp_-4_tl		
..... 12052		
\c_fp_exp_-50_tl		
..... 12052		
\c_fp_exp_-5_tl		
..... 12052		
\c_fp_exp_-60_tl		
..... 12052		

- \c_fp_two_pi_extended_int
..... [9878](#), [9888](#), [9889](#), [11473](#), [11479](#)
- \c_group_begin_token
..... [50](#), [2588](#), [2588](#), [2609](#),
[2950](#), [3100](#), [4735](#), [4769](#), [6488](#), [6534](#)
- \c_group_end_token [50](#), [2588](#), [2589](#), [2614](#),
[2951](#), [3101](#), [6493](#), [6494](#), [6539](#), [6540](#)
- \c_initex_cctab [176](#), [13334](#), [13343](#)
- \c_int_from_roman_C_int [3786](#)
- \c_int_from_roman_c_int [3786](#)
- \c_int_from_roman_D_int [3786](#)
- \c_int_from_roman_d_int [3786](#)
- \c_int_from_roman_I_int [3786](#)
- \c_int_from_roman_i_int [3786](#)
- \c_int_from_roman_L_int [3786](#)
- \c_int_from_roman_l_int [3786](#)
- \c_int_from_roman_M_int [3786](#)
- \c_int_from_roman_m_int [3786](#)
- \c_int_from_roman_V_int [3786](#)
- \c_int_from_roman_v_int [3786](#)
- \c_int_from_roman_X_int [3786](#)
- \c_int_from_roman_x_int [3786](#)
- \c_ior_log_stream [8032](#), [8035](#)
- \c_ior_streams_tl [8036](#), [8055](#), [8092](#)
- \c_ior_term_stream [8032](#), [8033](#)
- \c_iow_log_stream [8032](#), [8034](#), [8292](#), [8293](#)
- \c_iow_streams_tl [8036](#), [8036](#), [8055](#), [8105](#)
- \c_iow_term_stream [8032](#), [8032](#), [8294](#), [8295](#)
- \c_iow_wrap_end_marker_tl ... [8319](#), [8378](#)
- \c_iow_wrap_indent_marker_tl [8319](#), [8342](#)
- \c_iow_wrap_marker_tl
..... [8319](#), [8321](#), [8328](#), [8388](#), [8435](#)
- \c_iow_wrap_newline_marker_tl [8319](#), [8363](#)
- \c_iow_wrap_unindent_marker_tl
..... [8319](#), [8344](#)
- \c_job_name_tl [94](#), [4815](#), [4826](#)
- \c_keys_code_root_tl [9214](#), [9214](#), [9386](#),
[9391](#), [9655](#), [9657](#), [9669](#), [9675](#), [9680](#)
- \c_keys_props_root_tl
..... [9216](#), [9216](#), [9249](#), [9279](#),
[9286](#), [9456](#), [9458](#), [9460](#), [9462](#), [9464](#),
[9466](#), [9468](#), [9470](#), [9472](#), [9474](#), [9476](#),
[9478](#), [9480](#), [9482](#), [9484](#), [9486](#), [9488](#),
[9490](#), [9492](#), [9494](#), [9496](#), [9498](#), [9500](#),
[9502](#), [9504](#), [9506](#), [9508](#), [9510](#), [9512](#),
[9514](#), [9516](#), [9518](#), [9520](#), [9522](#), [9524](#),
[9526](#), [9528](#), [9530](#), [9532](#), [9534](#), [9536](#),
[9538](#), [9540](#), [9542](#), [9544](#), [9546](#), [9548](#)
- \c_keys_value_forbidden_tl .. [9217](#), [9217](#)
- \c_keys_value_required_tl ... [9217](#), [9218](#)
- \c_keys_vars_root_tl [9214](#), [9215](#), [9349](#),
[9368](#), [9375](#), [9378](#), [9380](#), [9395](#)–[9397](#),
[9400](#), [9443](#), [9616](#), [9618](#), [9621](#), [9629](#)
- \c_letter_token [3141](#), [3144](#)
- \c_luatex_is_engine_bool [1521](#), [1521](#)
- \c_math_shift_token [3141](#), [3143](#)
- \c_math_subscript_token
..... [50](#), [2588](#), [2598](#), [2642](#)
- \c_math_superscript_token
..... [50](#), [2588](#), [2596](#), [2637](#)
- \c_math_toggle_token
..... [50](#), [2588](#), [2592](#), [2619](#), [3143](#)
- \c_max_dim [75](#), [4096](#), [4098](#),
[4099](#), [4103](#), [4178](#), [7607](#)–[7610](#), [7623](#)
- \c_max_int [67](#), [3866](#), [3866](#)
- \c_max_register_int
..... [67](#), [1179](#), [1179](#), [3299](#), [13270](#)
- \c_max_skip [78](#), [4177](#), [4178](#)
- \c_minus_one [67](#), [1167](#), [1168](#), [1171](#), [1172](#),
[1181](#), [3297](#), [3341](#), [3848](#), [4367](#), [4368](#),
[4393](#), [4394](#), [4406](#), [4407](#), [8034](#), [8035](#),
[8210](#), [8223](#), [8320](#), [8356](#), [9948](#), [10056](#),
[10485](#), [10739](#), [10740](#), [10877](#), [10912](#),
[11155](#), [11203](#), [11207](#), [11372](#), [11496](#),
[11500](#), [11765](#), [11780](#), [11868](#), [11872](#),
[11945](#), [11953](#), [12361](#), [12365](#), [12489](#)
- \c_msg_coding_error_text_tl
..... [7989](#), [8000](#), [8556](#), [8556](#),
[8967](#), [8976](#), [8998](#), [9006](#), [9015](#), [9022](#),
[9029](#), [9036](#), [9693](#), [9700](#), [9721](#), [9728](#)
- \c_msg_continue_text_tl [8556](#), [8561](#), [8624](#)
- \c_msg_critical_text_tl [8556](#), [8563](#), [8743](#)
- \c_msg_fatal_text_tl [8556](#), [8565](#), [8732](#), [8874](#)
- \c_msg_help_text_tl [8556](#), [8567](#), [8632](#)
- \c_msg_hide_tl [8597](#), [8599](#)–[8601](#), [8669](#)
- \c_msg_hide_tl<dots> [8597](#)
- \c_msg_kernel_bug_more_text_tl
..... [9040](#), [9047](#), [9051](#)
- \c_msg_kernel_bug_text_tl [9040](#), [9042](#), [9049](#)
- \c_msg_more_text_prefix_tl [8525](#), [8526](#),
[8542](#), [8551](#), [8748](#), [8756](#), [8887](#), [8897](#)
- \c_msg_no_info_text_tl . [8556](#), [8569](#), [8622](#)
- \c_msg_on_line_text_tl [8574](#), [8593](#)
- \c_msg_on_line_tl [8556](#)
- \c_msg_return_text_tl [145](#), [8556](#), [8572](#),
[8575](#), [8971](#), [8979](#), [8986](#), [8993](#), [9055](#)
- \c_msg_text_prefix_tl
..... [8525](#), [8525](#), [8529](#), [8540](#), [8549](#), [8729](#),
[8740](#), [8753](#), [8762](#), [8773](#), [8781](#), [8787](#),
[8817](#), [8870](#), [8892](#), [8905](#), [8928](#), [8950](#)

- \c_msg_trouble_text_tl . 145, 8556, 8582
- \c_nine 67, 2518, 2550, 3848,
3854, 8047, 10061, 11409, 11641,
11727, 11973, 11982, 12201, 12493
- \c_one 67, 2502, 2534, 3339,
3848, 3848, 8039, 9950, 9961, 10020,
10032, 10080, 10086, 10128, 10153,
10350, 10709, 10713, 10715, 10722,
10862, 10891, 11151, 11188, 11193,
11214, 11337, 11405, 11448, 11462,
11513, 11528, 11563, 11575, 11636,
11722, 11770, 11777, 11792, 11818,
11840, 11845, 11853, 11859, 11947,
11951, 12153, 12163, 12264, 12289,
12300, 12304, 12326, 12346, 12352,
12456, 12460, 12491, 12508, 12560,
12583, 12589, 12590, 12634, 12652,
12690, 12711, 12717, 12871, 12873
- \c_one_fp . 172, 6579, 6713, 6715, 6758,
6778, 6796, 6798, 9891, 9891, 12770
- \c_one_hundred 67, 3863, 3863, 10083, 10084, 12162
- \c_one_hundred_million 9873, 9875, 11072, 12004
- \c_one_million 9873, 9874, 11335
- \c_one_thousand 67,
3863, 3864, 10982, 11238, 11282, 11333
- \c_one_thousand_million 9873, 9877, 10023,
10045, 10062, 10072, 10108, 10119,
10133, 10144, 10185, 10227, 10501,
10520, 10639, 10675, 10701, 10752,
10777, 10843, 10860, 10863, 10878,
10887, 10964, 11003, 11011, 11020,
11107, 11120, 11156, 11186, 11189,
11191, 11194, 11204, 11208, 11215,
11216, 11336, 11338, 11350, 11362,
11377, 11410, 11421, 11439, 11497,
11501, 11517, 11521, 11605, 11660,
11702, 11746, 11797, 11803, 11851,
11855, 11857, 11861, 11869, 11873,
11903, 12027, 12097, 12272, 12282,
12301, 12319, 12344, 12348, 12350,
12354, 12362, 12366, 12436, 12457,
12479, 12531, 12598, 12601, 12670,
12709, 12713, 12715, 12719, 12863
- \c_other_cctab 176, 13334, 13344, 13352, 13362
- \c_other_char_token 3141, 3145
- \c_parameter_token 50, 2588, 2595, 2628, 2631
- \c_pdftex_is_engine_bool 1521, 1522
- \c_pi_fp 172, 6625, 7521, 9892, 9892
- \c_seven 67, 1167, 1177, 2514, 2546, 3848, 8045
- \c_six 67, 1167, 1176,
2512, 2544, 3848, 8044, 11473, 11479
- \c_sixteen 67, 1167, 1174,
1183, 3783, 3848, 8032, 8033, 8067,
8070, 8091, 8093, 8104, 8106, 8139,
8158, 8176, 8195, 8212, 8225, 8487
- \c_space_tl 94, 4828, 4828, 4890,
5355, 5909, 6272, 6274, 7981, 7982,
8251, 8252, 8379, 8451, 8594, 9841
- \c_space_token 50, 2588, 2599,
2647, 2952, 2971, 3102, 4736, 4770
- \c_string_cctab 176, 13334, 13345, 13353, 13367
- \c_ten 67,
2520, 2552, 3623, 3848, 3855, 8048,
10073, 10120, 10145, 10297, 10361,
10417, 10519, 10524, 10636, 10672,
10711, 10714, 10724, 11119, 11173,
11349, 11398, 11440, 11452, 11530,
11932, 12553, 12786, 12820, 12825
- \c_ten_thousand 67, 3863, 3865
- \c_thirteen 67, 2526, 2558, 3848, 3857, 8051
- \c_thirty_two 67, 3860, 3860
- \c_three 67, 2506, 2538,
3848, 3850, 8041, 11471, 11789, 12122
- \c_tl_act_lowercase_tl . 4955, 4960, 4968
- \c_tl_act_uppercase_tl . 4955, 4955, 4966
- \c_tl_rescan_marker_tl 4348, 4356, 4366, 4378, 4405
- \c_token_A_int 2813, 2848
- \c_true_bool 20, 987,
1016, 1056, 1056, 1089, 1308, 1454,
1465, 1475, 1897, 1901, 1924, 2002,
2005, 2011, 2012, 2040, 2041, 2044,
2046, 2050, 2073, 3698, 3703, 3716
- \c_twelve 67, 1167, 1178, 2284,
2298, 2524, 2556, 2725, 3848, 8050
- \c_two 67, 2504, 2536, 3781,
3848, 3849, 8040, 10488, 11476,
11764, 11768, 11787, 11821, 11944,
11950, 12603–12605, 12618, 12694
- \c_two_hundred_fifty_five 67, 3861, 3861
- \c_two_hundred_fifty_six . 67, 3861, 3862
- \c_undefined:D 1294, 1302

- `\c_undefined_fp` [172](#), [9893](#), [9893](#), [11050](#),
[11960](#), [12753](#), [12803](#), [12810](#), [12930](#)
- `\c_xetex_is_engine_bool` [1521](#), [1523](#)
- `\c_zero` [67](#), [986](#), [994](#), [1002](#), [1009](#),
[1015](#), [1023](#), [1031](#), [1038](#), [1167](#), [1175](#),
[1482](#), [1487](#), [1579](#), [1588](#), [2113](#), [2204](#)–
[2214](#), [2217](#), [2220](#), [2278](#), [2280](#), [2427](#),
[2434](#), [2465](#), [2474](#), [2500](#), [2532](#), [2696](#),
[3228](#), [3258](#), [3262](#), [3263](#), [3269](#), [3316](#),
[3317](#), [3596](#), [3848](#), [4088](#), [4147](#), [4157](#),
[4158](#), [4800](#), [4837](#), [4882](#), [5444](#), [5457](#),
[5948](#), [5963](#), [5983](#), [8038](#), [8067](#), [8070](#),
[8509](#), [8511](#), [9073](#), [9970](#), [9981](#), [10019](#),
[10031](#), [10033](#), [10044](#), [10087](#)–[10089](#),
[10191](#), [10233](#), [10276](#), [10279](#), [10341](#),
[10408](#), [10543](#)–[10551](#), [10582](#)–[10590](#),
[10645](#), [10681](#), [10725](#), [10780](#), [10818](#),
[10834](#), [10876](#), [10880](#), [10882](#), [10948](#),
[10952](#), [10977](#), [11046](#), [11056](#), [11069](#),
[11070](#), [11091](#), [11095](#), [11116](#), [11125](#),
[11126](#), [11154](#), [11202](#), [11206](#), [11210](#),
[11211](#), [11230](#), [11274](#), [11348](#), [11376](#),
[11387](#), [11395](#), [11453](#), [11456](#), [11463](#)–
[11465](#), [11495](#), [11499](#), [11503](#), [11508](#),
[11531](#), [11532](#), [11537](#), [11556](#), [11560](#),
[11571](#), [11596](#), [11637](#), [11651](#), [11693](#),
[11723](#), [11737](#), [11763](#), [11776](#), [11778](#),
[11790](#), [11791](#), [11793](#), [11794](#), [11796](#),
[11798](#), [11801](#), [11812](#)–[11814](#), [11846](#),
[11847](#), [11867](#), [11871](#), [11894](#), [11943](#),
[11957](#), [11958](#), [11988](#), [11989](#), [12001](#),
[12002](#), [12018](#), [12085](#), [12088](#), [12124](#),
[12150](#), [12154](#)–[12156](#), [12164](#)–[12166](#),
[12178](#), [12211](#), [12229](#), [12230](#), [12262](#),
[12263](#), [12268](#), [12269](#), [12274](#), [12303](#),
[12339](#), [12340](#), [12360](#), [12364](#), [12403](#),
[12407](#), [12459](#), [12470](#), [12487](#), [12497](#)–
[12500](#), [12516](#), [12542](#), [12550](#), [12564](#),
[12565](#), [12567](#), [12570](#), [12616](#), [12617](#),
[12620](#), [12625](#), [12627](#), [12639](#), [12656](#),
[12663](#), [12669](#), [12679](#), [12687](#), [12702](#),
[12745](#), [12749](#), [12766](#), [12781](#), [12785](#),
[12792](#), [12817](#), [12822](#), [12824](#), [12874](#),
[12875](#), [12903](#), [12921](#)–[12924](#), [13011](#),
[13014](#), [13018](#), [13029](#), [13030](#), [13052](#)
- `\c_zero_dim` [75](#), [3932](#), [3973](#), [4096](#),
[4097](#), [4102](#), [4177](#), [6516](#), [6726](#), [6728](#),
[6729](#), [6866](#), [6876](#), [6888](#), [6891](#), [6894](#),
[6899](#), [7253](#), [7256](#), [7259](#), [7268](#), [7271](#),
[7274](#), [7283](#), [7290](#), [7356](#), [7361](#), [7368](#)
- `\c_zero_fp` [172](#), [6577](#), [6593](#), [6595](#), [6600](#),
[6809](#), [6818](#), [6833](#), [7669](#), [7684](#), [7687](#),
[9894](#), [9894](#), [10159](#), [10169](#), [10171](#),
[11060](#), [11936](#), [11991](#), [12114](#), [12141](#),
[12181](#), [12413](#), [12421](#), [12759](#), [12938](#)
- `\c_zero_muskip` [4193](#)
- `\c_zero_skip` [78](#), [4118](#), [4177](#), [4177](#), [4228](#), [4229](#), [6503](#)
- `\catcode` [3](#)–
[6](#), [9](#), [70](#)–[78](#), [84](#)–[91](#), [101](#), [281](#)–[288](#), [665](#)
- `\catcodetable` [758](#)
- `\CatcodeTableIniTeX` [13343](#)
- `\CatcodeTableLaTeX` [13342](#)
- `\CatcodeTableOther` [13344](#)
- `\CatcodeTableString` [13345](#)
- `\cctab_begin:N` [175](#), [13294](#), [13294](#), [13314](#), [13320](#)
- `\cctab_end` [175](#)
- `\cctab_end:...` [13294](#), [13304](#), [13315](#), [13321](#)
- `\cctab_gset:Nn` [175](#), [13325](#), [13325](#),
[13333](#), [13337](#), [13354](#), [13362](#), [13367](#)
- `\cctab_new:N` [175](#), [13264](#), [13264](#),
[13283](#), [13287](#), [13336](#), [13351](#)–[13353](#)
- `\char` [519](#)
- `\char_active_gset:Npn` [57](#), [3091](#)
- `\char_active_gset:Npx` [3092](#)
- `\char_active_gset:eq:NN` [58](#), [3077](#), [3094](#)
- `\char_active_set:Npn` [57](#), [3077](#), [3089](#)
- `\char_active_set:Npx` [3077](#), [3090](#)
- `\char_active_set:eq:NN` [58](#), [3077](#), [3093](#)
- `\char_make_active:N` [3150](#), [3166](#)
- `\char_make_active:n` [3150](#), [3184](#)
- `\char_make_alignment:N` [3150](#)
- `\char_make_alignment:n` [3150](#)
- `\char_make_alignment_tab:N` [3155](#)
- `\char_make_alignment_tab:n` [3173](#)
- `\char_make_begin_group:N` [3152](#)
- `\char_make_begin_group:n` [3170](#)
- `\char_make_comment:N` [3150](#), [3167](#)
- `\char_make_comment:n` [3150](#), [3185](#)
- `\char_make_end_group:N` [3153](#)
- `\char_make_end_group:n` [3171](#)
- `\char_make_end_line:N` [3150](#), [3156](#)
- `\char_make_end_line:n` [3150](#), [3174](#)
- `\char_make_escape:N` [3150](#), [3151](#)
- `\char_make_escape:n` [3150](#), [3169](#)
- `\char_make_group_begin:N` [3150](#)
- `\char_make_group_begin:n` [3150](#)
- `\char_make_group_end:N` [3150](#)
- `\char_make_group_end:n` [3150](#)

- \char_make_ignore:N 3150, 3162
- \char_make_ignore:n 3150, 3180
- \char_make_invalid:N 3150, 3168
- \char_make_invalid:n 3150, 3186
- \char_make_letter:N 3150, 3164
- \char_make_letter:n 3150, 3182
- \char_make_math_shift:N 3154
- \char_make_math_shift:n 3172
- \char_make_math_subscript:N . 3150, 3160
- \char_make_math_subscript:n . 3150, 3178
- \char_make_math_superscript:N 3150, 3158
- \char_make_math_superscript:n 3150, 3176
- \char_make_math_toggle:N 3150
- \char_make_math_toggle:n 3150
- \char_make_other:N 3150, 3165
- \char_make_other:n 3150, 3183
- \char_make_parameter:N 3150, 3157
- \char_make_parameter:n 3150, 3175
- \char_make_space:N 3150, 3163
- \char_make_space:n 3150, 3181
- \char_set_catcode:nn 48, 298–306, 2493, 2493, 2500, 2502, 2504, 2506, 2508, 2510, 2512, 2514, 2516, 2518, 2520, 2522, 2524, 2526, 2528, 2530, 2532, 2534, 2536, 2538, 2540, 2542, 2544, 2546, 2548, 2550, 2552, 2554, 2556, 2558, 2560, 2562, 2725
- \char_set_catcode:w 3113, 3114, 3121, 3123
- \char_set_catcode_active:N 47, 2499, 2525, 2604, 3078, 3166, 8656
- \char_set_catcode_active:n . 47, 2531, 2557, 3083, 3184, 9113, 9114, 13360
- \char_set_catcode_alignment:N 47, 2499, 2507, 2593, 3155
- \char_set_catcode_alignment:n 47, 316, 2531, 2539, 3173
- \char_set_catcode_comment:N 47, 2499, 2527, 3167
- \char_set_catcode_comment:n 47, 2531, 2559, 3185
- \char_set_catcode_end_line:N 47, 2499, 2509, 3156
- \char_set_catcode_end_line:n 47, 2531, 2541, 3174
- \char_set_catcode_escape:N 47, 2499, 2499, 3151
- \char_set_catcode_escape:n 47, 2531, 2531, 3169
- \char_set_catcode_group_begin:N 47, 2499, 2501, 3152
- \char_set_catcode_group_begin:n 47, 2531, 2533, 3170
- \char_set_catcode_group_end:N 47, 2499, 2503, 3153
- \char_set_catcode_group_end:n 47, 2531, 2535, 3171
- \char_set_catcode_ignore:N 47, 2499, 2517, 3162
- \char_set_catcode_ignore:n 47, 313, 314, 2531, 2549, 3180, 9959
- \char_set_catcode_invalid:N 47, 2499, 2529, 3168
- \char_set_catcode_invalid:n 47, 2531, 2561, 3186
- \char_set_catcode_letter:N 47, 2499, 2521, 3164, 8597
- \char_set_catcode_letter:n 47, 317, 319, 2531, 2553, 3182
- \char_set_catcode_math_subscript:N . 47, 2499, 2515, 2597, 3161
- \char_set_catcode_math_subscript:n . 47, 2531, 2547, 3179, 13359
- \char_set_catcode_math_superscript:N 47, 2499, 2513, 3159, 9058
- \char_set_catcode_math_superscript:n 47, 318, 2531, 2545, 3177
- \char_set_catcode_math_toggle:N 47, 2499, 2505, 2591, 3154
- \char_set_catcode_math_toggle:n 47, 2531, 2537, 3172
- \char_set_catcode_other:N 47, 2499, 2523, 2681, 2682, 2815, 3165, 8315, 8602
- \char_set_catcode_other:n 47, 315, 320, 2531, 2555, 3183, 13358, 13365, 13370
- \char_set_catcode_parameter:N 47, 2499, 2511, 3157
- \char_set_catcode_parameter:n 47, 2531, 2543, 3175
- \char_set_catcode_space:N 47, 2499, 2519, 3163
- \char_set_catcode_space:n .. 47, 321, 2531, 2551, 3181, 13356, 13357, 13371
- \char_set_lccode:nn 48, 2563, 2569, 2683–2685, 2718–2723, 2816–2818, 3085, 8316, 8654, 8655, 9059–9062, 9115, 9116
- \char_set_lccode:w 3113, 3116, 3127, 3129
- \char_set_mathcode:nn ... 49, 2563, 2563
- \char_set_mathcode:w 3113, 3115, 3124, 3126

\char_set_sfcode:nn	49, 2563, 2581	\clist_gclear_new:c	5591, 5594
\char_set_sfcode:w	3113, 3118, 3133, 3135	\clist_gclear_new:N	107, 5591, 5593
\char_set_uccode:nn	49, 2563, 2575	\clist_gconcat:ccc	5603
\char_set_uccode:w	3113, 3117, 3130, 3132	\clist_gconcat:NNN	107, 5603, 5605, 5617
\char_show_value_catcode:n	48, 2493, 2497	\clist_get:cn	5692, 6025
\char_show_value_catcode:w	3120, 3122	\clist_get:NN	112, 5692, 5692, 5696, 6024
\char_show_value_lccode:n	48, 2563, 2573	\clist_get_aux:wN	5692, 5693, 5694
\char_show_value_lccode:w	3120, 3128	\clist_gpop:cn	5697
\char_show_value_mathcode:n	49, 2563, 2567	\clist_gpop:NN	113, 5697, 5699, 5714
\char_show_value_mathcode:w	3120, 3125	\clist_gpush:cn	5715, 5727
\char_show_value_sfcode:n	50, 2563, 2585	\clist_gpush:co	5715, 5729
\char_show_value_sfcode:w	3120, 3134	\clist_gpush:cV	5715, 5728
\char_show_value_uccode:n	49, 2563, 2579	\clist_gpush:cx	5715, 5730
\char_show_value_uccode:w	3120, 3131	\clist_gpush:Nn	113, 5715, 5723
\char_tmp:NN	3079, 3089–3094	\clist_gpush:No	5715, 5725
\char_value_catcode:n	48, 298–306, 2493, 2495	\clist_gpush:NV	5715, 5724
\char_value_catcode:w	3120, 3121	\clist_gpush:Nx	5715, 5726
\char_value_lccode:n	48, 2563, 2571	\clist_gput_left:cn	5666, 5727
\char_value_lccode:w	3120, 3127	\clist_gput_left:co	5666, 5729
\char_value_mathcode:n	49, 2563, 2565	\clist_gput_left:cV	5666, 5728
\char_value_mathcode:w	3120, 3124	\clist_gput_left:cx	5666, 5730
\char_value_sfcode:n	49, 2563, 2583	\clist_gput_left:Nn	108, 5666, 5668, 5682, 5683, 5723
\char_value_sfcode:w	3120, 3133	\clist_gput_left:No	5666, 5725
\char_value_uccode:n	49, 2563, 2577	\clist_gput_left:NV	5666, 5724
\char_value_uccode:w	3120, 3130	\clist_gput_left:Nx	5666, 5726
\chardef	80, 93, 96, 328, 354	\clist_gput_right:cn	5684
\chk_if_exist_cs:c	1225, 1233	\clist_gput_right:co	5684
\chk_if_exist_cs:N	23, 1225, 1225, 1234, 1783	\clist_gput_right:cV	5684
\chk_if_free_cs:c	1202, 1223	\clist_gput_right:cx	5684
\chk_if_free_cs:N	23, 1202, 1202, 1212, 1224, 1239, 1283, 3290, 3305, 3927, 4113, 4187, 4246, 4252, 4257, 6370, 8082	\clist_gput_right:Nn	108, 5684, 5686, 5690, 5691
\cleaders	537	\clist_gput_right:No	5684
\clist_clear:c	5587, 5588	\clist_gput_right:NV	5684
\clist_clear:N	106, 5587, 5587, 5740, 5999, 9565	\clist_gput_right:Nx	5684
\clist_clear_new:c	5591, 5592	\clist_gremove_all:cn	5750
\clist_clear_new:N	106, 5591, 5591	\clist_gremove_all:Nn	109, 5750, 5752, 5780, 6029
\clist_concat:ccc	5603	\clist_gremove_duplicates:c	5734
\clist_concat:NNN	107, 5603, 5603, 5616	\clist_gremove_duplicates:N	109, 5734, 5736, 5749
\clist_concat_aux:NNNN	5603, 5604, 5606, 5607	\clist_gremove_element:Nn	6027, 6029
\clist_display:c	6031, 6033	\clist_gset:cn	5660
\clist_display:N	6031, 6032	\clist_gset:co	5660
\clist_gclear:c	5587, 5590	\clist_gset:cV	5660
\clist_gclear:N	106, 5587, 5589, 6001	\clist_gset:cx	5660
		\clist_gset:Nn	107, 5660, 5662, 5665
		\clist_gset:No	5660, 6036
		\clist_gset:NV	5660
		\clist_gset:Nx	5660

\clist_gset_eq:cc	5595, 5602	\clist_length:n	5918, 5927, 5971
\clist_gset_eq:cN	5595, 5601	\clist_length_aux:n	5918, 5923, 5926
\clist_gset_eq:Nc	5595, 5600	\clist_length_n_aux:n	5932, 5936
\clist_gset_eq:NN	107, 5595, 5599, 5737	\clist_map_aux_unbrace:Nw	5873, 5873, 5882
\clist_gset_from_seq:cc	5998	\clist_map_break	111
\clist_gset_from_seq:cN	5998	\clist_map_break:	5886, 5886
\clist_gset_from_seq:Nc	5998	\clist_map_break:n	112, 5886, 5887
\clist_gset_from_seq:NN	114, 5998, 6000, 6021, 6022	\clist_map_function:cN	5813
\clist_gtrim_spaces:c	6035	\clist_map_function:NN	110, 5493, 5503, 5813, 5813, 5827, 5835, 5902, 5923
\clist_gtrim_spaces:N	6035, 6036, 6038	\clist_map_function:nN	5498, 5508, 5813, 5845, 5874, 5874, 9353, 9413
\clist_if_empty:c	5782, 5783	\clist_map_function_aux:Nw	5813, 5817, 5821, 5825, 5932
\clist_if_empty:N	5782, 5782	\clist_map_function_n_aux:Nn	5874, 5876, 5879, 5883
\clist_if_empty:NF	5612, 5767, 5815, 5852	\clist_map_inline:cn	5829
\clist_if_empty:NTF	109, 5673, 5890	\clist_map_inline:Nn	110, 5741, 5829, 5829, 5849
\clist_if_eq:cc	5784, 5787	\clist_map_inline:nn	5829, 5839, 9334, 9428
\clist_if_eq:cN	5784, 5786	\clist_map_variable:cNn	5850
\clist_if_eq:Nc	5784, 5785	\clist_map_variable:NNn	111, 5850, 5850, 5870, 5872
\clist_if_eq:NN	5784, 5784	\clist_map_variable:nNn	5850, 5867
\clist_if_eq:NNTF	110	\clist_map_variable_aux:Nnw	5850, 5855, 5860, 5865
\clist_if_in:cn	5788	\clist_new:c	5585, 5586
\clist_if_in:co	5788	\clist_new:N	106, 5585, 5585, 5733, 5781, 5914-5917
\clist_if_in:cV	5788	\clist_pop:cN	5697
\clist_if_in:Nn	5788, 5788	\clist_pop:NN	112, 5697, 5697, 5713
\clist_if_in:nn	5788, 5792	\clist_pop_aux:NNN	5697, 5698, 5700, 5701
\clist_if_in:NnF	5743, 5806, 5807	\clist_pop_aux:NwNNN	5697
\clist_if_in:nnF	5811	\clist_pop_aux:w	5710, 5712
\clist_if_in:NnT	5804, 5805	\clist_pop_aux:wNN	5697
\clist_if_in:nnT	5810	\clist_pop_aux:wNNN	5703, 5705
\clist_if_in:NnTF	110, 5808, 5809	\clist_push:cn	5715, 5719
\clist_if_in:nnTF	5812	\clist_push:co	5715, 5721
\clist_if_in:No	5788	\clist_push:cV	5715, 5720
\clist_if_in:no	5788	\clist_push:cx	5715, 5722
\clist_if_in:NV	5788	\clist_push:Nn	113, 5715, 5715
\clist_if_in:nV	5788	\clist_push:No	5715, 5717
\clist_if_in_return:nn	5788, 5790, 5795, 5797	\clist_push:NV	5715, 5716
\clist_item:cn	5938	\clist_push:Nx	5715, 5718
\clist_item:Nn	114, 5938, 5938, 5967	\clist_put_aux:NNnnNn	5667, 5669, 5670, 5685, 5687
\clist_item:nn	5968, 5968	\clist_put_left:cn	5666, 5719
\clist_item_aux:nnNn	5938, 5940, 5946, 5970	\clist_put_left:co	5666, 5721
\clist_item_n_aux:nw	5968, 5973, 5976	\clist_put_left:cV	5666, 5720
\clist_item_n_end:n	5968, 5984, 5992	\clist_put_left:cx	5666, 5722
\clist_item_N_loop:nw	5938, 5943, 5961, 5965		
\clist_item_n_loop:nw	5968, 5977, 5978, 5981, 5986		
\clist_item_n_strip:w	5968, 5994, 5997		
\clist_length:c	5918		
\clist_length:N	113, 5918, 5918, 5937, 5941		

- \clist_put_left:Nn 108, 5666, 5666, 5680, 5681, 5715
- \clist_put_left:No 5666, 5717
- \clist_put_left:NV 5666, 5716
- \clist_put_left:Nx 5666, 5718
- \clist_put_right:cn 5684
- \clist_put_right:co 5684
- \clist_put_right:cV 5684
- \clist_put_right:cx 5684
- \clist_put_right:Nn 108, 5684, 5684, 5688, 5689, 5744
- \clist_put_right:No 5684
- \clist_put_right:NV 5684
- \clist_put_right:Nx 5684, 9646
- \clist_remove_all:cn 5750
- \clist_remove_all:Nn 109, 5750, 5750, 5779, 6028
- \clist_remove_all_aux: 5750, 5760, 5764, 5776
- \clist_remove_all_aux:NNn 5750, 5751, 5753, 5754
- \clist_remove_all_aux:w 5750, 5777, 5778
- \clist_remove_duplicates:c 5734
- \clist_remove_duplicates:N 109, 5734, 5734, 5748
- \clist_remove_duplicates_aux:NN 5734, 5735, 5737, 5738
- \clist_remove_element:Nn 6027, 6028
- \clist_set:cn 5660
- \clist_set:co 5660
- \clist_set:cV 5660
- \clist_set:cx 5660
- \clist_set:Nn . 107, 5660, 5660, 5664, 5794
- \clist_set:No 5660, 6035
- \clist_set:NV 5660
- \clist_set:Nx 5660
- \clist_set_eq:cc 5595, 5598
- \clist_set_eq:cN 5595, 5597
- \clist_set_eq:Nc 5595, 5596
- \clist_set_eq:NN 107, 5595, 5595, 5735, 9570
- \clist_set_from_seq:cc 5998
- \clist_set_from_seq:cN 5998
- \clist_set_from_seq:Nc 5998
- \clist_set_from_seq:NN 114, 5998, 5998, 6019, 6020
- \clist_set_from_seq_aux:NNNN 5999, 6001, 6002
- \clist_set_from_seq_aux:w ... 6010, 6018
- \clist_show:c 5888, 6033
- \clist_show:N . 113, 5888, 5888, 5913, 6032
- \clist_show_aux:n 5888, 5902, 5907
- \clist_show_aux:w 5888, 5904, 5912
- \clist_tmp:w 5584, 5584, 5618, 5634, 5756, 5777, 5799, 5801
- \clist_top:cN 6023, 6025
- \clist_top:NN 6023, 6024
- \clist_trim_spaces:c 6035
- \clist_trim_spaces:N ... 6035, 6035, 6037
- \clist_trim_spaces:n 114, 5638, 5638, 5661, 5663, 5672, 5869
- \clist_trim_spaces_aux:n 5638, 5640, 5643, 5653, 5657
- \clist_trim_spaces_aux_ii:nn 5638, 5646, 5648
- \clist_trim_spaces_generic:nw 5618, 5620, 5645, 5876, 5883
- \clist_trim_spaces_generic_aux:w ... 5618, 5629, 5635
- \clist_trim_spaces_generic_aux_ii:nn 5618, 5636, 5637
- \clist_use:c 5731, 5732
- \clist_use:N 108, 5731, 5731
- \closein 413
- \closeout 408
- \clubpenalties 721
- \clubpenalty 548
- \coffin_align:NnnNnnnnN 7352, 7389, 7407, 7415, 7415, 7505
- \coffin_attach:cnncnnnn 7387
- \coffin_attach:cnNnnnnn 7387
- \coffin_attach:Nnncnnnn 7387
- \coffin_attach:NnnNnnnn 135, 7387, 7387, 7414
- \coffin_attach_mark:NnnNnnnn 7387, 7405, 7798, 7819, 7835
- \coffin_calculate_intersection:Nnn . 7235, 7235, 7417, 7420, 7930
- \coffin_calculate_intersection:nnnnnnnn 7235, 7241, 7250, 7876
- \coffin_calculate_intersection_aux:nnnnnnN 7235, 7262, 7277, 7286, 7293, 7327, 7336
- \coffin_clear:c 6976
- \coffin_clear:N ... 133, 6976, 6976, 6984
- \coffin_display_attach:Nnnnn 7841, 7881, 7903, 7922, 7928
- \coffin_display_handles:cn ... 136, 7841
- \coffin_display_handles:Nn 7841, 7841, 7927

\coffin_display_handles_aux:nnnn ...	\coffin_resize_common:Nnn
..... 7841 , 7909 , 7914 , 7920 7660 , 7663 , 7663 , 7690
\coffin_display_handles_aux:nnnnnn .	\coffin_rotate:cn
..... 7841 , 7867 , 7871	\coffin_rotate:Nn . 134 , 7517 , 7517 , 7551
\coffin_dp:c	\coffin_rotate_bounding:nnn
\coffin_dp:N 7530 , 7564 , 7564
\coffin_end_user_dimensions:	\coffin_rotate_corner:Nnnn
..... 7137 , 7152 , 7169 , 7182 , 7676 7525 , 7564 , 7570
\coffin_find_bounding_shift:	\coffin_rotate_pole:Nnnnnn
..... 7532 , 7621 , 7621 7527 , 7576 , 7576
\coffin_find_bounding_shift_aux:nn .	\coffin_rotate_vector:nnNN
..... 7621 , 7625 , 7627	.. 7566 , 7572 , 7578 , 7579 , 7588 , 7588
\coffin_find_corner_maxima:N	\coffin_saved_Depth: 6956 , 6956 , 7140 , 7155
..... 7531 , 7605 , 7605	\coffin_saved_Height:
\coffin_find_corner_maxima_aux:nn 6956 , 6957 , 7139 , 7154
..... 7605 , 7612 , 7614	\coffin_saved_TotalHeight:
\coffin_get_pole:NnN 6956 , 6958 , 7141 , 7156
... 7106 , 7106 , 7237 , 7238 , 7470 ,	\coffin_saved_Width: 6956 , 6959 , 7142 , 7157
7471 , 7474 , 7475 , 7855 , 7856 , 7859	\coffin_scale:cn
\coffin_gset_eq_structure:NN 7123 , 7130	\coffin_scale:Nnn . 135 , 7678 , 7678 , 7693
\coffin_ht:c	\coffin_scale_corner:Nnnn 7666 , 7703 , 7703
\coffin_ht:N	\coffin_scale_pole:Nnnnnn 7668 , 7703 , 7709
\coffin_if_exist:NT	\coffin_scale_vector:nnNN
..... 6960 , 6960 , 6978 , 6998 , 7694 , 7694 , 7705 , 7711
7015 , 7042 , 7059 , 7089 , 7161 , 7174	\coffin_set_bounding:N . 7528 , 7552 , 7552
\coffin_join:cnncnnnn	\coffin_set_eq:cc
..... 7350	\coffin_set_eq:cN
\coffin_join:cnncnnnn 7087
..... 7350	\coffin_set_eq:Nc
\coffin_join:NnnNnnnn 135 , 7350 , 7350 , 7386 133 , 7087 ,
\coffin_mark_handle:cnnn	7087 , 7095 , 7384 , 7403 , 7432 , 7862
..... 7786	\coffin_set_eq_structure:NN
\coffin_mark_handle:Nnnn 7092 , 7123 , 7123
..... 136 , 7786 , 7786 , 7840	\coffin_set_horizontal_pole:cn ..
\coffin_mark_handle_aux:nnnnNnn 7159
..... 7786 , 7824 , 7829 , 7833	\coffin_set_horizontal_pole:Nnn
\coffin_new:c 134 , 7159 , 7159 , 7187
\coffin_new:N	\coffin_set_pole:Nnn ...
..... 133 , 6985 , 6985 , 7159 , 7185 , 7189
6995 , 7096 , 7098 , 7099 , 7733 – 7735	\coffin_set_pole:Nnx
\coffin_offset_corner:Nnnnn 7029 , 7072 , 7159 , 7164 , 7177 , 7446 ,
..... 7456 , 7458	7483 , 7487 , 7495 , 7499 , 7581 , 7712
\coffin_offset_corners:Nnn	\coffin_set_user_dimensions:N
.. 7372 , 7373 , 7379 , 7380 , 7453 , 7453	.. 7137 , 7137 , 7163 , 7176 , 7652 , 7681
\coffin_offset_corners:Nnnnn	\coffin_set_vertical_pole:cn ..
..... 7453 7159
\coffin_offset_pole:Nnnnnnn	\coffin_set_vertical_pole:Nnn
..... 7434 , 7437 , 7439 134 , 7159 , 7172 , 7188
\coffin_offset_poles:Nnn .	\coffin_shift_corner:Nnnn
..... 7370 , 7371 , 7547 , 7629 , 7629
7376 , 7377 , 7399 , 7400 , 7434 , 7434	\coffin_shift_pole:Nnnnnn
\coffin_reset_structure:N 7549 , 7629 , 7637
..... 6981 , 7007 , 7025 ,	\coffin_show_aux:n
7049 , 7068 , 7116 , 7116 , 7364 , 7394 7952
\coffin_resize:cn	\coffin_show_aux:nn
..... 7650 7969 , 7979
\coffin_resize:Nnn .	\coffin_show_aux:w
..... 134 , 7650 , 7650 , 7662 7952 , 7972 , 7984
	\coffin_show_structure:c
 7952

- \coffin_show_structure:N 4616, 5003, 11618, 11706, 11916, 12101, 12111, 12251, 12440, 12611
- 136, 7952, 7952, 7985
- \coffin_typeset:cnnnn 7503
- \coffin_typeset:Nnnnn 135, 7503, 7503, 7510
- \coffin_update_B:nnnnnnnnN 7468, 7476, 7491
- 7468, 7476, 7491
- \coffin_update_corners:N 7009, 7027, 7051, 7070, 7190, 7190
- .. 7009, 7027, 7051, 7070, 7190, 7190
- \coffin_update_poles:N .. 7008, 7026, 7050, 7069, 7201, 7201, 7367, 7398
- 7050, 7069, 7201, 7201, 7367, 7398
- \coffin_update_T:nnnnnnnnN 7468, 7472, 7479
- 7468, 7472, 7479
- \coffin_update_vertical_poles:NNN .. 7383, 7402, 7468, 7468
- 7383, 7402, 7468, 7468
- \coffin_wd:c 7100, 7105
- \coffin_wd:N 136, 7100, 7104
- 136, 7100, 7104
- \coffin_x_shift_corner:Nnnn 7672, 7718, 7718
- 7672, 7718, 7718
- \coffin_x_shift_pole:Nnnnnn 7674, 7718, 7725
- 7674, 7718, 7725
- \color 7794, 7806, 7849, 7890
- 7794, 7806, 7849, 7890
- \color_ensure_current 137
- 137
- \color_ensure_current: 7003, 7021, 7044, 7063, 8017, 8018, 8022
- 7003, 7021, 7044, 7063, 8017, 8018, 8022
- \color_group_begin 137
- 137
- \color_group_begin: 7002, 7020, 7044, 7063, 8011, 8011
- 7002, 7020, 7044, 7063, 8011, 8011
- \color_group_end 137
- 137
- \color_group_end: 7005, 7023, 7047, 7066, 8011, 8012
- 7005, 7023, 7047, 7066, 8011, 8012
- \copy 605
- 605
- \count 656
- 656
- \countdef 355
- 355
- \cr 380
- 380
- \crrcr 381
- 381
- \cs:w 16, 804, 806, 819, 851, 1115, 1143, 1357, 1389, 1543, 1582, 1596, 1598, 1600, 1604–1606, 1641, 1647, 1667, 1669, 1674, 1681, 1682, 1740, 1780, 2183, 2185, 3356, 4616, 5003, 12251, 12611
- 16, 804, 806, 819, 851, 1115, 1143, 1357, 1389, 1543, 1582, 1596, 1598, 1600, 1604–1606, 1641, 1647, 1667, 1669, 1674, 1681, 1682, 1740, 1780, 2183, 2185, 3356, 4616, 5003, 12251, 12611
- \cs_end 16
- 16
- \cs_end: 804, 807, 819, 851, 1109, 1115, 1137, 1143, 1297, 1357, 1389, 1543, 1582, 1596, 1598, 1600, 1604–1606, 1641, 1647, 1667, 1669, 1674, 1681, 1682, 1740, 1780, 2180, 2186–2189, 2191, 2193, 2195, 2197, 2199, 2201, 2203, 3356, 4615, 4616, 5003, 11618, 11706, 11916, 12101, 12111, 12251, 12440, 12611
- 804, 807, 819, 851, 1109, 1115, 1137, 1143, 1297, 1357, 1389, 1543, 1582, 1596, 1598, 1600, 1604–1606, 1641, 1647, 1667, 1669, 1674, 1681, 1682, 1740, 1780, 2180, 2186–2189, 2191, 2193, 2195, 2197, 2199, 2201, 2203, 3356, 4615, 4616, 5003, 11618, 11706, 11916, 12101, 12111, 12251, 12440, 12611
- \cs_generate_from_arg_count:cNnn ... 1013, 1021, 1029, 1037, 1345, 1388
- 1013, 1021, 1029, 1037, 1345, 1388
- \cs_generate_from_arg_count:NNnn ... 14, 1322, 1322, 1346, 1356
- 14, 1322, 1322, 1346, 1356
- \cs_generate_from_arg_count_aux:nwn 1322, 1325–1334, 1343
- 1322, 1325–1334, 1343
- \cs_generate_from_arg_count_error_msg:Nn 1322, 1336, 1347
- 1322, 1336, 1347
- \cs_generate_internal_variant:n 31, 1822, 1861, 1861
- 31, 1822, 1861, 1861
- \cs_generate_internal_variant_aux:N 1861, 1866, 1869, 1875
- 1861, 1866, 1869, 1875
- \cs_generate_variant:Nn 25, 1781, 1781, 1877–1884, 1895, 1904–1907, 1920, 1921, 1930–1933, 2079, 2080, 2085, 2086, 2151, 2174, 2440, 2441, 2458–2461, 2480–2483, 3294, 3315, 3318, 3319, 3321, 3322, 3324, 3325, 3334–3337, 3346–3349, 3353, 3354, 3728, 3931, 3934, 3935, 3939, 3940, 3942, 3943, 3945, 3946, 3955–3958, 3962, 3963, 3967, 3968, 4093, 4095, 4117, 4120, 4121, 4125, 4126, 4128, 4129, 4131, 4132, 4136, 4137, 4141, 4142, 4166, 4173, 4174, 4176, 4191, 4195, 4196, 4200, 4201, 4203, 4204, 4206, 4207, 4211, 4212, 4216, 4217, 4221, 4223, 4249, 4260, 4261, 4266, 4267, 4272, 4273, 4294–4299, 4316–4323, 4340–4347, 4381–4384, 4400, 4401, 4424–4427, 4468, 4469, 4474, 4475, 4478–4485, 4494–4497, 4506–4509, 4529–4532, 4551–4553, 4560–4562, 4575, 4596, 4607, 4611, 4632, 4633, 4685, 4686, 4694, 4695, 4723–4726, 4813, 4935, 4938, 4996, 4997, 5036, 5075, 5076, 5081–5084, 5089–5092, 5108, 5109, 5134, 5135, 5157–5162, 5171, 5187, 5188, 5212, 5239, 5240, 5265, 5294, 5305, 5306, 5359, 5382–5387, 5416–5427, 5437, 5461, 5463, 5488, 5489, 5511–5516, 5616, 5617, 5664, 5665, 5680–5683, 5688–5691, 5696, 5713, 5714, 5748, 5749, 5779, 5780, 5804–5812, 5827, 5849, 5872, 5913, 5937, 5967, 6019–6022, 6037, 6038, 6056, 6092–6095, 6104, 6105, 6123–6126, 6141,

6143, 6145, 6147, 6162, 6163, 6172–6175, 6197–6204, 6216–6221, 6235, 6236, 6247, 6278, 6297–6302, 6316, 6330, 6340, 6341, 6347–6349, 6352, 6374, 6379, 6380, 6393, 6394, 6399, 6400, 6405, 6406, 6410–6412, 6419–6421, 6424, 6425, 6441–6448, 6451–6454, 6459, 6460, 6475, 6479, 6480, 6485, 6486, 6491, 6492, 6510, 6511, 6519, 6520, 6525, 6526, 6531, 6532, 6537, 6538, 6549, 6550, 6723, 6764, 6784, 6806, 6849, 6879, 6902, 6984, 6995, 7012, 7039, 7056, 7086, 7095, 7187–7189, 7386, 7414, 7510, 7551, 7662, 7693, 7840, 7927, 7985, 8078, 8079, 8086, 8087, 8114, 8115, 8232, 8233, 8285, 8288, 8720–8723, 8850, 9236, 9401, 9454, 9455, 9558, 9559, 9572, 9573, 10161, 10167, 10172, 10173, 10206, 10207, 10254, 10255, 10270, 10332, 10335, 10402, 10627, 10630, 10662, 10665, 10746, 10747, 10771, 10772, 10797, 10798, 10900, 10901, 10918, 10919, 11031, 11032, 11583, 11584, 11680, 11681, 11881, 11882, 12074, 12075, 12377, 12392, 12393, 12726, 12727, 13256, 13259	\cs_gnew:cpx	1504
\cs_generate_variant_aux:N	\cs_gnew:Npn	1492
1784, 1833, 1846	\cs_gnew:Npx	1496
\cs_generate_variant_aux:NNn	\cs_gnew_eq:cc	1512
1781, 1795, 1817	\cs_gnew_eq:cN	1510
\cs_generate_variant_aux:nnNNn	\cs_gnew_eq:Nc	1511
1781, 1785, 1788	\cs_gnew_eq:NN	1509
\cs_generate_variant_aux:Nnnw	\cs_gnew_nopar:cpn	1499
1781, 1789, 1790, 1815	\cs_gnew_nopar:cpx	1503
\cs_generate_variant_aux:w	\cs_gnew_nopar:Npn	1491
1833, 1848, 1855	\cs_gnew_nopar:Npx	1495
\cs_get_arg_count_from_signature:c	\cs_gnew_protected:cpn	1502
1320, 1390	\cs_gnew_protected:cpx	1506
\cs_get_arg_count_from_signature:N	\cs_gnew_protected:Npn	1494
19, 932, 941, 948, 958, 1304, 1304, 1321, 1358	\cs_gnew_protected:Npx	1498
\cs_get_arg_count_from_signature_aux:nnN	\cs_gnew_protected_nopar:cpn	1501
1304, 1305, 1306	\cs_gnew_protected_nopar:cpx	1505
\cs_get_arg_count_from_signature_auxii:w	\cs_gnew_protected_nopar:Npn	1493
1304, 1314, 1319	\cs_gnew_protected_nopar:Npx	1497
\cs_get_function_name:N . 19, 1091, 1091	\cs_gset:cn	1393
\cs_get_function_signature:N	\cs_gset:cpn	1259, 1261, 4579, 4589, 5832, 5842, 6241, 8549, 8551
19, 1091, 1093	\cs_gset:cpx	1259, 1262
\cs_gnew:cpn	1500	1393
	\cs_gset:cx	1393
	\cs_gset:Nn	13, 1352
	\cs_gset:Npn	
	. 11, 837, 839, 1245, 1261, 3091, 5269	
	\cs_gset:Npx 837, 841, 1246, 1262, 3092, 5274	
	\cs_gset:Nx	1352
	\cs_gset_eq:cc . . . 1289, 1292, 1915, 4281	
	\cs_gset_eq:cN	
	. . . 1289, 1291, 1302, 1914, 4279, 5278, 6053, 8143, 8180, 9201, 9203	
	\cs_gset_eq:Nc 1289, 1290, 1913, 4280, 5285, 8135, 8151, 8172, 8188	
	\cs_gset_eq:NN	
	. 15, 1289, 1289–1292, 1294, 1901, 1903, 1912, 3094, 4247, 4278, 6052	
	\cs_gset_nopar:cn	1393
	\cs_gset_nopar:cpn	1251, 1255
	\cs_gset_nopar:cpx 1251, 1256, 2989	
	\cs_gset_nopar:cx	1393
	\cs_gset_nopar:Nn	14, 1352
	\cs_gset_nopar:Npn	12, 837, 837, 840, 844, 848, 1243, 1255, 2248
	\cs_gset_nopar:Npx 837, 838, 842, 846, 850, 1244, 1256, 2254, 4253, 4258, 4289, 4291, 4293, 4309, 4311, 4313, 4315, 4333, 4335, 4337, 4339	
	\cs_gset_nopar:Nx	1352

- \cs_gset_protected:cn [1393](#)
- \cs_gset_protected:cpn [1271](#), [1273](#)
- \cs_gset_protected:cpx [1271](#), [1274](#)
- \cs_gset_protected:cx [1393](#)
- \cs_gset_protected:Nn [14](#), [1352](#)
- \cs_gset_protected:Npn
..... [12](#), [837](#), [847](#), [1249](#), [1273](#)
- \cs_gset_protected:Npx [837](#), [849](#), [1250](#), [1274](#)
- \cs_gset_protected:Nx [1352](#)
- \cs_gset_protected_nopar:cn [1393](#)
- \cs_gset_protected_nopar:cpn [1265](#), [1267](#)
- \cs_gset_protected_nopar:cpx [1265](#), [1268](#)
- \cs_gset_protected_nopar:cx [1393](#)
- \cs_gset_protected_nopar:Nn ... [14](#), [1352](#)
- \cs_gset_protected_nopar:Npn
..... [12](#), [837](#), [843](#), [1247](#), [1267](#)
- \cs_gset_protected_nopar:Npx
..... [837](#), [845](#), [1248](#), [1268](#)
- \cs_gset_protected_nopar:Nx [1352](#)
- \cs_gundefine:c [1516](#)
- \cs_gundefine:N [1515](#)
- \cs_if_eq:cc [1417](#)
- \cs_if_eq:ccF [1433](#)
- \cs_if_eq:ccT [1432](#)
- \cs_if_eq:ccTF [1431](#)
- \cs_if_eq:cN [1417](#)
- \cs_if_eq:cnF [1425](#)
- \cs_if_eq:cNT [1424](#)
- \cs_if_eq:cNTF [1423](#)
- \cs_if_eq:Nc [1417](#)
- \cs_if_eq:NcF [1429](#)
- \cs_if_eq:NcT [1428](#)
- \cs_if_eq:NcTF [1427](#)
- \cs_if_eq:NN [1417](#), [1417](#)
- \cs_if_eq:NNF [1425](#), [1429](#), [1433](#)
- \cs_if_eq:NNT [1424](#), [1428](#), [1432](#)
- \cs_if_eq:NNTF . [21](#), [1423](#), [1427](#), [1431](#), [8715](#)
- \cs_if_eq_p:cc [1430](#)
- \cs_if_eq_p:cN [1422](#)
- \cs_if_eq_p:Nc [1426](#)
- \cs_if_eq_p:NN [1422](#), [1426](#), [1430](#)
- \cs_if_exist:c [1095](#), [1107](#)
- \cs_if_exist:cF .. [3816](#), [3823](#), [3825](#), [9374](#)
- \cs_if_exist:cT [8529](#), [9618](#)
- \cs_if_exist:cTF [1160](#),
[1162](#), [1164](#), [1166](#), [6964](#), [7954](#), [8134](#),
[8160](#), [8171](#), [8197](#), [8432](#), [8807](#), [8817](#),
[9249](#), [9348](#), [9655](#), [9669](#), [9675](#), [12974](#)
- \cs_if_exist:N [1095](#), [1095](#)
- \cs_if_exist:NF .. [1227](#), [9292](#), [9307](#), [9448](#)
- \cs_if_exist:NT
.. [1456](#), [1467](#), [8208](#), [8221](#), [9798](#), [9816](#)
- \cs_if_exist:NTF [21](#), [1152](#), [1154](#),
[1156](#), [1158](#), [1436](#), [2710](#), [4269](#), [4271](#),
[6055](#), [6058](#), [6383](#), [6389](#), [6962](#), [8485](#)
- \cs_if_exist_use:c [1151](#), [1165](#)
- \cs_if_exist_use:cF [1161](#)
- \cs_if_exist_use:cT [1163](#)
- \cs_if_exist_use:cTF [1159](#)
- \cs_if_exist_use:N [23](#), [1151](#), [1157](#)
- \cs_if_exist_use:NF [1153](#)
- \cs_if_exist_use:NT [1155](#)
- \cs_if_exist_use:NTF [1151](#)
- \cs_if_free:c [1123](#), [1135](#)
- \cs_if_free:cT [1863](#)
- \cs_if_free:N [1123](#), [1123](#)
- \cs_if_free:NF [1204](#), [1214](#)
- \cs_if_free:NTF [21](#), [1819](#), [8297](#), [8299](#), [13266](#)
- \cs_meaning:c [820](#), [820](#)
- \cs_meaning:N [15](#), [804](#), [808](#), [820](#)
- \cs_new:cn [1409](#)
- \cs_new:cpn ... [1259](#), [1263](#), [1500](#), [1971](#),
[1984](#), [2017](#), [2019](#), [2021](#), [2022](#), [2187](#)–
[2190](#), [2192](#), [2194](#), [2196](#), [2198](#), [2200](#),
[2202](#), [2204](#)–[2214](#), [3371](#), [3379](#), [3387](#),
[3395](#), [3403](#), [3411](#), [3419](#), [4010](#)–[4016](#)
- \cs_new:cpx [1259](#), [1264](#), [1504](#), [1865](#)
- \cs_new:cx [1409](#)
- \cs_new:Nn [12](#), [1377](#)
- \cs_new:Npn
.. [10](#), [907](#), [939](#), [1235](#), [1245](#), [1263](#),
[1304](#), [1306](#), [1319](#), [1347](#), [1492](#), [1537](#)–
[1542](#), [1544](#), [1546](#), [1558](#), [1564](#), [1583](#),
[1590](#), [1591](#), [1593](#), [1595](#), [1597](#), [1599](#),
[1601](#), [1608](#), [1610](#), [1615](#), [1620](#), [1626](#),
[1632](#), [1638](#), [1644](#), [1657](#), [1664](#), [1671](#),
[1678](#), [1712](#), [1713](#), [1718](#), [1720](#), [1725](#),
[1730](#), [1732](#), [1734](#), [1735](#), [1737](#), [1743](#),
[1749](#), [1754](#), [1761](#), [1763](#), [1765](#), [1767](#),
[1768](#), [1770](#), [1775](#), [1780](#), [1788](#), [1790](#),
[1817](#), [1855](#), [1869](#), [1916](#), [1918](#), [1944](#),
[1949](#), [1959](#), [1969](#), [1970](#), [1997](#)–[1999](#),
[2008](#), [2023](#), [2028](#), [2052](#), [2058](#), [2064](#),
[2068](#), [2069](#), [2075](#), [2077](#), [2081](#), [2083](#),
[2087](#), [2095](#), [2100](#), [2108](#), [2113](#), [2114](#),
[2119](#), [2121](#), [2127](#), [2132](#), [2134](#), [2140](#),
[2145](#), [2152](#), [2157](#), [2163](#), [2168](#), [2215](#),
[2226](#), [2235](#), [2410](#), [2416](#), [2424](#), [2431](#),
[2838](#), [2864](#), [2879](#), [2960](#), [3050](#), [3059](#),
[3068](#), [3081](#), [3223](#), [3225](#), [3233](#), [3244](#),

- 3255, 3280, 3281, 3356, 3359, 3369,
 3451, 3459, 3467, 3473, 3479, 3487,
 3495, 3501, 3507, 3508, 3522, 3528,
 3560, 3592, 3594, 3600, 3612, 3620,
 3653, 3655, 3657, 3659, 3669, 3695,
 3700, 3705, 3729, 3737, 3739, 3748,
 3750, 3759, 3761, 3771, 3780, 3782,
 3784, 3884, 3899, 3969, 3979, 3981,
 3998, 4073, 4075, 4077, 4084, 4163,
 4168, 4171, 4218, 4224, 4451, 4452,
 4462, 4510, 4563, 4570, 4621, 4631,
 4634, 4636, 4644, 4658, 4666, 4671,
 4677, 4687–4689, 4691, 4693, 4696,
 4704, 4750, 4758, 4806, 4833, 4843–
 4846, 4855, 4856, 4862, 4873, 4882,
 4883, 4890, 4896, 4898, 4900, 4905,
 4914, 4916, 4918, 4924, 4933, 4939,
 4951–4953, 4965, 4967, 4969, 4976,
 4977, 4985, 5035, 5037, 5046, 5207,
 5241, 5242, 5253, 5259, 5353, 5358,
 5428, 5436, 5481, 5510, 5530, 5560,
 5566, 5620, 5635, 5637, 5638, 5643,
 5648, 5776, 5778, 5821, 5873, 5874,
 5879, 5907, 5912, 5918, 5926, 5927,
 5936, 5938, 5946, 5961, 5968, 5976,
 5978, 5992, 5997, 6018, 6078, 6183,
 6189, 6227, 6270, 6277, 6303, 6308,
 6317, 6324, 6533, 7979, 8249, 8467,
 8474, 8719, 9078, 9164, 13243, 13254
 \cs_new:Npx
 .. [1235](#), 1246, 1264, 1496, 8340, 9065
 \cs_new:Nx [1377](#)
 \cs_new_eq:cc 963, [1281](#), 1288, 1512
 \cs_new_eq:cN [1281](#),
 1286, 1510, 6049, 11935, 11959, 11990
 \cs_new_eq:Nc [1281](#), 1287, 1511
 \cs_new_eq:NN [14](#),
[1281](#), 1281, 1286–1288, 1444–1455,
 1491–1506, 1509–1512, 1515, 1516,
 1519, 1521–1523, 1552, 1894, 1908–
 1915, 2587–2589, 2872–2874, 3114–
 3118, 3138, 3139, 3142–3145, 3148,
 3151–3158, 3160, 3162–3176, 3178,
 3180–3186, 3189–3204, 3213–3218,
 3355, 3846, 3847, 3874–3876, 3921–
 3923, 4092, 4094, 4102, 4103, 4165,
 4167, 4170, 4175, 4177, 4178, 4220,
 4222, 4274–4281, 4413, 4414, 4608,
 4609, 4612, 4814, 5006–5013, 5016–
 5023, 5026–5030, 5033, 5034, 5053–
 5070, 5243–5245, 5307–5332, 5569,
 5570, 5573, 5574, 5585–5602, 5715–
 5732, 5886, 5887, 6024, 6025, 6028,
 6029, 6032, 6033, 6048, 6059–6067,
 6248, 6249, 6332, 6333, 6344, 6407–
 6409, 6422, 6423, 6434–6436, 6462,
 6468, 6474, 6493–6500, 6508, 6509,
 6539–6548, 6903, 7100–7105, 8011,
 8031–8035, 8055, 8065, 8085, 8269,
 8284, 8301, 8514, 8515, 9082, 9085–
 9089, 9643, 9745–9747, 10258–
 10267, 13239, 13240, 13342–13345
 \cs_new_nopar:cn [1409](#)
 \cs_new_nopar:cpn
 [1251](#), 1257, 1499, 2035–2047,
 2049, 10309, 10310, 10312, 10314,
 10316, 10318, 10320, 10322, 10324,
 10370, 10372, 10374, 10376, 10378,
 10380, 10382, 10384, 10386, 10432,
 10437, 10442, 10447, 10452, 10457,
 10462, 10467, 10472, 10477, 10482
 \cs_new_nopar:cpx [1251](#), 1258, 1503
 \cs_new_nopar:cx [1409](#)
 \cs_new_nopar:Nn [12](#), [1377](#)
 \cs_new_nopar:Npn [10](#), [1235](#),
 1243, 1257, 1320, 1345, 1422–1434,
 1443, 1478, 1491, 1536, 1570, 1581,
 1650, 1686–1693, 1700–1704, 1751–
 1753, 1846, 2033, 2034, 2175, 2182,
 2184, 2186, 2277, 2279, 2288, 2293,
 2302, 2307, 2495, 2497, 2565, 2567,
 2571, 2573, 2577, 2579, 2583, 2585,
 2694, 2734, 2741, 2752, 2763, 2774,
 2785, 2793, 2801, 2809, 2830, 2837,
 2846, 2855, 2875–2878, 2929, 2938,
 2946, 3047, 3121, 3122, 3124, 3125,
 3127, 3128, 3130, 3131, 3133, 3134,
 3664, 3674, 3679–3694, 3800, 3809,
 3843, 3845, 3877, 4457, 4565, 4610,
 4613, 4626, 4702, 4710, 5156, 5360,
 5438, 5454, 5462, 5464, 5473, 5813,
 6222, 6956–6959, 7984, 8300, 8465,
 8588–8590, 8690–8695, 9290, 9305,
 9412, 9633, 9635, 9644, 9653, 9662,
 9679, 10268, 10271, 10288, 10289,
 10295, 10304, 10325, 10331, 10333,
 10336, 10348, 10359, 10368, 10387,
 10395, 10400, 10403, 10415, 10424,
 10426, 10483, 10496, 10515, 10535,
 10541, 10580, 10619–10621, 10623

\cs_new_nopar:Npx	8527, 8536, 8538, 8545, 8547, 8554,
..... 1235 , 1244 , 1258 , 1495 , 1852	8604, 8619, 8627, 8635, 8637, 8660,
\cs_new_nopar:Nx	1377
\cs_new_protected:cn	1409
\cs_new_protected:cpn ... 1271 , 1275 ,	8877, 8879, 8881, 8883, 8885, 8912,
1502, 9466, 9468, 9470, 9472, 9474,	8921, 8934, 8936, 8938, 8940, 8943,
9484, 9486, 9504, 9514, 9516, 9520	8956, 8958, 8960, 8962, 9092–9098,
\cs_new_protected:cpx .. 1271 , 1276 , 1506	9120, 9134, 9146, 9166, 9171, 9192,
\cs_new_protected:cx	1409
\cs_new_protected:Nn	12 , 1377
\cs_new_protected:Npn	10 ,
922, 956, 1235 , 1249 , 1275 , 1277 ,	\cs_new_protected:Npx
1281 , 1322 , 1494 , 1553 , 1781 , 1861 , 1235 , 1250 , 1276 , 1498
2245 , 2251 , 2262 , 2393 , 2394 , 2884 ,	\cs_new_protected:Nx
2890 , 2907 , 2909 , 2911 , 2925 , 2927 ,	1377
3295 , 3326 , 3328 , 3350 , 3925 , 3932 ,	\cs_new_protected_nopar:cn
3933 , 3936 , 3938 , 3941 , 3944 , 3947 ,	1409
3949 , 3951 , 3953 , 3959 , 3961 , 3964 ,	\cs_new_protected_nopar:cpn
3966 , 4111 , 4118 , 4119 , 4122 , 4124 , 1265 , 1269 , 1501 , 9456 ,
4127 , 4130 , 4133 , 4135 , 4138 , 4140 ,	9458 , 9460 , 9462 , 9464 , 9476 , 9478 ,
4185 , 4192 , 4194 , 4197 , 4199 , 4202 ,	9480 , 9482 , 9488 , 9490 , 9492 , 9494 ,
4205 , 4208 , 4210 , 4213 , 4215 , 4250 ,	9496 , 9498 , 9500 , 9502 , 9506 , 9508 ,
4255 , 4282 , 4284 , 4286 , 4288 , 4290 ,	9510 , 9512 , 9518 , 9522 , 9524 , 9526 ,
4292 , 4300 , 4302 , 4304 , 4306 , 4308 ,	9528 , 9530 , 9532 , 9534 , 9536 , 9538 ,
4310 , 4312 , 4314 , 4324 , 4326 , 4328 ,	9540 , 9542 , 9544 , 9546 , 9548 , 12981 ,
4330 , 4332 , 4334 , 4336 , 4338 , 4363 ,	13007 , 13042 , 13060 , 13092 , 13124
4377 , 4402 , 4428 , 4576 , 4586 , 4597 ,	\cs_new_protected_nopar:cpx
4601 , 4681 , 4683 , 4812 , 4991 , 5077 , 1265 , 1270 , 1505
5079 , 5085 , 5087 , 5094 , 5096 , 5098 ,	\cs_new_protected_nopar:cx
5110 , 5112 , 5114 , 5169 , 5182 , 5266 ,	1409
5271 , 5276 , 5288 , 5295 , 5490 , 5495 ,	\cs_new_protected_nopar:Nn
5500 , 5505 , 5584 , 5660 , 5662 , 5670 ,	13 , 1377
5684 , 5694 , 5705 , 5712 , 5734 , 5736 ,	\cs_new_protected_nopar:Npn
5738 , 5750 , 5752 , 5754 , 5797 , 5829 , 11 , 1235 , 1247 , 1252 ,
5839 , 5850 , 5860 , 5867 , 5998 , 6000 ,	1269 , 1278–1280 , 1286–1293 , 1295 ,
6002 , 6035 , 6036 , 6048–6054 , 6057 ,	1342 , 1493 , 1685 , 1694–1699 , 1705–
6070 , 6079 , 6086 , 6088 , 6090 , 6096 ,	1711 , 1894 , 1896 , 1898 , 1900 , 1902 ,
6102 , 6106 , 6112 , 6118 , 6127–6129 ,	2281 , 2314 , 2403 , 2493 , 2499 , 2501 ,
6133 , 6153 , 6211 , 6238 , 6291 , 6336 ,	2503 , 2505 , 2507 , 2509 , 2511 , 2513 ,
6338 , 6368 , 6375 , 6377 , 6381 , 6387 ,	2515 , 2517 , 2519 , 2521 , 2523 , 2525 ,
6395 , 6397 , 6401 , 6403 , 6413 , 6415 ,	2527 , 2529 , 2531 , 2533 , 2535 , 2537 ,
6417 , 6426 , 6428 , 6430 , 6432 , 6455 ,	2539 , 2541 , 2543 , 2545 , 2547 , 2549 ,
6457 , 6476–6478 , 6481 , 6483 , 6487 ,	2551 , 2553 , 2555 , 2557 , 2559 , 2561 ,
6489 , 6501 , 6503 , 6504 , 6506 , 6512–	2563 , 2569 , 2575 , 2581 , 2587 , 2880 ,
6514 , 6516–6518 , 6521 , 6523 , 6527 ,	2882 , 2969 , 2978 , 3096 , 3107 , 3109 ,
6529 , 6535 , 6551 , 6570 , 6587 , 6621 ,	3111 , 3288 , 3316 , 3317 , 3320 , 3323 ,
6629 , 6640 , 6651 , 6662 , 6673 , 6684 ,	3330 , 3332 , 3338 , 3340 , 3342 , 3344 ,
6697 , 6724 , 6744 , 6765 , 6785 , 6807 ,	3352 , 4244 , 4262 , 4264 , 4268 , 4270 ,
6823 , 6847 , 6850 , 6880 , 6960 , 6996 ,	4359 , 4361 , 4385 , 4387 , 4389 , 4416 ,
7013 , 8339 , 8346 , 8385 , 8508 , 8510 ,	4418 , 4420 , 4422 , 4464 , 4466 , 4470 ,
	4472 , 4548–4550 , 4599 , 4936 , 5000 ,
	5002 , 5071 , 5073 , 5163 , 5172 , 5174 ,
	5176 , 5189 , 5195 , 5213 , 5215 , 5217 ,
	5223 , 5246 , 5282 , 5334 , 5517 , 5519 ,

5521, 5523, 5536, 5538, 5540, 5603,
 5605, 5607, 5666, 5668, 5686, 5692,
 5697, 5699, 5701, 5888, 6149, 6151,
 6251, 6976, 6985, 7040, 7055, 7057,
 7085, 7087, 7106, 7116, 7123, 7130,
 7137, 7152, 7159, 7172, 7185, 7190,
 7201, 7235, 7250, 7336, 7350, 7387,
 7405, 7415, 7434, 7439, 7453, 7458,
 7468, 7479, 7491, 7503, 7517, 7552,
 7564, 7570, 7576, 7588, 7605, 7614,
 7621, 7627, 7629, 7637, 7650, 7663,
 7678, 7694, 7703, 7709, 7718, 7725,
 7786, 7833, 7841, 7871, 7920, 7928,
 7952, 8012, 8018, 8022, 8068, 8071,
 8080, 8088, 8101, 8116, 8124, 8132,
 8155, 8169, 8192, 8206, 8219, 8234,
 8254, 8286, 8289, 8290, 8293, 8295,
 8296, 8298, 8392, 8402, 8418, 8430,
 8439, 8453, 8459, 8499, 8501, 8503,
 8505, 8699, 8821, 8829, 8848, 8851,
 8853, 8855, 8857, 8859, 8861, 8863,
 9237, 9256, 9263, 9320, 9360, 9393,
 9402, 9407, 9414, 9440, 9446, 9452,
 9574, 9784, 9794, 9828, 9845, 9850,
 9852, 9943, 9945, 9956, 9966, 9991,
 9999, 10001, 10003, 10009, 10011,
 10028, 10040, 10095–10097, 10105,
 10115, 10130, 10140, 10155, 10156,
 10162, 10168, 10170, 10174–10176,
 10208, 10210, 10212, 10625, 10628,
 10631, 10660, 10663, 10666, 10696,
 10705, 10718, 10744, 10745, 10748,
 10769, 10770, 10773, 10795, 10796,
 10799, 10812, 10849, 10866, 10898,
 10899, 10902, 10916, 10917, 10920,
 10971, 11001, 11013, 11018, 11024,
 11029, 11030, 11033, 11068, 11114,
 11132, 11149, 11160, 11161, 11166,
 11175, 11181, 11197, 11220, 11264,
 11324, 11341, 11346, 11356, 11366,
 11374, 11384, 11407, 11417, 11431,
 11437, 11450, 11469, 11487, 11526,
 11544, 11554, 11581, 11582, 11585,
 11632, 11665, 11678, 11679, 11682,
 11718, 11751, 11774, 11810, 11825,
 11879, 11880, 11883, 11930, 11940,
 11969, 11999, 12072, 12073, 12076,
 12120, 12148, 12160, 12195, 12227,
 12247, 12253, 12260, 12324, 12372,
 12390, 12391, 12394, 12427, 12451,
 12485, 12504, 12514, 12528, 12540,
 12562, 12587, 12595, 12607, 12613,
 12667, 12677, 12692, 12724, 12725,
 12728, 12779, 12815, 12869, 12881,
 12972, 13158, 13164, 13170, 13176,
 13182, 13188, 13194, 13206, 13214,
 13219, 13228, 13264, 13294, 13304
 \cs_new_protected_nopar:Npx
 . . 1235, 1248, 1270, 1497, 1850, 8447
 \cs_new_protected_nopar:Nx 1377
 \cs_set:cn 1393
 \cs_set:cpn . . 1259, 1259, 8540, 8542, 9386
 \cs_set:cpx 1259, 1260,
2315, 2319, 2323, 2327, 2331, 2340,
2349, 2358, 2367, 2369, 2371, 2373,
2375, 2377, 2379, 2381, 2383, 9391
 \cs_set:cx 1393
 \cs_set:Nn 13, 1352
 \cs_set:Npn 11, 823, 825,
851, 860–889, 899, 930, 964, 965,
1052–1055, 1073, 1078, 1088, 1091,
1093, 1151, 1153, 1155, 1157, 1159,
1161, 1163, 1165, 1235, 1251, 1259,
1352, 1385, 1524–1527, 2390, 2391,
3079, 3089, 3220, 4017, 4025, 4033,
4039, 4045, 4053, 4061, 4067, 4556,
4642, 5618, 5756, 5799, 8323, 8373
 \cs_set:Npx 823, 827, 855, 1260, 3090, 4436
 \cs_set:Nx 1352
 \cs_set_eq:cc . 961, 1277, 1280, 1911, 4277
 \cs_set_eq:cN 1277, 1278, 1910, 4275, 6051
 \cs_set_eq:Nc 1277, 1279, 1909, 4276
 \cs_set_eq:NN 15, 1277, 1277–1280, 1284,
1289, 1458–1465, 1469–1476, 1858,
1897, 1899, 1908, 2628, 2888, 2892,
2913, 2915, 2974, 2992, 3093, 4274,
5525, 5526, 5528, 6050, 7139–7146,
7154–7157, 8075, 8076, 8363–8365,
9566, 9568, 11071, 11164, 12003, 12925
 \cs_set_eq:NwN 1528, 1528
 \cs_set_nopar:cn 1393
 \cs_set_nopar:cpn 1251, 1253
 \cs_set_nopar:cpx 1251, 1254
 \cs_set_nopar:cx 1393
 \cs_set_nopar:Nn 13, 1352
 \cs_set_nopar:Npn 11, 823, 823, 825–827,
829–831, 834, 890, 892, 1058, 1065,
1200, 1253, 2981, 2987, 8591, 10250
 \cs_set_nopar:Npx
 . . . 823, 824, 828, 832, 836, 1254,

- 1555, 2894, 2899, 2916, 2917, 4283,
 4285, 4287, 4301, 4303, 4305, 4307,
 4325, 4327, 4329, 4331, 8357–8361
 \cs_set_nopar:Nx 1352
 \cs_set_protected:cn 1393
 \cs_set_protected:cpn .. 1271, 1271, 8702
 \cs_set_protected:cpx .. 1271, 1272,
 1354, 1387, 8704, 8706, 8708, 8710
 \cs_set_protected:cx 1393
 \cs_set_protected:Nn 13, 1352
 \cs_set_protected:Npn
 11, 823, 833, 852, 894, 902,
 910, 914, 917, 925, 934, 944, 947,
 951, 960, 962, 966, 974, 982, 990,
 998, 1006, 1011, 1019, 1027, 1035,
 1040, 1042, 1271, 5145, 6068, 6072,
 6080, 7064, 8914, 8916, 8918, 9040
 \cs_set_protected:Npx 823, 835, 1272, 8801
 \cs_set_protected:Nx 1352
 \cs_set_protected_nopar:cn 1393
 \cs_set_protected_nopar:cpn . 1265, 1265
 \cs_set_protected_nopar:cpx . 1265, 1266
 \cs_set_protected_nopar:cx 1393
 \cs_set_protected_nopar:Nn 13, 1352
 \cs_set_protected_nopar:Npn
 11, 310, 823, 829, 833,
 835, 839, 841, 843, 845, 847, 849,
 1180, 1182, 1184, 1196, 1198, 1202,
 1212, 1223, 1225, 1233, 1237, 1265,
 7045, 8292, 8294, 10042, 10051,
 10059, 10068, 11004, 13283, 13287,
 13314, 13315, 13320, 13321, 13333
 \cs_set_protected_nopar:Npx
 296, 823, 831, 1266,
 8670, 8796, 10186, 10228, 10248,
 10640, 10676, 10753, 10829, 10940,
 11047, 11057, 11083, 11610, 11623,
 11710, 11908, 11921, 12105, 12410,
 12418, 12444, 12636, 12750, 12756,
 12767, 12800, 12807, 12853, 12888
 \cs_set_protected_nopar:Nx 1352
 \cs_show:c 820, 822, 9680
 \cs_show:N 15, 804, 809, 822, 4812
 \cs_split_function:NN 19, 898, 906, 913,
 921, 929, 938, 946, 955, 1048, 1049,
 1067, 1073, 1092, 1094, 1305, 1785
 \cs_split_function_aux:w 1067, 1075, 1078
 \cs_split_function_auxii:w
 1067, 1086, 1088
 \cs_tmp:w 852, 855, 858, 860,
 1235, 1243–1251, 1253–1276, 1352,
 1361–1385, 1393–1416, 1821, 1858
 \cs_to_str:N
 .. 4, 17, 1058, 1058, 1076, 2291, 8301
 \cs_to_str_aux:w 1058, 1061, 1065
 \cs_undefine:c 1293, 1295, 1516
 \cs_undefine:N 15, 1293, 1293, 1515
 \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
 \currentgrouptype 696
 \currentifbranch 692
 \currentiflevel 691
 \currentiftype 693
- ## D
- \d 1834, 2722
 \dagger 3889, 3895
 \day 651
 \ddagger 3890, 3896
 \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 2393, 2394, 9092–9098
 \Depth 7137, 7140, 7144, 7148, 7155
 \detokenize 32, 35, 80, 93, 96,
 167, 170, 176, 185, 190, 192, 198,
 201, 204, 214, 269, 271, 276, 278, 683
 \dim_abs:n 72, 3969, 3969
 \dim_add:cn 3959
 \dim_add:Nn 70, 3959, 3959, 3961, 3962
 \dim_compare:n 3988, 3988
 \dim_compare:nF 4027, 4042
 \dim_compare:nNn 3983, 3983
 \dim_compare:nNnF 4055, 4070
 \dim_compare:nNnT
 .. 3948, 3952, 4047, 4064, 7356, 7361
 \dim_compare:nNnTF 72,
 2136, 6726, 6729, 6858, 6868, 6888,

- 6894, 7253, 7256, 7259, 7268, 7271,
7274, 7283, 7290, 7368, 7481, 7493
- \dim_compare:nT 4019, 4036
- \dim_compare:nTF 73
- \dim_compare_<:nw 3988
- \dim_compare_=:nw 3988
- \dim_compare_>:nw 3988
- \dim_compare_aux:wNN ... 3988, 3990, 3998
- \dim_compare_p:nNn 3983
- \dim_do_until:nn ... 74, 4017, 4039, 4043
- \dim_do_until:nNnn .. 73, 4045, 4067, 4071
- \dim_do_while:nn 74, 4017, 4033
- \dim_do_while:nNnn
..... 73, 4037, 4045, 4061, 4065
- \dim_eval:n 74,
2130, 4073, 4073, 6852, 6855, 6859,
6863, 6869, 6873, 6882, 6885, 6893,
6898, 7032, 7076, 7166, 7179, 7197,
7199, 7205, 7216, 7230, 7464, 7465,
7633, 7634, 7641, 7642, 7722, 7729
- \dim_eval:w 81, 3921, 3922, 3937, 3960,
3965, 3972, 3973, 3976, 3982, 3985,
3990, 4010–4016, 4074, 4076, 4080,
6414, 6416, 6418, 6427, 6429, 6431,
6433, 6482, 6502, 6515, 6528, 6552
- \dim_eval_end 81
- \dim_eval_end: 3921,
3923, 3937, 3960, 3965, 3976, 3977,
3982, 3985, 3991, 4074, 4076, 4080,
6414, 6416, 6418, 6427, 6429, 6431,
6433, 6482, 6502, 6515, 6528, 6552
- \dim_gadd:cn 3959
- \dim_gadd:Nn 70, 3959, 3961, 3963
- \dim_gset:cn 3936
- \dim_gset:Nn 71, 3936, 3938, 3940
- \dim_gset_eq:cc 3941
- \dim_gset_eq:cN 3941
- \dim_gset_eq:Nc 3941
- \dim_gset_eq:NN 71, 3941, 3944–3946
- \dim_gset_max:cn 3947
- \dim_gset_max:Nn ... 71, 3947, 3949, 3956
- \dim_gset_min:cn 3947
- \dim_gset_min:Nn ... 71, 3947, 3953, 3958
- \dim_gsub:cn 3959
- \dim_gsub:Nn 71, 3959, 3966, 3968
- \dim_gzero:c 3932
- \dim_gzero:N 70, 3932, 3933, 3935
- \dim_new:c 3924
- \dim_new:N 70,
3924, 3925, 3931, 4097, 4098, 4105–
4109, 6556–6563, 6912, 6938, 6939,
6944–6947, 6952–6955, 7512–7516,
7648, 7649, 7773, 7775, 7776, 10256
- \dim_ratio:nn 72, 3979, 3979
- \dim_ratio_aux:n 3979, 3980, 3981
- \dim_set:cn 3936
- \dim_set:Nn 70, 3936, 3936,
3938, 3939, 3948, 3952, 4099, 6589–
6591, 6638, 6649, 6702–6704, 6727,
6728, 6731, 6733, 6737, 6739, 6749–
6751, 6770–6772, 6792–6794, 6811,
6812, 6815, 6816, 7019, 7062, 7147–
7150, 7255, 7260, 7270, 7275, 7285,
7292, 7325, 7348, 7359, 7418, 7419,
7421, 7423, 7441, 7442, 7558, 7602,
7603, 7607–7610, 7623, 7698, 7701,
7774, 7879, 7880, 7931–7933, 7935
- \dim_set_eq:cc 3941
- \dim_set_eq:cN 3941
- \dim_set_eq:Nc 3941
- \dim_set_eq:NN 71, 3941, 3941–3943
- \dim_set_max:cn 3947
- \dim_set_max:Nn
71, 3947, 3947, 3950, 3955, 7617, 7619
- \dim_set_min:cn 3947
- \dim_set_min:Nn 71, 3947,
3951, 3954, 3957, 7616, 7618, 7628
- \dim_show:c 4094
- \dim_show:N 75, 4094, 4094, 4095
- \dim_strip_bp:n 81, 4075, 4075
- \dim_strip_pt:n 81, 4076, 4077, 4077
- \dim_strip_pt:w 4077, 4080, 4084
- \dim_sub:cn 3959
- \dim_sub:Nn 71, 3959, 3964, 3966, 3967
- \dim_until_do:nn ... 74, 4017, 4025, 4030
- \dim_until_do:nNnn .. 73, 4045, 4053, 4058
- \dim_use:c 4092
- \dim_use:N 74, 3971, 3990,
4074, 4080, 4092, 4092, 4093, 7193,
7195, 7199, 7210, 7223, 7449, 7555,
7557, 7560, 7562, 7568, 7574, 7583–
7585, 7707, 7714, 7960–7962, 10220
- \dim_while_do:nn ... 74, 4017, 4017, 4022
- \dim_while_do:nNnn .. 73, 4045, 4045, 4050
- \dim_zero:c 3932
- \dim_zero:N . 70, 3932, 3932–3934, 6592,
6705, 6752, 6773, 6795, 7246, 7247
- \dimen 657

<code>\dimendef</code>	356	10561, 10564, 10567, 10570, 10573,
<code>\dimexpr</code>	710	10576, 10591, 10594, 10597, 10600,
<code>\directlua</code>	15, 759	10603, 10606, 10609, 10612, 10615,
<code>\discretionary</code>	520	10647, 10683, 10712, 10726, 10731,
<code>\displayindent</code>	485	10782, 10820, 10836, 10861, 10884,
<code>\displaylimits</code>	495	10894, 10954, 10957, 11052, 11062,
<code>\displaystyle</code>	473	11097, 11100, 11136, 11138, 11141,
<code>\displaywidowpenalties</code>	723	11187, 11192, 11213, 11389, 11461,
<code>\displaywidowpenalty</code>	484	11475, 11510, 11535, 11550, 11565,
<code>\displaywidth</code>	486	11598, 11615, 11619, 11638, 11653,
<code>\divide</code>	363	11695, 11707, 11724, 11739, 11767,
<code>\doublehyphendemerits</code>	553	11769, 11779, 11795, 11800, 11815,
<code>\dp</code>	664	11852, 11858, 11863, 11896, 11913,
<code>\driver_box_rotate_begin:</code>	6610	11917, 11934, 11946, 11949, 11952,
<code>\driver_box_rotate_end:</code>	6612	11961, 11965, 11992, 11995, 12020,
<code>\driver_box_scale_begin:</code>	6827	12090, 12102, 12113, 12129, 12134,
<code>\driver_box_scale_end:</code>	6829	12139, 12144, 12152, 12168, 12182,
<code>\driver_box_use_clip:N</code>	6848	12188, 12213, 12232, 12267, 12275,
<code>\driver_color_ensure_current:</code> ...	8019	12299, 12302, 12345, 12351, 12356,
<code>\dump</code>	647	12409, 12417, 12441, 12455, 12458,
		12472, 12490, 12496, 12536, 12544,
		12572, 12650, 12658, 12681, 12710,
		12716, 12755, 12762, 12772, 12784,
E		
<code>\E</code>	1840, 2724	12798, 12806, 12819, 12833, 12842,
<code>\edef</code>	33,	12848, 12932, 12940, 12990, 12994,
68, 82, 164, 166, 181, 201, 266, 273, 351		12998, 13002, 13017, 13021, 13026,
<code>\else</code> 14, 22, 63, 117, 137, 172, 187, 207, 404		13033, 13037, 13047, 13051, 13054,
<code>\else:</code> 785, 788, 1082, 1099, 1102,		13065, 13069, 13073, 13078, 13083,
1111, 1117, 1127, 1130, 1139, 1145,		13097, 13101, 13105, 13110, 13115
1299, 1310, 1335, 1420, 1483, 1488,		
1525, 1576, 1926, 1940, 1954, 1964,	<code>\emergencystretch</code>	568
1975, 1978, 1988, 1991, 2004, 2013,	<code>\end</code>	442
2270, 2272, 2274, 2276, 2420, 2436,	<code>\EndCatcodeRegime</code>	13321
2446, 2454, 2467, 2476, 2610, 2615,	<code>\endcsname</code> 13, 32, 35, 62, 80, 93, 96, 167,	
2620, 2625, 2632, 2638, 2643, 2648,	170, 176, 185, 190, 192, 201, 204,	
2653, 2658, 2663, 2668, 2673, 2678,	214, 229, 233, 269, 271, 276, 278, 444	
2698, 2706, 2713, 2747, 2758, 2769,	<code>\endgroup</code> .. 12, 61, 111, 120, 228, 232, 377	
2780, 2842, 2851, 2859, 2868, 2934,	<code>\endinput</code>	264, 416
2942, 2964, 3239, 3250, 3260, 3265,	<code>\endL</code>	731
3268, 3271, 3364, 3375, 3383, 3391,	<code>\endlinechar</code>	79, 92, 289, 458
3399, 3407, 3415, 3423, 3431, 3439,	<code>\endR</code>	733
3447, 3650, 3986, 3993, 4149, 4490,	<code>\eqno</code>	478
4502, 4515, 4525, 4541, 4617, 4719,	<code>\errhelp</code>	250, 424
4739, 4754, 4762, 4772, 4790, 4802,	<code>\errmessage</code>	418
4839, 6168, 6193, 6438, 6440, 6450,	<code>\ERROR</code>	2390, 2391
8478, 8489, 8492, 9949, 9978, 10024,	<code>\errorcontextlines</code>	425
10035, 10085, 10091, 10101, 10193,	<code>\errorstopmode</code>	439
10235, 10278, 10282, 10299, 10343,	<code>\escapechar</code>	457
10351, 10354, 10363, 10391, 10410,	<code>\etex_beginL:D</code>	730
10419, 10487, 10490, 10507, 10510,	<code>\etex_beginR:D</code>	732
10526, 10529, 10552, 10555, 10558,	<code>\etex_botmarks:D</code>	679

<code>\etex_clubpenalties:D</code>	721	<code>\etex_showtokens:D</code>	
<code>\etex_currentgrouplevel:D</code>	695		685, 4814, 5349, 5903, 7971, 8245, 8265
<code>\etex_currentgroupstype:D</code>	696	<code>\etex_splitbotmarks:D</code>	681
<code>\etex_currentifbranch:D</code>	692	<code>\etex_splitdiscards:D</code>	728
<code>\etex_currentiflevel:D</code>	691	<code>\etex_splitfirstmarks:D</code>	680
<code>\etex_currentifttype:D</code>	693	<code>\etex_TeXETstate:D</code>	729
<code>\etex_detokenize:D</code>	683, 4609, 4610	<code>\etex_topmarks:D</code>	677
<code>\etex_dimexpr:D</code>	710, 3922	<code>\etex_tracingassigns:D</code>	687
<code>\etex_displaywidowpenalties:D</code>	723	<code>\etex_tracinggroups:D</code>	694
<code>\etex_endL:D</code>	731	<code>\etex_tracingifs:D</code>	690
<code>\etex_endR:D</code>	733	<code>\etex_tracingnesting:D</code>	689
<code>\etex_eTeXrevision:D</code>	675	<code>\etex_tracingscantokens:D</code>	688
<code>\etex_eTeXversion:D</code>	674	<code>\etex_unexpanded:D</code>	682,
<code>\etex_everyeof:D</code> ..	735, 4366, 4392, 4405		803, 1767, 1769, 1772, 1777, 4646, 4937
<code>\etex_firstmarks:D</code>	678	<code>\etex_unless:D</code>	673, 790
<code>\etex_fontcharhp:D</code>	703	<code>\etex_widowpenalties:D</code>	722
<code>\etex_fontcharht:D</code>	702	<code>\eTeXrevision</code>	675
<code>\etex_fontcharic:D</code>	705	<code>\eTeXversion</code>	674
<code>\etex_fontcharwd:D</code>	704	<code>\everycr</code>	386
<code>\etex_glueexpr:D</code>	711, 4123,	<code>\everydisplay</code>	487
	4134, 4139, 4164, 4169, 4172, 10215	<code>\everyeof</code>	735
<code>\etex_glueshrink:D</code>	714, 4234	<code>\everyhbox</code>	626
<code>\etex_glueshrinkorder:D</code>	716, 4158	<code>\everyjob</code>	31, 655
<code>\etex_gluestretch:D</code>	713, 4233	<code>\everymath</code>	511
<code>\etex_gluestretchorder:D</code>	715, 4157	<code>\everypar</code>	574
<code>\etex_gluetomu:D</code>	717	<code>\everyvbox</code>	627
<code>\etex_ifcurname:D</code>	672, 800	<code>\exhyphenpenalty</code>	550
<code>\etex_ifdefined:D</code>	671, 799	<code>\exp_after:wN</code>	
<code>\etex_iffontchar:D</code>	701		29, 801, 801, 819, 856, 891, 893,
<code>\etex_interactionmode:D</code>	699		977, 1045, 1063, 1075, 1081, 1083,
<code>\etex_interlinepenalties:D</code>	720		1110, 1112, 1115, 1138, 1140, 1143,
<code>\etex_lastlinefit:D</code>	719		1298, 1300, 1309, 1311, 1314, 1357,
<code>\etex_lastnodetype:D</code>	700		1389, 1536, 1543, 1545, 1548, 1549,
<code>\etex_marks:D</code>	676		1556, 1560, 1561, 1566, 1567, 1572,
<code>\etex_middle:D</code>	724		1577, 1579, 1582, 1590, 1592, 1594,
<code>\etex_muexpr:D</code> ..	712, 4198, 4209, 4214, 4219		1596, 1598, 1600, 1603–1605, 1609,
<code>\etex_mutogluue:D</code>	718		1612, 1617, 1622–1624, 1628–1630,
<code>\etex_numexpr:D</code>	709, 3214		1634–1636, 1640–1642, 1646–1648,
<code>\etex_pagediscards:D</code>	727		1652–1655, 1659–1662, 1666–1668,
<code>\etex_parshapedimen:D</code>	708		1673–1676, 1680–1683, 1715, 1716,
<code>\etex_parshapeindent:D</code>	706		1719, 1722, 1723, 1727, 1728, 1731,
<code>\etex_parshapelength:D</code>	707		1733, 1734, 1736, 1739, 1740, 1745,
<code>\etex_predisplaydirection:D</code>	734		1746, 1750, 1756–1758, 1762, 1764,
<code>\etex_protected:D</code>	736, 818		1766, 1767, 1769, 1772, 1777, 1780,
<code>\etex_readline:D</code>	686, 8504, 8506		1793, 1799, 1848, 1873, 1974, 1977,
<code>\etex_savinghyphcodes:D</code>	725		1979, 1987, 1990, 1992, 1997, 1998,
<code>\etex_savingvdiscards:D</code>	726		2001, 2010, 2018, 2020–2022, 2025,
<code>\etex_scantokens:D</code> ..	684, 4371, 4396, 4409		2030, 2178, 2304, 2413, 2419, 2421,
<code>\etex_showgroups:D</code>	697		2428, 2435, 2437, 2691, 2712, 2731,
<code>\etex_showifs:D</code>	698		2738, 2748, 2759, 2770, 2781, 2790,

2798, 2806, 2826, 2849, 2850, 2852,
 2858, 2861, 2933, 2935, 2941, 2943,
 2956, 2963, 2965, 3054, 3063, 3072,
 3358, 3361, 3622, 3650, 3661, 3671,
 3804, 3990, 4079, 4371, 4409, 4446,
 4454, 4459, 4500, 4512, 4513, 4567,
 4568, 4610, 4616, 4678, 4682, 4684,
 4698, 4706, 4716, 4732, 4752, 4761,
 4764, 4782–4784, 4810, 4816, 4876,
 4903, 4987, 4988, 5149, 5166, 5179,
 5198, 5199, 5227, 5228, 5250, 5255,
 5349, 5350, 5364, 5370, 5391, 5398,
 5466–5468, 5475, 5476, 5693, 5703,
 5764, 5772, 5777, 5903, 5904, 5994,
 6075, 6230, 6266, 6267, 6311, 7971,
 7972, 8245, 8246, 8265, 8266, 8470,
 8477, 8480, 8651, 9070–9073, 9128,
 9130, 9169, 9277, 9405, 9758, 9770,
 9841, 9944, 9964, 9969, 9973, 9977,
 9980, 9984, 9987, 10006, 10025,
 10034, 10036, 10046, 10048, 10063,
 10065, 10077, 10092, 10100, 10102,
 10109, 10112, 10124, 10134, 10137,
 10149, 10198, 10219, 10240, 10269,
 10277, 10280, 10283, 10298, 10300,
 10334, 10342, 10344, 10362, 10364,
 10401, 10409, 10411, 10418, 10420,
 10486, 10489, 10491, 10503, 10522,
 10637, 10652, 10673, 10688, 10702,
 10741, 10761, 10787, 10792, 10793,
 10819, 10821, 10841, 10962, 11010,
 11021, 11063, 11105, 11121, 11127,
 11135, 11142–11144, 11351, 11353,
 11363, 11378, 11381, 11411, 11414,
 11425, 11428, 11444, 11460, 11466,
 11474, 11480–11482, 11551, 11564,
 11576, 11603, 11620, 11658, 11669,
 11671, 11673, 11675, 11700, 11708,
 11744, 11755, 11757, 11759, 11761,
 11807, 11822, 11876, 11901, 11918,
 11933, 11937, 11962, 11966, 11993,
 11996, 12025, 12095, 12103, 12126–
 12128, 12130–12132, 12136–12138,
 12145, 12151, 12157, 12167, 12171,
 12184–12186, 12190, 12191, 12204,
 12249, 12317, 12369, 12408, 12415,
 12423, 12434, 12442, 12477, 12494,
 12532, 12535, 12571, 12573, 12584,
 12592, 12609, 12657, 12659, 12671,
 12674, 12721, 12773, 12783, 12795–
 12797, 12818, 12827–12831, 12835–
 12840, 12844–12846, 12849, 12861
 \exp_arg_last_unbraced:nn
 .. 1712, 1712, 1715, 1719, 1722, 1727
 \exp_arg_next:nnn 1537,
 1537, 1545, 1548, 1556, 1560, 1566
 \exp_arg_next_nobrace:nnn 1537, 1538, 1543
 \exp_args:cc 1595, 1595
 \exp_args:Nc 26, 819,
 819–822, 984, 992, 1000, 1008, 1224,
 1234, 1252, 1278, 1286, 1291, 1321,
 1346, 1422–1425, 1443, 1595, 4581
 \exp_args:Ncc 1280,
 1288, 1292, 1430–1433, 1595, 1599
 \exp_args:Nccc 1595, 1601
 \exp_args:Ncco 1657, 1678
 \exp_args:Nccx 1700, 1709
 \exp_args:Ncf 1620, 1644
 \exp_args:NcNc 1657, 1664
 \exp_args:NcNo 1657, 1671
 \exp_args:Ncnx 1700, 1710
 \exp_args:Nco 1620, 1638
 \exp_args:Ncx 1686, 1695
 \exp_args:Nf 27, 1608,
 1608, 2117, 2130, 2221, 2222, 2231,
 2240, 3524, 3593, 3605, 3614, 3707,
 3720, 3734, 3744, 3755, 3766, 4902,
 4979, 5459, 5952, 5965, 5970, 5986
 \exp_args:Nff 1686, 1688
 \exp_args:Nfo 1686, 1687, 5940
 \exp_args:NNc 1048,
 1049, 1279, 1287, 1290, 1426–1429,
 1595, 1597, 1795, 2247, 2253, 5835
 \exp_args:Nnc 1686, 1686, 5845
 \exp_args:NNf 1620, 1620
 \exp_args:Nnf 927, 936, 945, 953, 1686, 1689
 \exp_args:Nnnc 1700, 1702
 \exp_args:NNNo . 28, 1590, 1593, 4372, 4397
 \exp_args:NNno 1700, 1700
 \exp_args:NNno 1700, 1703
 \exp_args:NNNV 1657, 1657
 \exp_args:NNnx 28, 1700, 1705
 \exp_args:Nnnx 1700, 1707
 \exp_args:NNo 27, 1590,
 1591, 3512, 4981, 6069, 8382, 9404
 \exp_args:Nno 27, 1686,
 1690, 3046, 3997, 5854, 6084, 9657
 \exp_args:NNoo 28, 1700, 1701
 \exp_args:NNox 1700, 1706
 \exp_args:Nnox 1700, 1708

- \exp_args:NNV 1620, 1632
- \exp_args:NNv 1620, 1626
- \exp_args:NnV 1686, 1691
- \exp_args:NNx 28, 1686, 1694
- \exp_args:Nnx 1686, 1696
- \exp_args:No 26, 1590, 1590,
3512, 3597, 4366, 4405, 4410, 4548–
4550, 4600, 4843–4846, 5563, 5636,
5771, 5790, 5795, 5980, 5984, 8466
- \exp_args:Noc 1686, 1692
- \exp_args:Noo 1686, 1693
- \exp_args:Nooo 1700, 1704
- \exp_args:Noox 1700, 1711
- \exp_args:Nox 1686, 1697
- \exp_args:NV 27, 1608, 1615
- \exp_args:Nv 27, 1608, 1610
- \exp_args:NVV 1620, 1650
- \exp_args:Nx 27, 1608, 1685, 1685
- \exp_args:Nxo 1686, 1698
- \exp_args:Nxx 1686, 1699
- \exp_eval_error_msg:w .. 1570, 1574, 1583
- \exp_eval_register:c 1567, 1570,
1581, 1613, 1630, 1728, 1733, 1778
- \exp_eval_register:N
..... 30, 1561, 1570, 1570,
1582, 1618, 1636, 1654, 1655, 1662,
1723, 1731, 1741, 1747, 1759, 1773
- \exp_last_two_unbraced:Noo
.... 29, 1763, 1763, 7240, 7472, 7476
- \exp_last_two_unbraced_aux:noN
..... 1764, 1765
- \exp_last_unbraced:NcV . 1730, 1737, 4591
- \exp_last_unbraced:Nf
.... 28, 1730, 1735, 2290, 3603, 6014
- \exp_last_unbraced:Nfo . 1730, 1753, 5440
- \exp_last_unbraced:NNNo 1730, 1761
- \exp_last_unbraced:NNNV 1730, 1754
- \exp_last_unbraced:NNo
.. 1730, 1749, 4889, 5817, 6224, 7446
- \exp_last_unbraced:Nno . 1730, 1751, 6305
- \exp_last_unbraced:NNV 1730, 1743
- \exp_last_unbraced:No
.. 1730, 1734, 7824, 7829, 7908, 7914
- \exp_last_unbraced:Noo
..... 1730, 1752, 6178, 6319
- \exp_last_unbraced:NV 1730, 1730
- \exp_last_unbraced:Nv 1730, 1732
- \exp_not:c 29, 1780,
1780, 1821, 1871, 2316, 2320, 2325,
2329, 2332, 2334–2336, 2341, 2343–
2345, 2350, 2352–2354, 2359, 2361–
2363, 2368, 2370, 2372, 2374, 2376,
2378, 2380, 2382, 2384–2386, 2993,
8705, 8707, 8709, 8711, 9075, 9153,
9175, 9295, 9297, 9310, 9312, 9450
- \exp_not:f 30, 1767, 1768
- \exp_not:N 29, 801,
802, 1356–1358, 1388–1390, 1536,
1572, 1780, 1821, 2258, 2321, 2324,
2328, 2332, 2341, 2350, 2359, 2427,
2434, 2465, 2474, 2605, 2609, 2614,
2619, 2624, 2631, 2637, 2642, 2647,
2652, 2657, 2662, 2672, 2677, 2705,
2712, 2896, 2901, 2919, 2932, 2962,
4084, 4085, 4088, 4377, 4379, 4392,
4439, 4440, 4714, 4716, 4730, 4732,
4760, 4767, 5299, 5562, 5564, 8380,
8450, 8798, 8806, 8807, 8809, 9075,
9076, 9295, 9297, 9310, 9312, 9338,
9339, 9364, 9365, 9410, 9432, 9433,
9450, 10189, 10231, 10250, 10643,
10679, 10756, 10943, 11050, 11060,
11613, 11626, 11713, 11911, 11924,
12108, 12413, 12421, 12447, 12640,
12642, 12644, 12891, 12893, 12895,
12897, 12899, 13128, 13130, 13132,
13134, 13136, 13138, 13140, 13142
- \exp_not:n 29, 801, 803,
1481, 1536, 2259, 2317, 2427, 2434,
2465, 2474, 2897, 2902, 2916, 2920,
2992, 2994, 4253, 4283, 4289, 4301,
4309, 4325, 4333, 4441, 4836, 5128,
5225, 5300, 5356, 5510, 5534, 5567,
5656, 5778, 5910, 6011, 6012, 6137,
6138, 6159, 6275, 7982, 8252, 8287,
8291, 8509, 8799, 8803, 8810, 9067,
9341, 9435, 9473, 13154, 13255, 13258
- \exp_not:o 30, 1767, 1767, 4285,
4291, 4301, 4303, 4305, 4307, 4309,
4311, 4313, 4315, 4325, 4327, 4329,
4331, 4333, 4335, 4337, 4339, 4451,
4463, 4808, 4836, 5072, 5074, 5124,
5611, 5613, 5677, 5771, 8671, 9155,
9177, 9180, 9187, 9196, 9648, 9650
- \exp_not:V
29, 1767, 1770, 4303, 4311, 4327, 4335
- \exp_not:v 30, 1767, 1775, 9367
- \exp_stop_f 30
- \exp_stop_f: . 1546, 1552, 2291, 4982, 5710

- \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, 1533,
1889, 2400, 2490, 3210, 3918, 4241,
5043, 5580, 6043, 6364, 6908, 8008,
8028, 8521, 9104, 9753, 9870, 13234
- \ExplFileDescription
. 113, 130, 334, 782, 1533,
1889, 2400, 2490, 3210, 3918, 4241,
5043, 5580, 6043, 6364, 6908, 8008,
8028, 8521, 9104, 9753, 9870, 13234
- \ExplFileName 114, 128, 334, 782, 1533,
1889, 2400, 2490, 3210, 3918, 4241,
5043, 5580, 6043, 6364, 6908, 8008,
8028, 8521, 9104, 9753, 9870, 13234
- \ExplFileVersion
. . . . 49, 115, 129, 334, 782, 1533,
1889, 2400, 2490, 3210, 3918, 4241,
5043, 5580, 6043, 6364, 6908, 8008,
8028, 8521, 9104, 9753, 9870, 13234
- \ExplSyntaxNamesOff 6, 266, 273
- \ExplSyntaxNamesOn 6, 266, 266
- \ExplSyntaxOff 3, 6,
67, 68, 178, 186, 208, 291, 296, 310, 325
- \ExplSyntaxOn 3,
6, 67, 82, 150, 155, 160, 206, 291, 292
- F**
- \F 2685, 2719, 2818
- \fam 366
- \fi 22, 44, 65,
123, 139, 174, 194, 209, 231, 265, 405
- \fi: 785,
789, 978, 1046, 1062, 1066, 1084,
1104, 1105, 1113, 1119, 1132, 1133,
1141, 1147, 1221, 1301, 1312, 1338,
1343, 1344, 1420, 1483, 1488, 1524–
1527, 1575, 1578, 1585, 1586, 1794,
1874, 1928, 1942, 1954, 1964, 1980,
1981, 1993, 1994, 2006, 2015, 2270,
2272, 2274, 2276, 2278, 2280, 2414,
2422, 2429, 2438, 2448, 2456, 2469,
2478, 2610, 2615, 2620, 2625, 2632,
2638, 2643, 2648, 2653, 2658, 2663,
2668, 2673, 2678, 2700, 2706, 2713,
2750, 2761, 2772, 2783, 2844, 2853,
2862, 2870, 2936, 2944, 2966, 3230,
3241, 3252, 3267, 3273, 3274, 3276,
3366, 3370, 3377, 3385, 3393, 3401,
3409, 3417, 3425, 3433, 3441, 3449,
3651, 3975, 3986, 3995, 4005, 4151,
4492, 4504, 4517, 4527, 4544, 4619,
4710, 4721, 4741, 4756, 4765, 4774,
4792, 4804, 4809, 4810, 4841, 4877,
5120, 5123, 5150, 5226, 5230, 5251,
5365, 6170, 6195, 6231, 6312, 6438,
6440, 6450, 8481, 8494, 8495, 9951,
9988, 9989, 10021, 10026, 10037,
10049, 10066, 10090, 10093, 10103,
10113, 10138, 10195, 10237, 10275,
10284, 10285, 10301, 10340, 10345,
10356, 10357, 10365, 10393, 10407,
10412, 10421, 10492, 10493, 10509,
10513, 10528, 10533, 10554, 10557,
10560, 10563, 10566, 10569, 10572,
10575, 10578, 10593, 10596, 10599,
10602, 10605, 10608, 10611, 10614,
10617, 10638, 10649, 10674, 10685,
10716, 10728, 10736–10738, 10742,
10784, 10822, 10838, 10864, 10879,
10893, 10896, 10956, 10959, 11064,
11065, 11099, 11102, 11129, 11130,
11145–11147, 11157, 11190, 11195,
11205, 11209, 11217, 11218, 11339,
11354, 11382, 11397, 11415, 11459,
11467, 11483–11485, 11498, 11502,
11523, 11524, 11533, 11542, 11552,
11578, 11579, 11600, 11622, 11629,
11642, 11655, 11676, 11697, 11709,
11728, 11741, 11762, 11766, 11771,
11772, 11799, 11804, 11808, 11823,
11856, 11862, 11870, 11874, 11875,
11877, 11898, 11920, 11927, 11938,
11948, 11954, 11955, 11964, 11967,
11994, 11997, 12022, 12092, 12104,
12115, 12133, 12142, 12143, 12146,
12158, 12187, 12192, 12193, 12215,
12224, 12235, 12244, 12284, 12285,
12297, 12305, 12306, 12349, 12355,
12363, 12367, 12368, 12370, 12416,
12424, 12443, 12461, 12462, 12474,
12492, 12501, 12525, 12538, 12552,
12574, 12580, 12585, 12593, 12599,
12602, 12653, 12660, 12675, 12689,
12714, 12720, 12722, 12761, 12775,

- 12776, 12805, 12812, 12813, 12841,
12847, 12851, 12852, 12934, 12942,
12993, 12997, 13001, 13005, 13024,
13025, 13036, 13039, 13040, 13056–
13058, 13086–13090, 13118–13122
- \file_add_path:nN
..... 163, 9784, 9784, 9823, 9830
- \file_add_path_search:nN 9784, 9788, 9794
- \file_if_exist:n 9821, 9821
- \file_if_exist:nTF 163
- \file_input:n 164, 9828, 9828
- \file_list 164
- \file_list: 9852, 9852
- \file_path_include:n ... 164, 9845, 9845
- \file_path_remove:n ... 164, 9845, 9850
- \finalhyphdemerits 554
- \firstmark 452
- \firstmarks 678
- \floatingpenalty 599
- \font 365
- \fontcharhp 703
- \fontcharht 702
- \fontcharic 705
- \fontcharwd 704
- \fontdimen 632
- \fontname 456
- \fp_abs:c 10744
- \fp_abs:N 169, 10744, 10744, 10746
- \fp_abs_aux:NN 10744, 10744, 10745, 10748
- \fp_add:cn 10795
- \fp_add:Nn 170, 6648,
7314, 7347, 7601, 10795, 10795, 10797
- \fp_add:NNNNNNNN
.. 11181, 11181, 12519, 12571, 12657
- \fp_add_aux:NNn 10795, 10795, 10796, 10799
- \fp_add_core: . 10795, 10809, 10812, 10913
- \fp_add_difference: . 10795, 10821, 10866
- \fp_add_sum: 10795, 10819, 10849
- \fp_compare:n 13148, 13148
- \fp_compare:NNN 12944, 12961
- \fp_compare:nNn 12944, 12944
- \fp_compare:NNNT 7669
- \fp_compare:NNNTF
..... 6577, 6579, 6593, 6595,
6600, 6713, 6715, 6758, 6778, 6796,
6798, 6809, 6818, 6833, 7684, 7687
- \fp_compare:nNnTF
..... 168, 7303, 13162, 13168,
13174, 13180, 13186, 13192, 13198
- \fp_compare:nTF 169
- \fp_compare_<: 12944
- \fp_compare_<_aux: 12944
- \fp_compare_>: 12944
- \fp_compare_absolute_a<b: 12944
- \fp_compare_absolute_a>b: 12944
- \fp_compare_aux:N
..... 12944, 12959, 12970, 12972
- \fp_compare_aux_i:w . 13148, 13154, 13158
- \fp_compare_aux_ii:w 13148, 13161, 13164
- \fp_compare_aux_iii:w 13148, 13167, 13170
- \fp_compare_aux_iv:w 13148, 13173, 13176
- \fp_compare_aux_v:w . 13148, 13179, 13182
- \fp_compare_aux_vi:w 13148, 13185, 13188
- \fp_compare_aux_vii:w 13148, 13191, 13194
- \fp_const:cn 10162
- \fp_const:Nn 165, 10162, 10162, 10167
- \fp_cos:cn 11678
- \fp_cos:Nn
.. 171, 6627, 7523, 11678, 11678, 11680
- \fp_cos_aux:NNn 11678, 11678, 11679, 11682
- \fp_cos_aux_i: 11678, 11708, 11718
- \fp_cos_aux_ii: 11678, 11721, 11751, 11956
- \fp_div:cn 11029
- \fp_div:Nn 170, 6624, 6708, 6712,
6756, 6776, 7301, 7302, 7323, 7345,
7520, 7656, 7659, 11029, 11029, 11031
- \fp_div_aux:NNn 11029, 11029, 11030, 11033
- \fp_div_divide: 11029, 11117, 11132, 11158
- \fp_div_divide_aux:
..... 11029, 11135, 11144, 11149
- \fp_div_integer:NNNNN . 11324, 11324,
11785, 11836, 11841, 12332, 12703
- \fp_div_internal:
.. 11029, 11063, 11068, 12621, 12879
- \fp_div_loop:
.. 11029, 11073, 11114, 11128, 12005
- \fp_div_loop_step:w 11121, 11175
- \fp_div_store: 11029,
11071, 11118, 11160, 11164, 12003
- \fp_div_store_decimal: 11029, 11164, 11166
- \fp_div_store_integer:
..... 11029, 11071, 11161, 12003
- \fp_exp:cn 12072
- \fp_exp:Nn 171, 12072, 12072, 12074
- \fp_exp_aux: . 12072, 12128, 12138, 12148
- \fp_exp_aux:NNn 12072, 12072, 12073, 12076
- \fp_exp_const:cx 12072, 12140, 12180, 12312
- \fp_exp_const:Nx 12072, 12372, 12377, 12925
- \fp_exp_decimal: 12072, 12157, 12245, 12260
- \fp_exp_integer: ... 12072, 12151, 12160

- \fp_exp_integer_const:n [12072](#), [12183](#), [12189](#),
[12212](#), [12214](#), [12231](#), [12233](#), [12247](#)
- \fp_exp_integer_const:nnnn [12072](#), [12250](#), [12253](#), [12610](#)
- \fp_exp_integer_tens:
.. [12072](#), [12167](#), [12186](#), [12191](#), [12195](#)
- \fp_exp_integer_units: [12072](#), [12225](#), [12227](#)
- \fp_exp_internal:
..... [12072](#), [12103](#), [12120](#), [12926](#)
- \fp_exp_overflow_msg:
..... [12132](#), [12145](#), [13208](#), [13214](#)
- \fp_exp_Taylor: [12072](#), [12290](#), [12324](#), [12369](#)
- \fp_extended_normalise: [11341](#),
[11341](#), [11454](#), [12123](#), [12788](#), [12823](#)
- \fp_extended_normalise_aux:NNNNNNNN [11341](#)
- \fp_extended_normalise_aux_i:
..... [11341](#), [11343](#), [11346](#), [11353](#)
- \fp_extended_normalise_aux_i:w
..... [11341](#), [11351](#), [11356](#)
- \fp_extended_normalise_aux_ii:
..... [11341](#), [11344](#), [11374](#), [11381](#)
- \fp_extended_normalise_aux_ii:w
..... [11341](#), [11363](#), [11366](#)
- \fp_extended_normalise_ii_aux:NNNNNNNN [11379](#), [11384](#)
- \fp_extended_normalise_output: [11407](#),
[11407](#), [11414](#), [12223](#), [12243](#), [12915](#)
- \fp_extended_normalise_output_aux:N
..... [11407](#), [11435](#), [11437](#)
- \fp_extended_normalise_output_aux_i:NNNNNNNN [10097](#), [10097](#), [10814](#)
..... [11407](#), [11412](#), [11417](#)
- \fp_extended_normalise_output_aux_ii:NNNNNNNN [10097](#), [10100](#), [10105](#), [10112](#)
..... [11407](#), [11428](#), [11431](#)
- \fp_gabs:c [10744](#)
- \fp_gabs:N [169](#), [10744](#), [10745](#), [10747](#)
- \fp_gadd:cn [10795](#)
- \fp_gadd:Nn [170](#), [10795](#), [10796](#), [10798](#)
- \fp_gcos:cn [11678](#)
- \fp_gcos:Nn [172](#), [11678](#), [11679](#), [11681](#)
- \fp_gdiv:cn [11029](#)
- \fp_gdiv:Nn [170](#), [11029](#), [11030](#), [11032](#)
- \fp_gexp:cn [12072](#)
- \fp_gexp:Nn [171](#), [12072](#), [12073](#), [12075](#)
- \fp_gln:cn [12390](#)
- \fp_gln:Nn [171](#), [12390](#), [12391](#), [12393](#)
- \fp_gmul:cn [10916](#)
- \fp_gmul:Nn [170](#), [10916](#), [10917](#), [10919](#)
- \fp_gneg:c [10769](#)
- \fp_gneg:N [169](#), [10769](#), [10770](#), [10772](#)
- \fp_gpow:cn [12724](#)
- \fp_gpow:Nn [171](#), [12724](#), [12725](#), [12727](#)
- \fp_ground_figures:cn [10625](#)
- \fp_ground_figures:Nn
..... [167](#), [10625](#), [10628](#), [10630](#)
- \fp_ground_places:cn [10660](#)
- \fp_ground_places:Nn
..... [168](#), [10660](#), [10663](#), [10665](#)
- \fp_gset:cn [10174](#)
- \fp_gset:Nn [166](#), [10165](#), [10174](#), [10175](#), [10207](#)
- \fp_gset_eq:cc [10258](#), [10265](#)
- \fp_gset_eq:cN [10258](#), [10263](#)
- \fp_gset_eq:Nc [10258](#), [10264](#)
- \fp_gset_eq:NN [166](#), [10258](#), [10262](#)
- \fp_gset_from_dim:cn [10208](#)
- \fp_gset_from_dim:Nn
..... [166](#), [10208](#), [10210](#), [10255](#)
- \fp_gsin:cn [11581](#)
- \fp_gsin:Nn [171](#), [11581](#), [11582](#), [11584](#)
- \fp_gsub:cn [10898](#)
- \fp_gsub:Nn [170](#), [10898](#), [10899](#), [10901](#)
- \fp_gtan:cn [11879](#)
- \fp_gtan:Nn [172](#), [11879](#), [11880](#), [11882](#)
- \fp_gzero:c [10168](#)
- \fp_gzero:N [166](#), [10168](#), [10170](#), [10173](#)
- \fp_if_undefined:N [12928](#), [12928](#)
- \fp_if_undefined:NTF [168](#)
- \fp_if_zero:N [12936](#), [12936](#)
- \fp_if_zero:NTF [168](#)
- \fp_level_input_exponents:
..... [10097](#), [10097](#), [10814](#)
- \fp_level_input_exponents_a:
..... [10097](#), [10100](#), [10105](#), [10112](#)
- \fp_level_input_exponents_a:NNNNNNNN
..... [10097](#), [10110](#), [10115](#)
- \fp_level_input_exponents_b:
..... [10097](#), [10102](#), [10130](#), [10137](#)
- \fp_level_input_exponents_b:NNNNNNNN
..... [10097](#), [10135](#), [10140](#)
- \fp_ln:cn [12390](#)
- \fp_ln:Nn [171](#), [12390](#), [12390](#), [12392](#)
- \fp_ln_aux: [12390](#), [12408](#), [12427](#)
- \fp_ln_aux:NNn [12390](#), [12390](#), [12391](#), [12394](#)
- \fp_ln_const:nn [12507](#), [12517](#), [12568](#), [12607](#)
- \fp_ln_error_msg:
..... [12415](#), [12423](#), [13216](#), [13219](#)
- \fp_ln_exponent: ... [12390](#), [12442](#), [12451](#)
- \fp_ln_exponent_tens: [12390](#)
- \fp_ln_exponent_tens:NN .. [12494](#), [12504](#)

- \fp_ln_exponent_units: [12390](#), [12502](#), [12514](#)
- \fp_ln_fixed: . [12390](#), [12622](#), [12667](#), [12674](#)
- \fp_ln_fixed_aux:NNNNNNNNN [12390](#), [12672](#), [12677](#)
- \fp_ln_integer_const:nn [12390](#)
- \fp_ln_internal: [12390](#), [12453](#), [12485](#), [12887](#)
- \fp_ln_mantissa: ... [12390](#), [12526](#), [12562](#)
- \fp_ln_mantissa_aux: [12390](#), [12566](#), [12587](#), [12592](#)
- \fp_ln_mantissa_divide_two: [12390](#), [12591](#), [12595](#)
- \fp_ln_normalise: [12390](#), [12518](#), [12528](#), [12535](#), [12569](#), [12655](#)
- \fp_ln_normalise_aux:NNNNNNNNN [12533](#), [12540](#)
- \fp_ln_nornalise_aux:NNNNNNNNN .. [12390](#)
- \fp_ln_Taylor: [12390](#), [12584](#), [12613](#)
- \fp_ln_Taylor_aux: [12390](#), [12635](#), [12692](#), [12721](#)
- \fp_mul:cn [10916](#)
- \fp_mul:Nn [170](#), [6625](#), [6635](#), [6636](#), [6646](#), [6647](#), [7312](#), [7317](#), [7346](#), [7521](#), [7594](#), [7595](#), [7599](#), [7600](#), [7697](#), [7700](#), [10916](#), [10916](#), [10918](#)
- \fp_mul:NNNNNN .. [11220](#), [11220](#), [11781](#), [11828](#), [11832](#), [12328](#), [12630](#), [12695](#)
- \fp_mul:NNNNNNNNN [11264](#), [11264](#), [12216](#), [12236](#), [12291](#), [12904](#)
- \fp_mul_aux:NNn [10916](#), [10916](#), [10917](#), [10920](#)
- \fp_mul_end_level: [10916](#), [10987](#), [10991](#), [10994](#), [10998](#), [11018](#), [11244](#), [11249](#), [11253](#), [11258](#), [11260](#), [11261](#), [11290](#), [11297](#), [11303](#), [11310](#), [11314](#), [11317](#), [11321](#)
- \fp_mul_end_level:NNNNNNNNN [10916](#), [11022](#), [11024](#)
- \fp_mul_internal: .. [10916](#), [10930](#), [10971](#)
- \fp_mul_product:NN [10979](#)–[10981](#), [10983](#)–[10986](#), [10988](#)–[10990](#), [10992](#), [10993](#), [10997](#), [11013](#), [11232](#)–[11237](#), [11239](#)–[11243](#), [11245](#)–[11248](#), [11250](#)–[11252](#), [11256](#), [11257](#), [11259](#), [11276](#)–[11281](#), [11283](#)–[11289](#), [11291](#)–[11296](#), [11298](#)–[11302](#), [11306](#)–[11309](#), [11311](#)–[11313](#), [11315](#), [11316](#), [11320](#)
- \fp_mul_split:NNNN [10916](#), [10973](#), [10975](#), [11001](#), [11222](#), [11224](#), [11226](#), [11228](#), [11266](#), [11268](#), [11270](#), [11272](#)
- \fp_mul_split:w [10916](#)
- \fp_mul_split_aux:w [11004](#), [11010](#)
- \fp_neg:c [10769](#)
- \fp_neg:N [169](#), [10769](#), [10769](#), [10771](#)
- \fp_neg:NN [10769](#)
- \fp_neg_aux:NN [10769](#), [10770](#), [10773](#)
- \fp_new:c [10156](#)
- \fp_new:N [165](#), [6553](#)–[6555](#), [6565](#)–[6569](#), [6695](#), [6696](#), [6913](#), [6932](#)–[6936](#), [6942](#), [6943](#), [6948](#)–[6951](#), [7646](#), [7647](#), [10156](#), [10156](#), [10161](#), [10164](#)
- \fp_overflow_msg: [10025](#), [10092](#), [13200](#), [13206](#)
- \fp_pow:cn [12724](#)
- \fp_pow:Nn [170](#), [12724](#), [12724](#), [12726](#)
- \fp_pow_aux:NNn [12724](#), [12724](#), [12725](#), [12728](#)
- \fp_pow_aux_i: [12724](#), [12774](#), [12779](#)
- \fp_pow_aux_ii: [12724](#), [12783](#), [12797](#), [12815](#)
- \fp_pow_aux_iii: ... [12724](#), [12840](#), [12869](#)
- \fp_pow_aux_iv: [12724](#), [12818](#), [12832](#), [12846](#), [12850](#), [12872](#), [12881](#)
- \fp_pow_negative: [12724](#)
- \fp_pow_positive: [12724](#)
- \fp_read:N [9943](#), [9943](#), [10634](#), [10669](#), [10751](#), [10776](#), [10802](#), [10905](#), [10923](#), [11036](#), [12731](#), [12964](#), [12969](#)
- \fp_read_aux:w [9943](#), [9944](#), [9945](#)
- \fp_round: ... [10637](#), [10673](#), [10696](#), [10696](#)
- \fp_round_aux:NNNNNNNNN [10696](#), [10703](#), [10705](#)
- \fp_round_figures:cn [10625](#)
- \fp_round_figures:Nn [167](#), [10625](#), [10625](#), [10627](#)
- \fp_round_figures_aux:NNn [10625](#), [10626](#), [10629](#), [10631](#)
- \fp_round_loop:N [10696](#), [10707](#), [10718](#), [10741](#)
- \fp_round_places:cn [10660](#)
- \fp_round_places:Nn [168](#), [10660](#), [10660](#), [10662](#)
- \fp_round_places_aux:NNn [10660](#), [10661](#), [10664](#), [10666](#)
- \fp_set:cn [10174](#)
- \fp_set:Nn [166](#), [6575](#), [6790](#), [6791](#), [7519](#), [7682](#), [7683](#), [10174](#), [10174](#), [10206](#)
- \fp_set_aux:NNn [10174](#), [10174](#)–[10176](#)
- \fp_set_eq:cc [10258](#), [10261](#)
- \fp_set_eq:cN [10258](#), [10259](#)
- \fp_set_eq:Nc [10258](#), [10260](#)
- \fp_set_eq:NN [165](#), [6623](#), [6633](#), [6634](#), [6644](#), [6645](#), [6757](#), [6777](#), [7592](#), [7593](#), [7597](#), [7598](#), [10258](#), [10258](#)
- \fp_set_from_dim:cn [10208](#)

\fp_set_from_dim:Nn 166, 6631,
6632, 6642, 6643, 6706, 6707, 6709,
6710, 6753, 6754, 6774, 6775, 7297–
7300, 7307, 7308, 7311, 7316, 7339–
7343, 7590, 7591, 7654, 7655, 7657,
7658, 7696, 7699, 10208, 10208, 10254
\fp_set_from_dim_aux:NNn
..... 10208, 10209, 10211, 10212
\fp_set_from_dim_aux:w
.. 10208, 10219, 10248, 10250, 10253
\fp_show:c 10266, 10267
\fp_show:N 166, 10266, 10266
\fp_sin:cn 11581
\fp_sin:Nn
. 171, 6626, 7522, 11581, 11581, 11583
\fp_sin_aux:NNn 11581, 11581, 11582, 11585
\fp_sin_aux_i: 11581, 11621, 11632
\fp_sin_aux_ii: 11581, 11635, 11665, 11979
\fp_split:Nn 9956,
9956, 10179, 10217, 10803, 10906,
10924, 11037, 11588, 11685, 11886,
12079, 12397, 12736, 12947, 12953
\fp_split_aux_i:w 9956, 9995, 9999
\fp_split_aux_ii:w ... 9956, 10000, 10001
\fp_split_aux_iii:w .. 9956, 10002, 10003
\fp_split_decimal:w .. 9956, 10006, 10009
\fp_split_decimal_aux:w 9956, 10010, 10011
\fp_split_exponent: 9956
\fp_split_exponent:w 9964, 9991
\fp_split_sign: 9956, 9962, 9966, 9977, 9987
\fp_standardise:NNNN 10028,
10028, 10180, 10222, 10804, 10824,
10907, 10925, 10935, 11038, 11078,
11589, 11643, 11686, 11729, 11887,
11974, 11983, 12010, 12080, 12307,
12398, 12463, 12737, 12948, 12954
\fp_standardise_aux:
..... 10028, 10042, 10048,
10058, 10059, 10065, 10082, 10095
\fp_standardise_aux:NNNN
..... 10028, 10036, 10040
\fp_standardise_aux:w 10028,
10046, 10052, 10064, 10069, 10096
\fp_sub:cn 10898
\fp_sub:Nn 170, 6637, 7309, 7319,
7321, 7344, 7596, 10898, 10898, 10900
\fp_sub:NNNNNNNN 11197, 11197,
11536, 11557, 11568, 12573, 12659
\fp_sub_aux:NNn 10898, 10898, 10899, 10902
\fp_tan:cn 11879
\fp_tan:Nn 172, 11879, 11879, 11881
\fp_tan_aux:NNn 11879, 11879, 11880, 11883
\fp_tan_aux_i: 11879, 11919, 11930
\fp_tan_aux_ii: 11879, 11933, 11940
\fp_tan_aux_iii: 11879, 11963, 11966, 11969
\fp_tan_aux_iv: 11879, 11993, 11996, 11999
\fp_tmp:w 10155,
10155, 10186, 10204, 10228, 10246,
10640, 10658, 10676, 10694, 10753,
10767, 10810, 10829, 10914, 10940,
10969, 11047, 11057, 11066, 11083,
11610, 11623, 11630, 11710, 11716,
11908, 11921, 11928, 12105, 12118,
12410, 12418, 12425, 12444, 12636,
12647, 12750, 12756, 12767, 12777,
12800, 12807, 12853, 12888, 12902
\fp_to_dim:c 10331
\fp_to_dim:N
167, 6638, 6649, 7326, 7348, 7602,
7603, 7698, 7701, 10331, 10331, 10332
\fp_to_int:c 10333
\fp_to_int:N 167, 10333, 10333, 10335
\fp_to_int_aux:w ... 10333, 10334, 10336
\fp_to_int_large:w .. 10333, 10344, 10359
\fp_to_int_large_aux:nnn
10333, 10371, 10373, 10375, 10377,
10379, 10381, 10383, 10385, 10387
\fp_to_int_large_aux_1:w 10333
\fp_to_int_large_aux_2:w 10333
\fp_to_int_large_aux_3:w 10333
\fp_to_int_large_aux_4:w 10333
\fp_to_int_large_aux_5:w 10333
\fp_to_int_large_aux_6:w 10333
\fp_to_int_large_aux_7:w 10333
\fp_to_int_large_aux_8:w 10333
\fp_to_int_large_aux_i:w
..... 10333, 10362, 10368
\fp_to_int_large_aux_ii:w
..... 10333, 10364, 10395
\fp_to_int_none:w 10333
\fp_to_int_small:w .. 10333, 10342, 10348
\fp_to_tl:c 10400
\fp_to_tl:N 167, 10400, 10400, 10402
\fp_to_tl_aux:w 10400, 10401, 10403
\fp_to_tl_large:w .. 10400, 10411, 10415
\fp_to_tl_large_0:w 10400
\fp_to_tl_large_1:w 10400
\fp_to_tl_large_2:w 10400
\fp_to_tl_large_3:w 10400
\fp_to_tl_large_4:w 10400

<code>\fp_to_tl_large_5:w</code>	10400	<code>\fp_use_large:w</code>	10268 , 10277 , 10295
<code>\fp_to_tl_large_6:w</code>	10400	<code>\fp_use_large_aux_1:w</code>	10268
<code>\fp_to_tl_large_7:w</code>	10400	<code>\fp_use_large_aux_2:w</code>	10268
<code>\fp_to_tl_large_8:w</code>	10400	<code>\fp_use_large_aux_3:w</code>	10268
<code>\fp_to_tl_large_8_aux:w</code>	10400	<code>\fp_use_large_aux_4:w</code>	10268
<code>\fp_to_tl_large_9:w</code>	10400	<code>\fp_use_large_aux_5:w</code>	10268
<code>\fp_to_tl_large_aux_i:w</code>	10400 , 10418 , 10424	<code>\fp_use_large_aux_6:w</code>	10268
<code>\fp_to_tl_large_aux_ii:w</code>	10400 , 10420 , 10426	<code>\fp_use_large_aux_7:w</code>	10268
<code>\fp_to_tl_large_zeros:NNNNNNNN</code>	10400 , 10429 , 10435 , 10440 , 10445 , 10450 , 10455 , 10460 , 10465 , 10470 , 10480 , 10538 , 10541	<code>\fp_use_large_aux_8:w</code>	10268
<code>\fp_to_tl_small:w</code> ..	10400 , 10409 , 10483	<code>\fp_use_large_aux_i:w</code> ..	10268 , 10298 , 10304
<code>\fp_to_tl_small_aux:w</code> ..	10400 , 10491 , 10535	<code>\fp_use_large_aux_ii:w</code> ..	10268 , 10300 , 10325
<code>\fp_to_tl_small_one:w</code> ..	10400 , 10486 , 10496	<code>\fp_use_none:w</code>	10268 , 10283 , 10288
<code>\fp_to_tl_small_two:w</code> ..	10400 , 10489 , 10515	<code>\fp_use_small:w</code>	10268 , 10281 , 10289
<code>\fp_to_tl_small_zeros:NNNNNNNN</code>	10400 , 10503 , 10512 , 10522 , 10532 , 10580	<code>\fp_zero:c</code>	10168
<code>\fp_trig_calc_cos:</code>	11671 , 11673 , 11755 , 11761 , 11774 , 11774	<code>\fp_zero:N</code>	166 , 10168 , 10168 , 10172
<code>\fp_trig_calc_sin:</code>	11669 , 11675 , 11757 , 11759 , 11774 , 11810	<code>\frozen@everydisplay</code>	767
<code>\fp_trig_calc_Taylor:</code> 11774 , 11807 , 11822 , 11825 , 11876	<code>\frozen@everymath</code>	768
<code>\fp_trig_normalise:</code> 11450 , 11450 , 11634 , 11720 , 11942	<code>\futurelet</code>	361
<code>\fp_trig_normalise_aux:</code> 11450 , 11455 , 11469 , 11474 , 11482		
<code>\fp_trig_octant:</code> ...	11460 , 11526 , 11526		
<code>\fp_trig_octant_aux_i:</code> 11526 , 11529 , 11544 , 11564 , 11577		
<code>\fp_trig_octant_aux_ii:</code> 11526 , 11551 , 11554		
<code>\fp_trig_overflow_msg:</code> 11466 , 11937 , 13222 , 13228		
<code>\fp_trig_sub:NNN</code> ..	11450 , 11472 , 11478 , 11487		
<code>\fp_use:c</code>	10268		
<code>\fp_use:N</code>	166 , 6732 , 6734 , 6738 , 6740 , 10268 , 10268 , 10270 , 10331		
<code>\fp_use_aux:w</code>	10268 , 10269 , 10271		
<code>\fp_use_i_to_iix:NNNNNNNN</code>	10400 , 10500 , 10505 , 10623		
<code>\fp_use_i_to_vii:NNNNNNNN</code>	10400 , 10519 , 10524 , 10621		
<code>\fp_use_iix_ix:NNNNNNNN</code>	10400 , 10517 , 10619		
<code>\fp_use_ix:NNNNNNNN</code> ..	10400 , 10498 , 10620		

G

<code>\G</code>	2724
<code>\g</code>	2283
<code>\g_cctab_allocate_int</code>	13260 , 13260 , 13261 , 13268 , 13270 , 13272
<code>\g_cctab_stack_int</code>	13260 , 13262 , 13298 , 13299 , 13301 , 13302 , 13306
<code>\g_cctab_stack_seq</code>	13260 , 13263 , 13296 , 13307
<code>\g_clist_map_inline_int</code>	5828 , 5828 , 5831 , 5832 , 5836 , 5837 , 5841 , 5842 , 5846 , 5847
<code>\g_file_current_name_tl</code>	163 , 9756 , 9756 , 9761 , 9765 , 9773 , 9839 , 9840 , 9842
<code>\g_file_record_seq</code>	164 , 9768 , 9768 , 9773 , 9834 , 9854 , 9856 , 9863
<code>\g_file_stack_seq</code>	164 , 9767 , 9767 , 9839 , 9842
<code>\g_file_test_stream</code>	9784 , 9786 , 9787 , 9790 , 9808 , 9809 , 9819
<code>\g_ior_streams_prop</code>	8056 , 8057 , 8062 , 8097 , 8199 , 8214 , 8236 , 8244
<code>\g_ior_tmp_stream</code> ..	8132 , 8178 , 8179 , 8182
<code>\g_iow_streams_prop</code>	8056 , 8056 , 8059 – 8061 , 8110 , 8118 , 8126 , 8162 , 8227 , 8256 , 8264
<code>\g_iow_tmp_stream</code> ..	8132 , 8141 , 8142 , 8145
<code>\g_keyval_level_int</code>	9107 , 9107 , 9154 , 9176 , 9200 , 9202 , 9204 , 9206
<code>\g_peek_token</code>	54 , 2872 , 2873 , 2883

<code>\g_prg_stepwise_level_int</code>	10750, 10775, 10801, 10904, 10922,
.. 2244 , 2244 , 2249 , 2255 , 2265 , 2267	11035, 11587, 11684, 11885, 12078,
<code>\g_prop_map_inline_int</code>	12396, 12615, 12730, 12787, 12821,
.. 6237 , 6237 , 6240 , 6241 , 6244 , 6245	12883, 12946, 12963, 13150, 13327
<code>\g_seq_nesting_depth_int</code>	<code>\group_end</code>
.. 3872 , 5266 , 5278 , 5280 , 5284 , 5286	9
<code>\g_tl_inline_level_int</code>	<code>\group_end:</code>
..... 3873 , 4576 , 4578 , 4579 ,	810 ,
4582 , 4584 , 4588 , 4589 , 4592 , 4594	812 , 857 , 1072 , 1845 , 2287 , 2301 ,
<code>\g_tmpa_bool</code>	2602 , 2606 , 2634 , 2688 , 2728 , 2820 ,
36 , 1934 , 1935	3045 , 3086 , 3095 , 4355 , 4372 , 4397 ,
<code>\g_tmpa_clist</code>	4410 , 4539 , 4542 , 5154 , 6584 , 6720 ,
113 , 5916	6761 , 6781 , 6803 , 8015 , 8318 , 8338 ,
<code>\g_tmpa_dim</code>	8382 , 8611 , 8659 , 9080 , 9119 , 9130 ,
75 , 4105 , 4108	10188 , 10230 , 10642 , 10678 , 10755 ,
<code>\g_tmpa_int</code>	10792 , 10831 , 10942 , 11049 , 11059 ,
67 , 3867 , 3870	11085 , 11612 , 11625 , 11712 , 11910 ,
<code>\g_tmpa_skip</code>	11923 , 12107 , 12412 , 12420 , 12446 ,
78 , 4179 , 4182	12638 , 12752 , 12758 , 12769 , 12793 ,
<code>\g_tmpa_tl</code>	12799 , 12802 , 12809 , 12826 , 12834 ,
94 , 4829 , 4829 , 5914	12843 , 12855 , 12890 , 12977 , 12988 ,
<code>\g_tmpb_clist</code>	12991 , 12995 , 12999 , 13003 , 13015 ,
113 , 5917	13019 , 13022 , 13031 , 13034 , 13045 ,
<code>\g_tmpb_dim</code>	13049 , 13063 , 13067 , 13071 , 13076 ,
75 , 4105 , 4109	13081 , 13084 , 13095 , 13099 , 13103 ,
<code>\g_tmpb_int</code>	13108 , 13113 , 13116 , 13153 , 13330
67 , 3867 , 3871	<code>\group_execute_after:N</code>
<code>\g_tmpb_skip</code>	1519
78 , 4179 , 4183	<code>\group_insert_after:N</code>
<code>\g_tmpb_tl</code>	9 , 815 , 815 , 1519
94 , 4829 , 4830 , 5914	
<code>\gdef</code>	
352	
<code>\GetIdInfo</code>	
6 , 97 , 98	
<code>\GetIdInfoAuxCVS</code>	
97 , 136 , 141	
<code>\GetIdInfoAuxI</code>	
97 , 102 , 104	
<code>\GetIdInfoAuxII</code>	
97 , 121 , 126	
<code>\GetIdInfoAuxIII</code>	
97 , 131 , 133	
<code>\GetIdInfoAuxSVN</code>	
97 , 138 , 143	
<code>\GetIdInfoFull</code>	
97	
<code>\global</code>	
336 , 367	
<code>\globaldefs</code>	
371	
<code>\glueexpr</code>	
711	
<code>\glueshrink</code>	
714	
<code>\glueshrinkorder</code>	
716	
<code>\gluestretch</code>	
713	
<code>\gluestretchorder</code>	
715	
<code>\gluetomu</code>	
717	
<code>\group_align_safe_begin</code>	
41	
<code>\group_align_safe_begin:</code>	
..... 1946 , 2277 , 2277 , 2904 , 2922	
<code>\group_align_safe_end</code>	
41	
<code>\group_align_safe_end:</code> ..	
2043 , 2044 ,	
2277 , 2279 , 2886 , 2896 , 2901 , 2919	
<code>\group_begin</code>	
9	
<code>\group_begin:</code> 810 , 811 , 854 , 1067 , 1833 ,	
2282 , 2296 , 2590 , 2603 , 2627 , 2680 ,	
2717 , 2814 , 2980 , 3077 , 3084 , 4348 ,	
4365 , 4391 , 4404 , 4535 , 5143 , 6574 ,	
6701 , 6748 , 6769 , 6789 , 8011 , 8314 ,	
8319 , 8348 , 8606 , 8653 , 9057 , 9112 ,	
9122 , 10178 , 10214 , 10633 , 10668 ,	

H

<code>\H</code>	2724
<code>\halign</code>	378
<code>\hangafter</code>	556
<code>\hangindent</code>	557
<code>\hbadness</code>	618
<code>\hbox</code>	613
<code>\hbox:n</code> ..	127 , 6476 , 6476 , 6608 , 7791 , 7846
<code>\hbox_gset:cn</code>	6477
<code>\hbox_gset:cw</code>	6487 , 6499
<code>\hbox_gset:Nn</code>	128 , 6477 , 6478 , 6480
<code>\hbox_gset:Nw</code> ..	129 , 6487 , 6489 , 6492 , 6498
<code>\hbox_gset_end</code>	129
<code>\hbox_gset_end:</code>	6487 , 6494 , 6500
<code>\hbox_gset_inline_begin:c</code> ..	6495 , 6499
<code>\hbox_gset_inline_begin:N</code> ..	6495 , 6498
<code>\hbox_gset_inline_end:</code>	6495 , 6500
<code>\hbox_gset_to_wd:cnn</code>	6481
<code>\hbox_gset_to_wd:Nnn</code> ..	128 , 6481 , 6483 , 6486
<code>\hbox_overlap_left:n</code> ..	128 , 6504 , 6504
<code>\hbox_overlap_right:n</code> ..	128 , 6504 , 6506 , 6828
<code>\hbox_set:cn</code>	6477
<code>\hbox_set:cw</code>	6487 , 6496

- \hbox_set:Nn 128,
6477, 6477–6479, 6572, 6604, 6605,
6699, 6746, 6767, 6787, 6825, 6848,
6853, 6861, 6871, 6883, 6890, 6897,
7000, 7097, 7354, 7425, 7534, 7937
 - \hbox_set:Nw
128, 6487, 6487, 6490, 6491, 6495, 7044
 - \hbox_set_end 128
 - \hbox_set_end: ... 6487, 6493, 6497, 7048
 - \hbox_set_inline_begin:c 6495, 6496
 - \hbox_set_inline_begin:N 6495, 6495
 - \hbox_set_inline_end: 6495, 6497
 - \hbox_set_to_wd:cnn 6481
 - \hbox_set_to_wd:Nnn
..... 128, 6481, 6481, 6484, 6485
 - \hbox_to_wd:nn 128, 6501, 6501, 6835
 - \hbox_to_zero:n 128, 6501, 6503, 6505, 6507
 - \hbox_unpack:c 6508
 - \hbox_unpack:N
... 129, 6508, 6508, 6510, 7358, 7507
 - \hbox_unpack_clear:c 6508
 - \hbox_unpack_clear:N 129, 6508, 6509, 6511
 - \hcoffin_set:cn 6996
 - \hcoffin_set:cw 7040
 - \hcoffin_set:Nn 133, 6996,
6996, 7012, 7788, 7800, 7843, 7884
 - \hcoffin_set:Nw ... 133, 7040, 7040, 7056
 - \hcoffin_set_end 133
 - \hcoffin_set_end: 7040, 7045, 7055
 - \Height 7137, 7139, 7143, 7147, 7154
 - \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
 - \if:w 22, 785,
791, 976, 1044, 1060, 1792, 2857,
2962, 3363, 9947, 10273, 10338, 10405
 - \if_bool:N 41, 1892, 1892
 - \if_box_empty:N ... 132, 6434, 6436, 6450
 - \if_case:w 69,
1324, 3213, 3218, 3623, 11667, 11753
 - \if_catcode:w 22, 785, 793,
2609, 2614, 2619, 2624, 2631, 2637,
2642, 2647, 2652, 2657, 2662, 2672,
2705, 2931, 4729, 4767, 4781, 8476
 - \if_charcode:w
. 22, 785, 792, 2677, 4707, 4713, 4760
 - \if_cs_exist:N 22, 799, 799, 1100, 1128, 2866
 - \if_cs_exist:w 799,
800, 1109, 1137, 1297, 4615, 11616,
11706, 11914, 12101, 12110, 12440
 - \if_dim:w
80, 3921, 3921, 3973, 3985, 4010–4016
 - \if_eof:w 143, 8031, 8031, 8490
 - \if_false 22
 - \if_false: 785, 786, 2278,
4809, 4810, 5120, 5123, 5226, 5230
 - \if_hbox:N 131, 6434, 6434, 6438
 - \if_int_compare:w
..... 68, 813, 813, 1481, 1487,
2278, 2280, 2426, 2433, 2464, 2473,
2696, 3213, 3228, 3236, 3247, 3258,
3262, 3263, 3269, 3373, 3381, 3389,
3397, 3405, 3413, 3421, 3429, 4145,
4796, 4835, 8487, 9968, 9979, 10014,
10022, 10030, 10044, 10061, 10083,
10084, 10099, 10107, 10132, 10191,
10233, 10276, 10279, 10297, 10341,
10350, 10352, 10361, 10389, 10408,
10417, 10485, 10488, 10498, 10499,
10517, 10518, 10543–10551, 10582–
10590, 10636, 10645, 10672, 10681,
10711, 10720, 10724, 10733, 10734,
10740, 10780, 10815, 10834, 10860,
10876, 10880, 10882, 10945, 10949,
11043, 11053, 11088, 11092, 11123,
11126, 11134, 11137, 11139, 11154,
11186, 11191, 11202, 11206, 11210,
11211, 11336, 11348, 11376, 11387,
11409, 11452, 11456, 11471, 11476,
11477, 11495, 11499, 11503, 11505,
11530, 11546, 11556, 11566, 11596,
11609, 11636, 11651, 11693, 11722,
11737, 11763, 11764, 11768, 11776,
- I
- \I 2724
 - \if 184, 387

11790, 11791, 11813, 11846, 11847, 11851, 11857, 11867, 11871, 11894, 11907, 11932, 11943, 11944, 11950, 11957, 11958, 11988, 11989, 12018, 12088, 12122, 12124, 12125, 12135, 12150, 12162, 12178, 12179, 12201, 12211, 12229, 12230, 12262, 12263, 12269, 12298, 12301, 12336, 12340, 12344, 12350, 12360, 12364, 12403, 12404, 12454, 12457, 12470, 12487, 12493, 12516, 12530, 12542, 12567, 12570, 12581, 12589, 12649, 12656, 12669, 12679, 12699, 12709, 12715, 12742, 12746, 12763, 12781, 12786, 12789, 12817, 12820, 12824, 12825, 12983–12986, 13009, 13012, 13018, 13027, 13030, 13044, 13048, 13052, 13062, 13066, 13070, 13074, 13079, 13094, 13098, 13102, 13106, 13111	\iffontchar 701
\if_int_odd:w 69, 3213, 3217, 3437, 3445, 11534, 12597, 12600	\ifhbox 394
\if_meaning:w 22, 785, 794, 1080, 1097, 1115, 1125, 1143, 1308, 1419, 1572, 1573, 1872, 1924, 1954, 1964, 1973, 1976, 1986, 1989, 2002, 2011, 2412, 2418, 2444, 2452, 2667, 2712, 2745, 2756, 2767, 2778, 2840, 2940, 3370, 4005, 4488, 4500, 4512, 4523, 4538, 4752, 4875, 5148, 5248, 5362, 6166, 6191, 6229, 6310, 12930, 12938	\ifhmode 400
\if_mode_horizontal 23	\ifinner 403
\if_mode_horizontal: 795, 796, 2272	\ifmmode 401
\if_mode_inner 23	\ifnum 390
\if_mode_inner: 795, 798, 2274	\ifodd 169, 205, 391
\if_mode_math 23	\iftrue 399
\if_mode_math: 795, 795, 2276	\ifvbox 395
\if_mode_vertical 23	\ifvmode 402
\if_mode_vertical: 795, 797, 2270	\ifvoid 396
\if_num:w 68, 2848, 3213, 3216	\ifx 13, 62, 108, 135, 229, 233, 397
\if_predicate:w 41, 1892, 1893, 1938	\ignorespaces 445
\if_true 22	\immediate 407
\if_true: 785, 785	\indent 541
\if_vbox:N 132, 6434, 6435, 6440	\initcatcodetable 760
\ifcase 388	\input 415
\ifcat 389	\input@path 9798, 9801, 9816
\ifcsname 672	\inputlineno 417
\ifdefined 671	\insert 597
\ifdim 392	\insertpenalties 600
\ifeof 393	\int_abs:n 59, 3225, 3225
\iffalse 398	\int_add:cn 3326
	\int_add:Nn 61, 3326, 3326, 3331, 3334, 8396, 8409, 8449
	\int_compare:n 3357, 3357
	\int_compare:nF 3461, 3476
	\int_compare:nNn 3427, 3427
	\int_compare:nNnF 2228, 2237, 3489, 3504, 8210, 8212, 8223, 8225
	\int_compare:nNnT . . 3481, 3498, 4088, 5444, 8139, 8158, 8176, 8195, 13299
	\int_compare:nNnTF 62, 2071, 2123, 2217, 2220, 3297, 3299, 3510, 3596, 3602, 3749, 3773, 3777, 3827, 5457, 5948, 5950, 5955, 5963, 5983, 8093, 8106, 8397, 13269
	\int_compare:nT 3453, 3470
	\int_compare:nTF 62
	\int_compare:<:w 3357
	\int_compare:=:w 3357
	\int_compare:>:w 3357
	\int_compare_aux:Nw 3357, 3361, 3369
	\int_compare_aux:nw 3357, 3358, 3359
	\int_compare_p:nNn 4157, 4158
	\int_const:cn 3295, 3786–3799
	\int_const:Nn 60, 3295, 3295, 3315, 3848–3866, 9873–9877
	\int_convert_from_base_ten:n . 3874, 3874
	\int_convert_to_base_ten:n . 3874, 3876

- `\int_convert_to_symbols:nnn` . 3874, 3875
- `\int_decr:c` 3338
- `\int_decr:N` 61, 3338, 3340, 3345, 3347
- `\int_div_round:nn` 59, 3255, 3280
- `\int_div_truncate:nn`
 - 60, 3255, 3255, 3284, 3525, 3615
- `\int_do_until:nn` ... 63, 3451, 3473, 3477
- `\int_do_until:nNnn` .. 63, 3479, 3501, 3505
- `\int_do_while:nn` 63, 3451, 3467
- `\int_do_while:nNnn`
 - 63, 3471, 3479, 3495, 3499
- `\int_eval:n`
 - . 59, 1350, 2117, 2223, 2232, 2241,
 - 3219, 3220, 3223, 3280, 3507, 3593,
 - 3662, 3672, 3731, 3745, 3749, 3752,
 - 3767, 3776, 4623, 4628, 4941, 5430,
 - 5442, 5920, 5929, 5952, 5965, 5987
- `\int_eval:w` 69, 1324, 2179, 2494, 2496,
 - 2498, 2564, 2566, 2568, 2570, 2572,
 - 2574, 2576, 2578, 2580, 2582, 2584,
 - 2586, 3213, 3214, 3220, 3223, 3228,
 - 3231, 3235, 3237, 3246, 3248, 3257,
 - 3258, 3262, 3263, 3269, 3283, 3307,
 - 3327, 3329, 3351, 3358, 3373, 3381,
 - 3389, 3397, 3405, 3413, 3421, 3429,
 - 3437, 3445, 3623, 3650, 3805, 8469,
 - 9994, 9997, 10015, 10031, 10392,
 - 10500, 10504, 10519, 10523, 10722,
 - 10723, 10816, 10842, 10853, 10857,
 - 10869, 10873, 10886, 10890, 10932,
 - 10946, 10950, 10963, 11016, 11044,
 - 11054, 11075, 11089, 11093, 11106,
 - 11124, 11169, 11178, 11183–11185,
 - 11199–11201, 11211, 11215, 11216,
 - 11328, 11335, 11360, 11370, 11490,
 - 11492, 11494, 11506, 11512, 11516,
 - 11520, 11604, 11659, 11701, 11745,
 - 11902, 12007, 12026, 12096, 12175,
 - 12208, 12271, 12277, 12281, 12318,
 - 12337, 12405, 12435, 12478, 12582,
 - 12700, 12743, 12747, 12764, 12790,
 - 12862, 12912, 13009, 13012, 13027
- `\int_eval_end` 69
- `\int_eval_end:` 1324, 2179, 2494,
 - 2496, 2498, 2564, 2566, 2568, 2570,
 - 2572, 2574, 2576, 2578, 2580, 2582,
 - 2584, 2586, 3213, 3215, 3220, 3223,
 - 3231, 3237, 3242, 3248, 3253, 3278,
 - 3285, 3307, 3327, 3329, 3351, 3373,
 - 3381, 3389, 3397, 3405, 3413, 3421,
 - 3429, 3437, 3445, 3623, 3650, 8472,
 - 10392, 10506, 10525, 11108, 11172,
 - 11178, 11183–11185, 11199–11201,
 - 11215, 11216, 11328, 11335, 11490,
 - 11492, 11494, 11514, 11518, 11522,
 - 12009, 12177, 12210, 12273, 12283
- `\int_from_alph:n` 66, 3729, 3729
- `\int_from_alph_aux:N` ... 3729, 3745, 3748
- `\int_from_alph_aux:n` ... 3729, 3734, 3737
- `\int_from_alph_aux:nN`
 - 3729, 3738, 3739, 3744
- `\int_from_base:nn`
 - 66, 3750, 3750, 3781, 3783, 3785, 3876
- `\int_from_base_aux:N` ... 3750, 3767, 3771
- `\int_from_base_aux:nn` .. 3750, 3755, 3759
- `\int_from_base_aux:nnN`
 - 3750, 3760, 3761, 3766
- `\int_from_binary:n` 66, 3780, 3780
- `\int_from_hexadecimal:n` . 66, 3780, 3782
- `\int_from_octal:n` 66, 3780, 3784
- `\int_from_roman:n` 66, 3800, 3800
- `\int_from_roman_aux:NN`
 - 3800, 3806, 3809, 3834, 3838
- `\int_from_roman_clean_up:w`
 - 3800, 3817, 3824, 3826, 3845
- `\int_from_roman_end:w` .. 3800, 3804, 3843
- `\int_gadd:cn` 3326
- `\int_gadd:Nn`
 - .. 61, 3326, 3330, 3335, 13268, 13298
- `\int_gdecr:c` 3338
- `\int_gdecr:N`
 - . 61, 2267, 3338, 3344, 3349, 4584,
 - 4594, 5284, 5837, 5847, 6245, 9206
- `\int_get_digits:n` 68, 3695, 3700, 3734, 3756
- `\int_get_sign:n` 68, 3695, 3695, 3733, 3754
- `\int_get_sign_and_digits_aux:nNNN` ..
 - 3695, 3697, 3702, 3705, 3728
- `\int_get_sign_and_digits_aux:oNNN` ..
 - 3695, 3711, 3715, 3721
- `\int_gincr:c` 3338
- `\int_gincr:N`
 - . 61, 2265, 3338, 3342, 3348, 4578,
 - 4588, 5280, 5831, 5841, 6240, 9200
- `\int_gset:cn` 3350
- `\int_gset:Nn` 61, 3302, 3312, 3350, 3352, 3354
- `\int_gset_eq:cc` 3320
- `\int_gset_eq:cN` 3320
- `\int_gset_eq:Nc` 3320
- `\int_gset_eq:NN` 60, 3320, 3323–3325
- `\int_gsub:cn` 3326

- \int_gsub:Nn . . . [61](#), [3326](#), [3332](#), [3337](#), [13306](#)
- \int_gzero:c [3316](#)
- \int_gzero:N [60](#), [3316](#), [3317](#), [3319](#)
- \int_if_even:n [3435](#), [3443](#)
- \int_if_even:nTF [62](#)
- \int_if_odd:n [3435](#), [3435](#)
- \int_if_odd:nTF [62](#)
- \int_incr:c [3338](#)
- \int_incr:N [61](#), [3338](#), [3338](#), [3343](#), [3346](#),
[8157](#), [8194](#), [8415](#), [9343](#), [9370](#), [9437](#)
- \int_max:nn [60](#), [3225](#), [3233](#)
- \int_min:nn [60](#), [3225](#), [3244](#)
- \int_mod:nn [60](#), [3255](#), [3281](#), [3515](#), [3606](#)
- \int_new:c [3287](#)
- \int_new:N [60](#), [2244](#), [3287](#), [3288](#), [3294](#),
[3301](#), [3311](#), [3867](#)–[3873](#), [5828](#), [6237](#),
[8064](#), [8302](#), [8304](#)–[8307](#), [9107](#), [9219](#),
[9878](#), [9880](#), [9882](#), [9884](#), [9886](#), [9888](#),
[9896](#)–[9924](#), [9926](#)–[9930](#), [9933](#), [9934](#),
[9936](#), [9937](#), [9939](#)–[9942](#), [13260](#), [13262](#)
- \int_set:cn [3350](#)
- \int_set:Nn [61](#), [3350](#), [3350](#),
[3352](#), [3353](#), [8091](#), [8104](#), [8120](#), [8128](#),
[8142](#), [8179](#), [8303](#), [8349](#), [8362](#), [8394](#),
[8422](#), [8689](#), [9339](#), [9365](#), [9433](#), [9879](#),
[9881](#), [9883](#), [9885](#), [9887](#), [9889](#), [10635](#),
[10670](#), [13128](#), [13130](#), [13132](#), [13134](#),
[13136](#), [13138](#), [13140](#), [13142](#), [13261](#)
- \int_set_eq:cc [3320](#)
- \int_set_eq:cN [3320](#)
- \int_set_eq:Nc [3320](#)
- \int_set_eq:NN
. [60](#), [3320](#), [3320](#)–[3322](#), [8320](#), [8356](#)
- \int_show:c [3846](#), [3847](#)
- \int_show:N [66](#), [1443](#), [3846](#), [3846](#)
- \int_sub:cn [3326](#)
- \int_sub:Nn [61](#), [3326](#), [3328](#), [3333](#), [3336](#), [8455](#)
- \int_to_Alph:n [64](#), [3528](#), [3560](#)
- \int_to_alph:n [64](#), [3528](#), [3528](#)
- \int_to_arabic:n [64](#), [3507](#), [3507](#)
- \int_to_base:nn
[65](#), [3592](#), [3592](#), [3654](#), [3656](#), [3658](#), [3874](#)
- \int_to_base_aux_i:nn . . . [3592](#), [3593](#), [3594](#)
- \int_to_base_aux_ii:nnN
. [3592](#), [3597](#), [3598](#), [3600](#), [3614](#)
- \int_to_base_aux_iii:nnnN [3592](#), [3605](#), [3612](#)
- \int_to_binary:n [65](#), [3653](#), [3653](#)
- \int_to_hexadecimal:n . . . [65](#), [3653](#), [3655](#)
- \int_to_letter:n [68](#), [3592](#), [3603](#), [3606](#), [3620](#)
- \int_to_octal:n [65](#), [3653](#), [3657](#)
- \int_to_Roman:n [66](#), [3659](#), [3669](#)
- \int_to_roman:n [66](#), [3659](#), [3659](#)
- \int_to_roman:w [68](#), [813](#), [814](#),
[891](#), [893](#), [1060](#), [1066](#), [1076](#), [2026](#),
[2031](#), [2177](#), [3213](#), [3362](#), [3662](#), [3672](#)
- \int_to_Roman_aux:N [3671](#), [3674](#), [3677](#)
- \int_to_roman_aux:N [3659](#), [3661](#), [3664](#), [3667](#)
- \int_to_Roman_c:w [3659](#), [3691](#)
- \int_to_roman_c:w [3659](#), [3683](#)
- \int_to_Roman_d:w [3659](#), [3692](#)
- \int_to_roman_d:w [3659](#), [3684](#)
- \int_to_Roman_i:w [3659](#), [3687](#)
- \int_to_roman_i:w [3659](#), [3679](#)
- \int_to_Roman_l:w [3659](#), [3690](#)
- \int_to_roman_l:w [3659](#), [3682](#)
- \int_to_Roman_m:w [3659](#), [3693](#)
- \int_to_roman_m:w [3659](#), [3685](#)
- \int_to_Roman_Q:w [3659](#), [3694](#)
- \int_to_roman_Q:w [3659](#), [3686](#)
- \int_to_Roman_v:w [3659](#), [3688](#)
- \int_to_roman_v:w [3659](#), [3680](#)
- \int_to_Roman_x:w [3659](#), [3689](#)
- \int_to_roman_x:w [3659](#), [3681](#)
- \int_to_symbol:n [3877](#), [3877](#)
- \int_to_symbol_math:n . . . [3877](#), [3881](#), [3884](#)
- \int_to_symbol_text:n . . . [3877](#), [3882](#), [3899](#)
- \int_to_symbols:nnn . . . [65](#), [3508](#), [3508](#),
[3524](#), [3530](#), [3562](#), [3875](#), [3886](#), [3901](#)
- \int_to_symbols_aux:nnnn . . . [3512](#), [3522](#)
- \int_until_do:nn . . . [64](#), [3451](#), [3459](#), [3464](#)
- \int_until_do:nNnn . . . [63](#), [3479](#), [3487](#), [3492](#)
- \int_use:c [3355](#), [3356](#)
- \int_use:N [62](#), [2249](#),
[2255](#), [3355](#), [3355](#), [3356](#), [4579](#), [4582](#),
[4589](#), [4592](#), [5278](#), [5286](#), [5832](#), [5836](#),
[5842](#), [5846](#), [6241](#), [6244](#), [8134](#), [8135](#),
[8144](#), [8149](#), [8151](#), [8160](#), [8171](#), [8172](#),
[8181](#), [8186](#), [8188](#), [8197](#), [8590](#), [9154](#),
[9176](#), [9202](#), [9204](#), [9340](#), [9366](#), [9434](#),
[10007](#), [10047](#), [10064](#), [10077](#), [10111](#),
[10125](#), [10136](#), [10150](#), [10196](#), [10199](#),
[10201](#), [10238](#), [10241](#), [10243](#), [10650](#),
[10653](#), [10655](#), [10686](#), [10689](#), [10691](#),
[10703](#), [10730](#), [10759](#), [10762](#), [10764](#),
[10785](#), [10788](#), [10790](#), [10839](#), [10845](#),
[10960](#), [10966](#), [11010](#), [11022](#), [11103](#),
[11110](#), [11122](#), [11352](#), [11364](#), [11380](#),
[11392](#), [11402](#), [11413](#), [11426](#), [11445](#),
[11601](#), [11607](#), [11656](#), [11662](#), [11698](#),
[11704](#), [11742](#), [11748](#), [11899](#), [11905](#),

- 12023, 12029, 12093, 12099, 12172,
- 12205, 12231, 12234, 12315, 12321,
- 12432, 12438, 12475, 12482, 12495,
- 12517, 12534, 12547, 12557, 12568,
- 12641, 12643, 12645, 12673, 12684,
- 12859, 12865, 12892, 12894, 12896,
- 12898, 12900, 13129, 13131, 13133,
- 13135, 13137, 13139, 13141, 13143
- \int_value:w 69, 1997, 1998,
- 2018, 2020–2022, 2179, 3213, 3213,
- 3220, 3223, 3227, 3235, 3246, 3257,
- 3283, 3358, 3650, 3805, 3982, 6964,
- 6988–6990, 6992, 7109, 7118, 7120,
- 7125–7128, 7132–7135, 7186, 7192,
- 7194, 7196, 7198, 7203, 7208, 7213,
- 7220, 7227, 7366, 7396, 7397, 7436,
- 7455, 7461, 7524, 7526, 7546, 7548,
- 7573, 7611, 7631, 7639, 7665, 7667,
- 7671, 7673, 7706, 7720, 7727, 7854,
- 7954, 7968, 8469, 10392, 10504,
- 10523, 10842, 10963, 11106, 11604,
- 11659, 11701, 11745, 11902, 12026,
- 12096, 12318, 12435, 12478, 12862
- \int_while_do:nn ... 64, 3451, 3451, 3456
- \int_while_do:nNnn .. 63, 3479, 3479, 3484
- \int_zero:c 3316
- \int_zero:N 60, 3316, 3316, 3318,
- 8350, 8352, 8443, 9333, 9352, 9427
- \interactionmode 699
- \interlinepenalties 720
- \interlinepenalty 579
- \io_new:c 138
- \ior_alloc_read:n 8092, 8116, 8124
- \ior_close:N 139, 8090, 8206, 8232, 9790, 9819
- \ior_gto:NN 142, 8499, 8501
- \ior_if_eof:N 8483
- \ior_if_eof:Nf 9809
- \ior_if_eof:Ntf 143, 9787
- \ior_if_eof_p:N 8483
- \ior_list_streams 139
- \ior_list_streams: 8234, 8234, 8514
- \ior_new:c 8080
- \ior_new:N ... 138, 8080, 8080, 8085, 8086
- \ior_open:cn 8088
- \ior_open:Nn 138, 8088, 8088, 8114, 9786, 9808
- \ior_open_streams: 8513, 8514
- \ior_raw_new:c 8066, 8186
- \ior_raw_new:N 143, 8066, 8068, 8076, 8078, 8178
- \ior_show_aux:nn . 8234, 8244, 8249, 8269
- \ior_str_gto:NN 142, 8503, 8505
- \ior_str_to:NN 142, 8503, 8503
- \ior_stream_alloc:N 8096, 8132, 8169
- \ior_stream_alloc_aux: 8132, 8175, 8192, 8200, 8202
- \ior_to:NN 142, 8499, 8499
- \iow_alloc_write:n 8105, 8116, 8116
- \iow_char:N 140, 5356, 5910, 6273, 6275, 8301, 8301
- \iow_close:c 8206
- \iow_close:N .. 139, 8103, 8206, 8219, 8233
- \iow_indent:n 141, 8339, 8339, 8365
- \iow_indent_expandable:n 8339, 8340, 8365
- \iow_list_streams 139
- \iow_list_streams: 8234, 8254, 8515
- \iow_log:n ... 139, 8292, 8293, 9855–9857
- \iow_log:x 1180, 1180, 1219,
- 1825, 8292, 8292, 8677, 8679, 8680
- \iow_new:c 8080
- \iow_new:N 138, 8080, 8085, 8087
- \iow_newline 140
- \iow_newline: 5355, 5909, 6272, 7958–7962, 7981,
- 8251, 8300, 8300, 8373, 8664, 8666
- \iow_now:Nn 139, 8290,
- 8290, 8293, 8295, 8297, 8509, 8511
- \iow_now:Nx 8289, 8289, 8291, 8292, 8294, 8299
- \iow_now_buffer_safe:Nn 8507, 8508
- \iow_now_buffer_safe:Nx 8507, 8510
- \iow_now_when_avail:Nn . 139, 8296, 8296
- \iow_now_when_avail:Nx 8296, 8298
- \iow_open:cn 8088
- \iow_open:Nn 138, 8088, 8101, 8115
- \iow_open_streams: 8513, 8515
- \iow_raw_new:c 8066, 8149
- \iow_raw_new:N 143, 8066, 8071, 8075, 8079, 8141
- \iow_shipout:Nn ... 139, 8286, 8286, 8288
- \iow_shipout:Nx 8286
- \iow_shipout_x:Nn 140, 8284, 8284, 8285, 8287, 8289
- \iow_shipout_x:Nx 8284
- \iow_show_aux:nn 8234, 8264, 8269
- \iow_stream_alloc:N 8109, 8132, 8132
- \iow_stream_alloc_aux: 8132, 8138, 8155, 8163, 8165

- \iow_term:n [139](#), [8292](#), [8295](#)
- \iow_term:x [1180](#), [1182](#), [5338](#),
[5342](#), [5892](#), [5896](#), [6255](#), [6259](#), [7956](#),
[7964](#), [7975](#), [8238](#), [8242](#), [8258](#), [8262](#),
[8292](#), [8294](#), [8662](#), [8684](#), [8686](#), [8687](#)
- \iow_wrap:xnnnN
..... [141](#), [8346](#), [8346](#), [8509](#), [8511](#),
[8621](#), [8624](#), [8629](#), [8632](#), [8678](#), [8685](#)
- \iow_wrap_end: [8459](#)
- \iow_wrap_end:w [8430](#)
- \iow_wrap_indent: [8447](#)
- \iow_wrap_indent:w [8430](#)
- \iow_wrap_loop:w
..... [8376](#), [8385](#), [8385](#), [8400](#), [8437](#)
- \iow_wrap_new_marker:n
..... [8319](#), [8323](#), [8334](#)–[8337](#)
- \iow_wrap_newline: [8439](#)
- \iow_wrap_newline:w [8430](#)
- \iow_wrap_special:w [8389](#), [8430](#), [8430](#), [8436](#)
- \iow_wrap_unindent: [8453](#)
- \iow_wrap_unindent:w [8430](#)
- \iow_wrap_word: [8390](#), [8392](#), [8392](#)
- \iow_wrap_word_fits: ... [8392](#), [8398](#), [8402](#)
- \iow_wrap_word_newline: [8392](#), [8399](#), [8418](#)

- J**
- \jobname [654](#)

- K**
- \K [2724](#)
- \kern [532](#)
- \kernel_register_show:c [1434](#), [1443](#), [3847](#)
- \kernel_register_show:N
... [1434](#), [1434](#), [3846](#), [4094](#), [4175](#), [4222](#)
- \keys_bool_set:NN [9290](#), [9290](#), [9457](#), [9459](#)
- \keys_bool_set_inverse:NN
..... [9305](#), [9305](#), [9461](#), [9463](#)
- \keys_choice_code_store:x
..... [9372](#), [9372](#), [9473](#), [9475](#)
- \keys_choice_find:n [9323](#), [9413](#), [9662](#), [9662](#)
- \keys_choice_make: [9293](#),
[9308](#), [9320](#), [9320](#), [9332](#), [9351](#), [9465](#)
- \keys_choices_generate:n [9346](#), [9346](#), [9505](#)
- \keys_choices_generate_aux:n
..... [9346](#), [9353](#), [9360](#)
- \keys_choices_make:nn ... [9330](#), [9330](#), [9467](#)
- \keys_cmd_set:nn [9298](#), [9313](#), [9322](#), [9324](#),
[9383](#), [9383](#), [9404](#), [9416](#), [9418](#), [9469](#)
- \keys_cmd_set:nx
..... [9294](#), [9296](#), [9309](#), [9311](#), [9336](#), [9362](#),
[9383](#), [9388](#), [9409](#), [9430](#), [9449](#), [9471](#)
- \keys_cmd_set_aux:n [9383](#), [9385](#), [9390](#), [9393](#)
- \keys_default_set:n
... [9303](#), [9318](#), [9399](#), [9399](#), [9401](#), [9485](#)
- \keys_default_set:V [9399](#), [9487](#)
- \keys_define:nn ... [153](#), [9228](#), [9228](#), [9701](#)
- \keys_define_aux:nnn ... [9228](#), [9230](#), [9236](#)
- \keys_define_aux:onnn [9228](#), [9229](#)
- \keys_define_elt:n [9233](#), [9237](#), [9237](#)
- \keys_define_elt:nn [9233](#), [9237](#), [9242](#)
- \keys_define_elt_aux:nn
..... [9237](#), [9240](#), [9245](#), [9247](#)
- \keys_define_key:n [9250](#), [9273](#), [9273](#)
- \keys_define_key_aux:w . [9273](#), [9277](#), [9288](#)
- \keys_execute: [9608](#), [9633](#), [9633](#)
- \keys_execute:nn
... [9633](#), [9634](#), [9637](#), [9653](#), [9664](#), [9665](#)
- \keys_execute_unknown:
... [9566](#), [9568](#), [9633](#), [9634](#), [9635](#), [9643](#)
- \keys_execute_unknown_alt:
..... [9566](#), [9633](#), [9644](#)
- \keys_execute_unknown_std:
..... [9568](#), [9633](#), [9643](#)
- \keys_if_choice_exist:nnn ... [9673](#), [9673](#)
- \keys_if_choice_exist:nnTF [161](#)
- \keys_if_exist:nn [9667](#), [9667](#)
- \keys_if_exist:nnTF [161](#)
- \keys_if_value:n [9626](#)
- \keys_if_value_p:n [9591](#), [9601](#), [9626](#)
- \keys_meta_make:n [9402](#), [9402](#), [9515](#)
- \keys_meta_make:x [9402](#), [9407](#), [9517](#)
- \keys_multichoice_find:n [9412](#), [9412](#), [9417](#)
- \keys_multichoice_make:
..... [9412](#), [9414](#), [9426](#), [9519](#)
- \keys_multichoices_make:nn
..... [9412](#), [9424](#), [9521](#)
- \keys_property_find:n . [9248](#), [9256](#), [9256](#)
- \keys_property_find_aux:w
..... [9256](#), [9260](#), [9263](#), [9269](#)
- \keys_set:nn
... [160](#), [9405](#), [9410](#), [9550](#), [9550](#), [9558](#)
- \keys_set:no [9550](#)
- \keys_set:nV [9550](#)
- \keys_set:nv [9550](#)
- \keys_set_aux:nnn [9550](#), [9552](#), [9559](#)
- \keys_set_aux:onnn [9550](#), [9551](#)
- \keys_set_elt:n . [9555](#), [9567](#), [9574](#), [9574](#)
- \keys_set_elt:nn . [9555](#), [9567](#), [9574](#), [9579](#)

- \keys_set_elt_aux:nn [9574](#), [9577](#), [9582](#), [9584](#)
 - \keys_set_known:nnN [161](#), [9560](#), [9560](#), [9572](#)
 - \keys_set_known:noN [9560](#)
 - \keys_set_known:nVN [9560](#)
 - \keys_set_known:nvN [9560](#)
 - \keys_set_known_aux:nnnN [9560](#), [9562](#), [9573](#)
 - \keys_set_known_aux:onnN [9560](#), [9561](#)
 - \keys_show:nn [161](#), [9679](#), [9679](#)
 - \keys_value_or_default:n [9588](#), [9611](#), [9611](#)
 - \keys_value_requirement:n
..... [9440](#), [9440](#), [9547](#), [9549](#)
 - \keys_variable_set:cnN .. [9446](#), [9479](#),
[9491](#), [9499](#), [9509](#), [9525](#), [9533](#), [9537](#)
 - \keys_variable_set:cnNN . [9446](#), [9483](#),
[9495](#), [9503](#), [9513](#), [9529](#), [9541](#), [9545](#)
 - \keys_variable_set:NnN
..... [9446](#), [9452](#), [9455](#), [9477](#),
[9489](#), [9497](#), [9507](#), [9523](#), [9531](#), [9535](#)
 - \keys_variable_set:NnNN
... [9446](#), [9446](#), [9453](#), [9454](#), [9481](#),
[9493](#), [9501](#), [9511](#), [9527](#), [9539](#), [9543](#)
 - \keyval_parse:n [9112](#), [9120](#), [9205](#)
 - \keyval_parse:NnN [162](#), [9198](#),
[9198](#), [9233](#), [9555](#), [9567](#), [9745](#)–[9747](#)
 - \keyval_parse_elt:w
..... [9128](#), [9134](#), [9134](#), [9137](#), [9142](#)
 - \keyval_split_key:w [9148](#), [9166](#), [9166](#)
 - \keyval_split_key_value:w [9141](#), [9146](#), [9146](#)
 - \keyval_split_key_value_aux:wTF
..... [9146](#), [9159](#), [9164](#)
 - \keyval_split_value:w .. [9160](#), [9171](#), [9171](#)
 - \keyval_split_value_aux:w ... [9189](#), [9192](#)
 - \KV_process_no_space_removal_no_sanitiz:NnN
..... [9744](#), [9747](#)
 - \KV_process_space_removal_no_sanitiz:NnN
..... [9744](#), [9746](#)
 - \KV_process_space_removal_sanitiz:NnN
..... [9744](#), [9745](#)
- L**
- \L [2724](#)
 - \l@expl@log@functions@bool [1211](#)
 - \l_box_angle_fp .. [6553](#), [6553](#), [6575](#), [6623](#)
 - \l_box_bottom_dim .. [6556](#), [6557](#), [6590](#),
[6655](#), [6659](#), [6664](#), [6670](#), [6675](#), [6679](#),
[6688](#), [6690](#), [6703](#), [6711](#), [6734](#), [6740](#),
[6750](#), [6755](#), [6771](#), [6793](#), [6812](#), [6815](#)
 - \l_box_bottom_new_dim
[6560](#), [6561](#), [6616](#), [6656](#), [6667](#), [6678](#),
[6689](#), [6733](#), [6739](#), [6812](#), [6816](#), [6832](#)
 - \l_box_cos_fp [6554](#), [6554](#),
[6579](#), [6595](#), [6600](#), [6627](#), [6635](#), [6646](#)
 - \l_box_left_dim [6556](#), [6558](#), [6592](#),
[6655](#), [6657](#), [6666](#), [6670](#), [6675](#), [6681](#),
[6686](#), [6690](#), [6705](#), [6752](#), [6773](#), [6795](#)
 - \l_box_left_new_dim [6560](#), [6562](#),
[6607](#), [6618](#), [6658](#), [6669](#), [6680](#), [6691](#)
 - \l_box_right_dim [6556](#),
[6559](#), [6591](#), [6653](#), [6659](#), [6664](#), [6668](#),
[6677](#), [6679](#), [6688](#), [6692](#), [6704](#), [6707](#),
[6751](#), [6772](#), [6775](#), [6794](#), [6819](#), [6820](#)
 - \l_box_right_new_dim [6560](#), [6563](#),
[6618](#), [6660](#), [6671](#), [6682](#), [6693](#), [6727](#),
[6728](#), [6819](#), [6820](#), [6835](#), [6837](#), [6843](#)
 - \l_box_scale_x_fp [6695](#),
[6695](#), [6706](#), [6708](#), [6713](#), [6757](#), [6774](#),
[6776](#)–[6778](#), [6790](#), [6796](#), [6818](#), [6833](#)
 - \l_box_scale_y_fp
..... [6695](#), [6696](#), [6709](#), [6712](#),
[6715](#), [6732](#), [6734](#), [6738](#), [6740](#), [6753](#),
[6756](#)–[6758](#), [6777](#), [6791](#), [6798](#), [6809](#)
 - \l_box_sin_fp [6554](#),
[6555](#), [6577](#), [6593](#), [6626](#), [6636](#), [6647](#)
 - \l_box_tmp_box [6564](#), [6564](#),
[6604](#), [6605](#), [6611](#), [6615](#)–[6617](#), [6619](#),
[6825](#), [6831](#), [6832](#), [6838](#), [6843](#), [6844](#)
 - \l_box_tmp_fp
[6564](#), [6565](#), [6623](#)–[6627](#), [6634](#), [6636](#),
[6637](#), [6645](#), [6647](#), [6648](#), [6707](#), [6708](#),
[6710](#), [6712](#), [6754](#), [6756](#), [6775](#), [6776](#)
 - \l_box_top_dim [6556](#), [6556](#), [6589](#),
[6653](#), [6657](#), [6666](#), [6668](#), [6677](#), [6681](#),
[6686](#), [6692](#), [6702](#), [6711](#), [6732](#), [6738](#),
[6749](#), [6755](#), [6770](#), [6792](#), [6811](#), [6816](#)
 - \l_box_top_new_dim
[6560](#), [6560](#), [6615](#), [6654](#), [6665](#), [6676](#),
[6687](#), [6731](#), [6737](#), [6811](#), [6815](#), [6831](#)
 - \l_box_x_fp
.. [6566](#), [6566](#), [6631](#), [6633](#), [6642](#), [6645](#)
 - \l_box_x_new_fp
.. [6566](#), [6568](#), [6633](#), [6635](#), [6637](#), [6638](#)
 - \l_box_y_fp
.. [6566](#), [6567](#), [6632](#), [6634](#), [6643](#), [6644](#)
 - \l_box_y_new_fp
.. [6566](#), [6569](#), [6644](#), [6646](#), [6648](#), [6649](#)
 - \l_cctab_tmp_tl [13294](#), [13307](#)–[13310](#), [13324](#)
 - \l_clist_if_in_clist [5781](#), [5781](#), [5794](#), [5795](#)
 - \l_clist_remove_clist
.. [5733](#), [5733](#), [5740](#), [5743](#), [5744](#), [5746](#)
 - \l_clist_show_tl [5901](#), [5904](#)

- \l_clist_tmpa_tl 5583, 5583,
5672, 5674, 5676, 5677, 5869, 5870
- \l_coffin_aligned_coffin
..... 7096, 7098, 7353,
7354, 7358, 7364, 7366, 7367, 7383,
7384, 7390–7394, 7396, 7398, 7402,
7403, 7408–7412, 7446, 7461, 7506,
7508, 7937, 7944, 7946, 7948, 7950
- \l_coffin_aligned_internal_coffin ..
..... 7096, 7099, 7425, 7432
- \l_coffin_bottom_corner_dim
..... 7513, 7515,
7538, 7542, 7609, 7618, 7634, 7642
- \l_coffin_bounding_prop
..... 7511, 7511, 7529,
7554, 7556, 7559, 7561, 7567, 7624
- \l_coffin_bounding_shift_dim
..... 7512, 7512, 7536, 7623, 7628
- \l_coffin_calc_a_fp
..... 6932, 6932, 7297, 7301, 7308, 7310–
7312, 7315–7317, 7320, 7340, 7344
- \l_coffin_calc_b_fp
..... 6932, 6933, 7298, 7301,
7304, 7313, 7321, 7324, 7341, 7347
- \l_coffin_calc_c_fp
.. 6932, 6934, 7299, 7302, 7342, 7346
- \l_coffin_calc_d_fp . 6932, 6935, 7300,
7302, 7304, 7318, 7322, 7343, 7345
- \l_coffin_calc_result_fp
... 6932, 6936, 7307, 7309, 7314,
7319, 7323, 7326, 7339, 7344–7348
- \l_coffin_cos_fp
..... 6942, 6943, 7523, 7594, 7599
- \l_coffin_Depth_dim 6952, 6952, 7144
- \l_coffin_display_coffin
..... 7733, 7733, 7862, 7869,
7939, 7940, 7945, 7947, 7949, 7950
- \l_coffin_display_coord_coffin
..... 7733, 7734,
7800, 7820, 7836, 7884, 7904, 7923
- \l_coffin_display_font_tl
.. 7778, 7778, 7780, 7783, 7808, 7892
- \l_coffin_display_handles_prop
7736, 7736, 7737, 7739, 7741, 7743,
7745, 7747, 7749, 7751, 7753, 7755,
7757, 7759, 7761, 7763, 7765, 7767,
7769, 7771, 7811, 7815, 7895, 7899
- \l_coffin_display_offset_dim . 7773,
7773, 7774, 7837, 7838, 7924, 7925
- \l_coffin_display_pole_coffin
.. 7733, 7735, 7788, 7799, 7843, 7882
- \l_coffin_display_poles_prop
..... 7777, 7777,
7853, 7858, 7861, 7864, 7866, 7873
- \l_coffin_display_x_dim
..... 7775, 7775, 7879, 7934
- \l_coffin_display_y_dim
..... 7775, 7776, 7880, 7936
- \l_coffin_error_bool 6937, 6937, 7239,
7243, 7257, 7272, 7305, 7875, 7877
- \l_coffin_handles_tmp_prop
..... 7785, 7785, 7863
- \l_coffin_Height_dim ... 6952, 6953, 7143
- \l_coffin_left_corner_dim 7513, 7513,
7537, 7545, 7610, 7616, 7633, 7641
- \l_coffin_offset_x_dim
..... 6938, 6938, 7356,
7357, 7360, 7368, 7370, 7372, 7378,
7381, 7401, 7421, 7429, 7933, 7941
- \l_coffin_offset_y_dim
... 6938, 6939, 7371, 7373, 7378,
7381, 7401, 7423, 7430, 7935, 7942
- \l_coffin_pole_a_tl
... 6940, 6940, 7237, 7242, 7470,
7473, 7474, 7477, 7855, 7857, 7860
- \l_coffin_pole_b_tl
6940, 6941, 7238, 7242, 7471, 7473,
7475, 7477, 7856, 7857, 7859, 7860
- \l_coffin_right_corner_dim
..... 7513, 7514, 7545, 7608, 7617
- \l_coffin_scale_x_fp 7646, 7646,
7654, 7656, 7669, 7682, 7687, 7697
- \l_coffin_scale_y_fp 7646,
7647, 7657, 7659, 7683, 7684, 7700
- \l_coffin_scaled_total_height_dim ..
..... 7648, 7648, 7685, 7686, 7691
- \l_coffin_scaled_width_dim
..... 7648, 7649, 7688, 7689, 7691
- \l_coffin_sin_fp
..... 6942, 6942, 7522, 7595, 7600
- \l_coffin_tmp_box 6911, 6911,
7028, 7032, 7036, 7071, 7076, 7081
- \l_coffin_tmp_dim 6911, 6912,
7359, 7361, 7362, 7558, 7560, 7562
- \l_coffin_tmp_fp 6911, 6913, 7519–7523,
7593, 7595, 7596, 7598, 7600, 7601,
7655, 7656, 7658, 7659, 7696–7701
- \l_coffin_tmp_tl .. 6911, 6914, 6921–
6931, 7444, 7445, 7447, 7812, 7813,

- 7816, 7817, 7825, 7830, 7896, 7897,
7900, 7901, 7910, 7915, 7965, 7972
- \l_coffin_top_corner_dim
..... 7513, 7516, 7542, 7607, 7619
- \l_coffin_TotalHeight_dim 6952, 6954, 7145
- \l_coffin_Width_dim 6952, 6955, 7146
- \l_coffin_x_dim 6944, 6944, 7246,
7255, 7275, 7278, 7285, 7292, 7294,
7325, 7328, 7418, 7422, 7441, 7449,
7566, 7568, 7572, 7574, 7578, 7583,
7705, 7707, 7711, 7714, 7879, 7931
- \l_coffin_x_fp 6948, 6948, 7590, 7592, 7598
- \l_coffin_x_prime_dim ... 6944, 6946,
7418, 7422, 7580, 7584, 7931, 7934
- \l_coffin_x_prime_fp
.. 6948, 6950, 7592, 7594, 7596, 7602
- \l_coffin_y_dim
6944, 6945, 7247, 7260, 7263, 7270,
7287, 7329, 7419, 7424, 7442, 7449,
7566, 7568, 7572, 7574, 7578, 7583,
7705, 7707, 7711, 7714, 7880, 7932
- \l_coffin_y_fp 6948, 6949, 7591, 7593, 7597
- \l_coffin_y_prime_dim ... 6944, 6947,
7419, 7424, 7580, 7585, 7932, 7936
- \l_coffin_y_prime_fp
.. 6948, 6951, 7597, 7599, 7601, 7603
- \l_doc_pTF_name_tl 21, 22, 36,
37, 40, 41, 44, 51, 52, 53, 54, 62,
72, 73, 77, 86, 87, 88, 92, 93, 99,
109, 110, 118, 126, 127, 143, 161, 168
- \l_exp_tl 30, 1536, 1536, 1555, 1556
- \l_expl_status_bool
..... 96, 294, 309, 323, 327, 328
- \l_expl_status_stack_tl 197
- \l_file_name_tl 164, 9776,
9776, 9823, 9824, 9830, 9831, 9841
- \l_file_search_path_saved_seq
..... 164, 9778, 9779, 9800, 9817
- \l_file_search_path_seq
.... 164, 9777, 9777, 9800, 9802,
9803, 9806, 9817, 9847, 9848, 9851
- \l_file_tmpa_seq
164, 9781, 9782, 9801, 9803, 9862, 9863
- \l_fp_arg_tl 9895, 9895, 11594,
11613, 11617, 11627, 11648, 11649,
11691, 11706, 11714, 11734, 11735,
11892, 11911, 11915, 11925, 11935,
11959, 11990, 12015, 12016, 12086,
12101, 12110, 12112, 12140, 12180,
12312, 12429, 12440, 12448, 12468
- \l_fp_count_int 9896, 9896,
11116, 11151, 11163, 11171, 11789,
11821, 11838, 11840, 11843, 11845,
12289, 12326, 12334, 12564, 12567,
12568, 12590, 12634, 12694, 12705
- \l_fp_div_offset_int 9897, 9897,
11072, 11126, 11171, 11173, 12004
- \l_fp_exp_decimal_int ... 9898, 9899,
12164, 12198, 12217, 12237, 12256,
12265, 12270, 12276, 12292, 12341,
12346, 12350, 12353, 12357, 12361,
12364, 12366, 12510, 12520, 12531,
12534, 12543, 12551, 12577, 12640,
12648, 12652, 12663, 12706, 12707
- \l_fp_exp_exponent_int
..... 9898, 9901, 12166,
12200, 12222, 12242, 12258, 12508,
12512, 12530, 12537, 12560, 12644
- \l_fp_exp_extended_int 9898,
9900, 12165, 12199, 12217, 12237,
12257, 12266, 12269, 12274, 12280,
12292, 12342, 12344, 12347, 12358,
12360, 12362, 12511, 12520, 12553,
12557, 12559, 12577, 12642, 12649,
12651, 12654, 12663, 12706, 12708
- \l_fp_exp_integer_int
..... 9898, 9898, 12163, 12197,
12217, 12237, 12255, 12264, 12268,
12292, 12352, 12365, 12509, 12520,
12542, 12547, 12550, 12577, 12639
- \l_fp_input_a_decimal_int
.. 9902, 9904, 9953, 10144, 10145,
10150, 10152, 10183, 10185, 10199,
10225, 10227, 10241, 10639, 10653,
10675, 10689, 10701, 10703, 10710,
10714, 10752, 10762, 10777, 10788,
10858, 10874, 10973, 11055, 11120,
11122, 11124, 11140, 11153, 11154,
11156, 11179, 11350, 11352, 11361,
11369, 11370, 11377, 11380, 11388,
11396, 11477, 11491, 11492, 11496,
11499, 11501, 11507, 11515, 11517,
11530, 11531, 11538, 11540, 11548,
11558, 11561, 11567, 11569, 11573,
11592, 11605, 11689, 11702, 11776,
11782, 11783, 11813, 11816, 11819,
11830, 11834, 11890, 11903, 11957,
11981, 11986, 11988, 12083, 12097,
12262, 12265, 12272, 12278, 12287,
12329, 12401, 12406, 12436, 12583,

12597, 12601, 12604, 12619, 12624,
 12628, 12631, 12632, 12696, 12698,
 12701, 12704, 12734, 12740, 12748,
 12765, 12791, 12824, 12874, 12885,
 12905, 12918, 12951, 12967, 12985,
 13010, 13080, 13112, 13132, 13141
 \l_fp_input_a_exponent_int
 9902, 9905, 9954,
 10099, 10107, 10132, 10153, 10184,
 10201, 10226, 10243, 10655, 10671,
 10691, 10715, 10764, 10790, 10823,
 10933, 11076, 11348, 11372, 11376,
 11405, 11452, 11593, 11607, 11609,
 11690, 11704, 11891, 11905, 11907,
 11932, 11982, 11987, 12008, 12084,
 12099, 12122, 12402, 12438, 12487,
 12488, 12493, 12495, 12506, 12516,
 12517, 12617, 12626, 12735, 12741,
 12786, 12820, 12875, 12886, 12913,
 12920, 12952, 12968, 12987, 13062,
 13066, 13094, 13098, 13134, 13143
 \l_fp_input_a_extended_int
 9910, 9910, 11362,
 11364, 11371, 11398, 11402, 11404,
 11453, 11493–11495, 11497, 11507,
 11519, 11521, 11532, 11539, 11541,
 11549, 11559, 11562, 11570, 11574,
 11782, 11783, 11817, 11820, 11830,
 11834, 11866, 12085, 12266, 12282,
 12288, 12329, 12359, 12565, 12598,
 12605, 12625, 12629, 12631, 12632,
 12696, 12698, 12701, 12704, 12785,
 12791, 12822, 12903, 12906, 12919
 \l_fp_input_a_integer_int
 .. 9902, 9903, 9952, 10133, 10136,
 10143, 10182, 10196, 10224, 10238,
 10650, 10686, 10709, 10711, 10713,
 10759, 10785, 10854, 10870, 10983,
 10988, 10992, 10997, 11055, 11119,
 11124, 11134, 11137, 11152, 11155,
 11177, 11178, 11349, 11359, 11360,
 11387, 11392, 11395, 11456, 11458,
 11471, 11476, 11489, 11490, 11500,
 11503, 11509, 11511, 11513, 11538,
 11540, 11556, 11558, 11561, 11569,
 11573, 11591, 11601, 11688, 11698,
 11889, 11899, 11958, 11980, 11985,
 11989, 12082, 12093, 12125, 12135,
 12150, 12162, 12172, 12174, 12176,
 12201, 12205, 12207, 12209, 12229,
 12231, 12234, 12400, 12406, 12432,
 12583, 12589, 12600, 12603, 12616,
 12623, 12733, 12739, 12748, 12765,
 12825, 12873, 12884, 12905, 12917,
 12950, 12966, 12984, 13010, 13070,
 13075, 13102, 13107, 13130, 13139
 \l_fp_input_a_sign_int
 9902, 9902, 9948, 9950,
 10181, 10191, 10223, 10233, 10645,
 10681, 10780, 10817, 10851, 10895,
 10947, 11090, 11457, 11462, 11504,
 11590, 11596, 11644, 11651, 11687,
 11693, 11730, 11737, 11763, 11765,
 11770, 11888, 11894, 11943, 11984,
 12081, 12088, 12124, 12178, 12211,
 12230, 12263, 12286, 12327, 12399,
 12403, 12732, 12738, 12817, 12871,
 12916, 12949, 12965, 12983, 13030,
 13044, 13048, 13052, 13128, 13137
 \l_fp_input_b_decimal_int
 9902, 9908, 10119, 10120,
 10125, 10127, 10807, 10858, 10874,
 10910, 10928, 10975, 11041, 11045,
 11140, 11153, 11972, 11977, 12287,
 12330, 12331, 12333, 12335, 12338,
 12341, 12357, 12619, 12633, 12697,
 12734, 12744, 12877, 12885, 12895,
 12907, 12957, 12967, 12985, 13013,
 13028, 13080, 13112, 13133, 13140
 \l_fp_input_b_exponent_int
 . 9902, 9909, 10099, 10107, 10128,
 10132, 10808, 10911, 10929, 10933,
 11042, 11076, 11973, 11978, 12008,
 12620, 12735, 12878, 12886, 12899,
 12913, 12958, 12968, 12987, 13062,
 13066, 13094, 13098, 13135, 13142
 \l_fp_input_b_extended_int
 9910, 9911, 12288,
 12330, 12331, 12333, 12335, 12338,
 12343, 12633, 12697, 12897, 12908
 \l_fp_input_b_integer_int
 9902, 9907, 10108, 10111,
 10118, 10806, 10854, 10870, 10909,
 10927, 10986, 10990, 10993, 10997,
 11040, 11045, 11134, 11137, 11152,
 11971, 11976, 12618, 12733, 12744,
 12876, 12884, 12893, 12907, 12956,
 12966, 12984, 13013, 13028, 13070,
 13075, 13102, 13107, 13131, 13138
 \l_fp_input_b_sign_int .. 9902, 9906,

10805, 10817, 10881, 10908, 10912,
 10926, 10947, 11039, 11090, 11975,
 12286, 12327, 12340, 12732, 12781,
 12891, 12916, 12955, 12965, 12983,
 13018, 13044, 13048, 13129, 13136
 \l_fp_mul_a_i_int 9912, 9912,
 10974, 10979, 10984, 10989, 10993,
 11223, 11232, 11239, 11245, 11250,
 11256, 11259, 11267, 11276, 11284,
 11292, 11299, 11307, 11312, 11316
 \l_fp_mul_a_ii_int
 9912, 9913, 10974, 10980,
 10985, 10990, 11223, 11233, 11240,
 11246, 11251, 11257, 11267, 11277,
 11285, 11293, 11300, 11308, 11313
 \l_fp_mul_a_iii_int 9912,
 9914, 10974, 10981, 10986, 11223,
 11234, 11241, 11247, 11252, 11267,
 11278, 11286, 11294, 11301, 11309
 \l_fp_mul_a_iv_int 9912,
 9915, 11225, 11235, 11242, 11248,
 11269, 11279, 11287, 11295, 11302
 \l_fp_mul_a_v_int
 9912, 9916, 11225, 11236,
 11243, 11269, 11280, 11288, 11296
 \l_fp_mul_a_vi_int 9912, 9917,
 11225, 11237, 11269, 11281, 11289
 \l_fp_mul_b_i_int 9912, 9918,
 10976, 10981, 10985, 10989, 10992,
 11227, 11237, 11243, 11248, 11252,
 11257, 11259, 11271, 11281, 11288,
 11295, 11301, 11308, 11312, 11315
 \l_fp_mul_b_ii_int
 9912, 9919, 10976, 10980,
 10984, 10988, 11227, 11236, 11242,
 11247, 11251, 11256, 11271, 11280,
 11287, 11294, 11300, 11307, 11311
 \l_fp_mul_b_iii_int 9912,
 9920, 10976, 10979, 10983, 11227,
 11235, 11241, 11246, 11250, 11271,
 11279, 11286, 11293, 11299, 11306
 \l_fp_mul_b_iv_int 9912,
 9921, 11229, 11234, 11240, 11245,
 11273, 11278, 11285, 11292, 11298
 \l_fp_mul_b_v_int
 9912, 9922, 11229, 11233,
 11239, 11273, 11277, 11284, 11291
 \l_fp_mul_b_vi_int 9912, 9923,
 11229, 11232, 11273, 11276, 11283
 \l_fp_mul_output_int 9924, 9924, 10977,
 10982, 11015, 11016, 11020, 11022,
 11027, 11230, 11238, 11274, 11282
 \l_fp_mul_output_t1 9924, 9925,
 10978, 10995, 10996, 10999, 11026,
 11231, 11254, 11255, 11262, 11275,
 11304, 11305, 11318, 11319, 11322
 \l_fp_output_decimal_int
 9926, 9928, 10827, 10843,
 10856, 10860, 10863, 10872, 10876,
 10878, 10882, 10885, 10887, 10938,
 10951, 10964, 10995, 11070, 11081,
 11094, 11107, 11168, 11170, 11421,
 11426, 11434, 11464, 11639, 11640,
 11646, 11660, 11725, 11726, 11732,
 11746, 11778, 11793, 11797, 11802,
 11806, 11814, 11816, 11848, 11853,
 11857, 11860, 11864, 11868, 11871,
 11873, 11971, 11980, 12002, 12013,
 12027, 12154, 12198, 12218, 12220,
 12238, 12240, 12293, 12295, 12300,
 12301, 12303, 12310, 12319, 12456,
 12457, 12459, 12466, 12479, 12498,
 12510, 12521, 12523, 12575, 12578,
 12624, 12627, 12628, 12641, 12661,
 12664, 12670, 12673, 12680, 12688,
 12707, 12711, 12715, 12718, 12863,
 12877, 12896, 12909, 12918, 12922
 \l_fp_output_exponent_int 9926,
 9929, 10823, 10828, 10845, 10931,
 10939, 10966, 11074, 11082, 11110,
 11448, 11465, 11637, 11641, 11647,
 11662, 11723, 11727, 11733, 11748,
 12006, 12014, 12029, 12156, 12200,
 12222, 12242, 12311, 12321, 12467,
 12482, 12500, 12512, 12530, 12537,
 12626, 12645, 12669, 12690, 12865,
 12878, 12900, 12911, 12920, 12924
 \l_fp_output_extended_int
 9930, 9930, 11439,
 11440, 11445, 11447, 11640, 11726,
 11794, 11798, 11803, 11805, 11817,
 11849, 11851, 11854, 11865, 11867,
 11869, 11972, 11981, 12155, 12199,
 12219, 12221, 12239, 12241, 12294,
 12296, 12298, 12454, 12499, 12511,
 12522, 12524, 12576, 12579, 12629,
 12643, 12662, 12665, 12708, 12709,
 12712, 12898, 12910, 12919, 12923
 \l_fp_output_integer_int

- 9926, 9927, 10826,
10839, 10852, 10862, 10868, 10877,
10880, 10883, 10889, 10891, 10937,
10951, 10960, 10999, 11069, 11080,
11094, 11103, 11163, 11409, 11410,
11413, 11420, 11463, 11636, 11639,
11645, 11656, 11722, 11725, 11731,
11742, 11777, 11792, 11796, 11801,
11812, 11859, 11872, 12001, 12012,
12023, 12153, 12179, 12197, 12218,
12220, 12238, 12240, 12293, 12295,
12304, 12309, 12315, 12460, 12465,
12475, 12497, 12509, 12521, 12523,
12575, 12578, 12623, 12661, 12664,
12679, 12684, 12687, 12717, 12859,
12876, 12894, 12909, 12917, 12921
- \l_fp_output_sign_int
..... 9926, 9926, 10825,
10834, 10851, 10881, 10895, 10936,
11079, 11945, 11947, 11951, 11953,
12011, 12018, 12308, 12464, 12470,
12489, 12491, 12570, 12656, 12892
- \l_fp_round_carry_bool .. 9931, 9931,
10698, 10708, 10721, 10727, 10735
- \l_fp_round_decimal_tl .. 9932, 9932,
10700, 10710, 10729, 10730, 10732
- \l_fp_round_position_int . 9933, 9933,
10699, 10720, 10733, 10739, 10740
- \l_fp_round_target_int
..... 9933, 9934, 10635,
10636, 10670, 10672, 10720, 10733
- \l_fp_sign_tl
.... 9935, 9935, 12782, 12794, 12858
- \l_fp_split_sign_int
..... 9936, 9936, 9961, 9963, 9976
- \l_fp_tmp_dim . 10208, 10216, 10220, 10256
- \l_fp_tmp_int 9937,
9937, 10005, 10007, 10722–10725,
10730, 11326–11328, 11333–11335
- \l_fp_tmp_skip 10208, 10215, 10216, 10257
- \l_fp_tmp_tl .. 9938, 9938, 9958–9960,
9964, 9969, 9971, 9974, 9980, 9982,
9985, 10074, 10079, 10121, 10127,
10146, 10152, 10778, 10793, 11390,
11396, 11399, 11404, 11422, 11429,
11441, 11447, 12169, 12176, 12183,
12189, 12202, 12209, 12212, 12214,
12545, 12551, 12554, 12559, 12682,
12688, 13126, 13145, 13151, 13156
- \l_fp_trig_decimal_int
..... 9940, 9941, 11784, 11786,
11788, 11791, 11806, 11819, 11829,
11831, 11833, 11835, 11837, 11839,
11842, 11844, 11846, 11848, 11864
- \l_fp_trig_extended_int . 9940, 9942,
11784, 11786, 11788, 11790, 11805,
11820, 11829, 11831, 11833, 11835,
11837, 11839, 11842, 11844, 11850
- \l_fp_trig_octant_int
..... 9939, 9939, 11528, 11534,
11546, 11547, 11563, 11575, 11667,
11753, 11764, 11768, 11944, 11950
- \l_fp_trig_sign_int
9940, 9940, 11780, 11818, 11827, 11847
- \l_ior_stream_int
..... 8064, 8065, 8091, 8093,
8097, 8128, 8171, 8172, 8176, 8179,
8186, 8188, 8194, 8195, 8197, 8199
- \l_iow_current_indentation_int
..... 8305, 8307,
8350, 8410, 8425, 8449, 8455, 8457
- \l_iow_current_indentation_tl 8308,
8310, 8351, 8408, 8428, 8450, 8456
- \l_iow_current_line_int
..... 8305, 8305, 8352,
8396, 8397, 8409, 8415, 8422, 8443
- \l_iow_current_line_tl
..... 8308, 8308, 8353, 8407,
8413, 8421, 8427, 8442, 8444, 8462
- \l_iow_current_word_int
..... 8305, 8306, 8394, 8396, 8424
- \l_iow_current_word_tl .. 8308, 8309,
8387, 8388, 8395, 8408, 8414, 8428
- \l_iow_line_length_int
..... 141, 8302, 8302, 8303, 8349
- \l_iow_line_start_bool
.. 8313, 8313, 8355, 8404, 8406, 8445
- \l_iow_stream_int 8064,
8064, 8065, 8104, 8106, 8110, 8120,
8134, 8135, 8139, 8142, 8144, 8149,
8151, 8157, 8158, 8160, 8162, 8181
- \l_iow_target_length_int
..... 8304, 8304, 8349, 8397
- \l_iow_wrap_tl
.. 8311, 8311, 8354, 8368, 8371, 8377
- \l_iow_wrapped_tl
.. 8312, 8312, 8383, 8420, 8441, 8461
- \l_keys_choice_int .. 159, 9219, 9219,
9333, 9339, 9340, 9343, 9352, 9365,

- 9366, 9370, 9427, 9433, 9434, 9437
- \l_keys_choice_tl . 159, 9338, 9364, 9432
- \l_keys_choices_tl 9219, 9220
- \l_keys_key_tl 160, 9221, 9221, 9301, 9316, 9586, 9587, 9648
- \l_keys_module_tl 9222, 9222, 9229, 9232, 9234, 9258, 9405, 9410, 9551, 9554, 9556, 9561, 9564, 9569, 9587, 9637, 9640
- \l_keys_no_value_bool 9223, 9223, 9239, 9244, 9275, 9576, 9581, 9592, 9602, 9614, 9649
- \l_keys_path_tl 160, 9224, 9224, 9253, 9258, 9265, 9268, 9283, 9294, 9296, 9298, 9309, 9311, 9313, 9322, 9324, 9327, 9336, 9349, 9357, 9362, 9368, 9375, 9378, 9380, 9400, 9404, 9409, 9416, 9418, 9421, 9430, 9443, 9449, 9469, 9471, 9587, 9596, 9606, 9616, 9618, 9621, 9629, 9634, 9640, 9664, 9665
- \l_keys_property_tl . 9225, 9225, 9249, 9253, 9271, 9278, 9279, 9282, 9286
- \l_keys_unknown_clist 9226, 9226, 9565, 9570, 9646
- \l_keys_value_tl 160, 9227, 9227, 9606, 9613, 9620, 9650, 9658
- \l_keyval_key_tl 9108, 9108, 9155, 9168, 9177
- \l_keyval_parse_tl . . 9110, 9111, 9127, 9131, 9151, 9173, 9182, 9186, 9195
- \l_keyval_sanitise_tl 9110, 9110, 9123–9126, 9129
- \l_keyval_value_tl 9108, 9109, 9179, 9181, 9184, 9194, 9196
- \l_last_box 6903, 6903
- \l_msg_class_tl 8790, 8791, 8841, 8842, 8845
- \l_msg_current_class_tl 8790, 8792, 8823, 8842
- \l_msg_current_module_tl 8790, 8793, 8824
- \l_msg_redirect_classes_prop 8697, 8697
- \l_msg_redirect_classes_seq 8790, 8790, 8798, 8803, 8806
- \l_msg_redirect_kernel_info_prop . 8942
- \l_msg_redirect_kernel_warning_prop 8920
- \l_msg_redirect_names_prop 8697, 8698, 8825, 8856
- \l_msg_text_tl 8603, 8603, 8636, 8669, 8671
- \l_msg_tmp_tl 8524, 8524, 8640, 8643, 8651
- \l_peek_search_tl 2875, 2875, 2893, 2914, 2957
- \l_peek_search_token 2874, 2874, 2892, 2913, 2932, 2940
- \l_peek_token 54, 2872, 2872, 2881, 2932, 2940, 2950–2952, 2971, 3100–3102
- \l_prop_show_tl 6250, 6250, 6264, 6267, 8243, 8246, 8263, 8266
- \l_seq_remove_seq 5093, 5093, 5100, 5103, 5104, 5106
- \l_seq_show_tl . . . 5333, 5333, 5347, 5350
- \l_seq_tmpa_tl 5051, 5051, 5125, 5130, 5144, 5148, 5545, 5551, 5556, 5557
- \l_seq_tmpb_tl 5051, 5052, 5121, 5125, 5147, 5148
- \l_tl_replace_tl 4415, 4415
- \l_tl_rescan_tl 4358, 4358, 4370, 4373, 4379, 4396, 4398, 4411
- \l_tl_tmpa_tl 4533, 4536, 4538, 4546
- \l_tl_tmpb_tl 4533, 4537, 4538, 4547
- \l_tmpa_bool 36, 1934, 1934
- \l_tmpa_box 127, 6467, 6468, 6471
- \l_tmpa_clist 113, 5914, 5914
- \l_tmpa_dim 75, 4105, 4105
- \l_tmpa_int 67, 3867, 3867
- \l_tmpa_skip 78, 4179, 4179
- \l_tmpa_tl 4, 94, 4831, 4831
- \l_tmpb_box 127, 6467, 6473
- \l_tmpb_clist 113, 5914, 5915
- \l_tmpb_dim 75, 4105, 4106
- \l_tmpb_int 67, 3867, 3868
- \l_tmpb_skip 78, 4179, 4180
- \l_tmpb_tl 94, 4831, 4832
- \l_tmpc_dim 75, 4105, 4107
- \l_tmpc_int 67, 3867, 3869
- \l_tmpc_skip 78, 4179, 4181
- \language 449
- \lastbox 606
- \lastkern 539
- \lastlinefit 719
- \lastnodetype 700
- \lastpenalty 645
- \lastskip 540
- \latelua 761
- \lccode 668
- \leaders 536
- \left 504
- \lefthyphenmin 560
- \leftskip 562
- \leqno 479

<code>\let</code>	59, 230, 336, 337, 349	<code>\mathbin</code>	491
<code>\limits</code>	496	<code>\mathchar</code>	462
<code>\linepenalty</code>	552	<code>\mathchardef</code>	359
<code>\lineskip</code>	546	<code>\mathchoice</code>	459
<code>\lineskiplimit</code>	547	<code>\mathclose</code>	492
<code>\long</code>	33, 339, 368	<code>\mathcode</code>	670
<code>\looseness</code>	564	<code>\mathinner</code>	493
<code>\lower</code>	601	<code>\mathop</code>	494
<code>\lowercase</code>	640	<code>\mathopen</code>	498
<code>\lua_now:n</code>	174, 13237, 13254	<code>\mathord</code>	499
<code>\lua_now:x</code>	4821, 13237, 13239, 13243, 13246, 13255	<code>\mathparagraph</code>	3892
<code>\lua_shipout:n</code> ..	174, 13237, 13257, 13259	<code>\mathpunct</code>	500
<code>\lua_shipout:x</code>	13237	<code>\mathrel</code>	501
<code>\lua_shipout_x:n</code>	175, 13237, 13240, 13248, 13251, 13256, 13258	<code>\mathsection</code>	3891
<code>\lua_shipout_x:x</code>	13237	<code>\mathsurround</code>	512
<code>\lua_wrong_engine:</code>	13283, 13314, 13315, 13333	<code>\maxdeadcycles</code>	582
<code>\luaescapestring</code>	39, 40	<code>\maxdepth</code>	583
<code>\luatex_catcodetable:D</code>	758, 773, 13296, 13297, 13302, 13310	<code>\maxdimen</code>	4103
<code>\luatex_directlua:D</code>	759, 1467, 13239	<code>\meaning</code>	642
<code>\luatex_if_engine:</code>	1444	<code>\medmuskip</code>	513
<code>\luatex_if_engine:F</code>	1445, 1470, 13282, 13312, 13332	<code>\message</code>	419
<code>\luatex_if_engine:T</code> .	1444, 1469, 4819, 13285, 13318, 13334, 13340, 13349	<code>\MessageBreak</code>	222, 238–244
<code>\luatex_if_engine:TF</code> ..	1446, 1471, 13237	<code>\middle</code>	724
<code>\luatex_if_engine_p:</code> ..	1453, 1475, 1521	<code>\mkern</code>	466
<code>\luatex_if_engineTF</code>	21	<code>\mode_if_horizontal:</code>	2271, 2271
<code>\luatex_initcatcodetable:D</code>	760, 774, 13273, 13290	<code>\mode_if_horizontalTF</code>	40
<code>\luatex_latelua:D</code>	761, 775, 13240	<code>\mode_if_inner:</code>	2273, 2273
<code>\luatex luatexversion:D</code>	762	<code>\mode_if_innerTF</code>	40
<code>\luatex_savecatcodetable:D</code>	763, 776, 13301, 13329	<code>\mode_if_math:</code>	2275, 2275
<code>\luatexcatcodetable</code>	773	<code>\mode_if_math:TF</code>	3880
<code>\luatexinitcatcodetable</code>	774	<code>\mode_if_mathTF</code>	40
<code>\luatexlatelua</code>	775	<code>\mode_if_vertical:</code>	2269, 2269
<code>\luatexsavecatcodetable</code>	776	<code>\mode_if_verticalTF</code>	41
<code>\luatexversion</code>	762	<code>\month</code>	652
M		<code>\moveleft</code>	602
<code>\M</code>	2681, 2724	<code>\moveright</code>	603
<code>\m@ne</code>	1168	<code>\msg_class_new:nn</code>	9081, 9082
<code>\mag</code>	448	<code>\msg_class_set:nn</code>	146, 8699, 8699, 8724, 8735, 8746, 8768, 8776, 8784, 8789, 9082
<code>\mark</code>	450	<code>\msg_critical:nn</code>	8735
<code>\marks</code>	676	<code>\msg_critical:nnx</code>	8735
<code>\mathaccent</code>	461	<code>\msg_critical:nnxx</code>	8735
		<code>\msg_critical:nnxxx</code>	8735
		<code>\msg_critical:nnxxxx</code>	146, 8735
		<code>\msg_critical_text:n</code> 145, 8690, 8691, 8738	
		<code>\msg_direct_interrupt:xxxxx</code> .	9091, 9096
		<code>\msg_direct_log:xx</code>	9091, 9097
		<code>\msg_direct_term:xx</code>	9091, 9098
		<code>\msg_error:nn</code>	8746

\msg_error:nnx	8746	\msg_kernel_error:nn 1184 , 1198 , 7245 , 8094 ,
\msg_error:nnxx	8746		8107 , 8531 , 8804 , 8885 , 8918 , 9161 ,
\msg_error:nnxxx	8746		13207 , 13215 , 13220 , 13229 , 13300
\msg_error:nnxxxx	147 , 8746	\msg_kernel_error:nnx 1184 , 1196 , 1439 , 4432 ,
\msg_error_text:n	145 , 8690 , 8692 , 8751 , 8760 , 8890 , 8903		5249 , 6967 , 6972 , 8813 , 8885 , 8916 ,
\msg_expandable_error:n 151 , 1587 , 2218 , 4618 , 5048 ,		9261 , 9300 , 9315 , 9356 , 9595 , 13278
	5518 , 8434 , 9057 , 9065 , 13245 , 13250	\msg_kernel_error:nnxx	1184 , 1184 , 1197 , 1199 , 1206 , 1216 ,
\msg_expandable_error_aux:w .	9071 , 9078		1229 , 1349 , 7111 , 8819 , 8885 , 8914 ,
\msg_fatal:nn	8724		9252 , 9281 , 9326 , 9420 , 9605 , 9639
\msg_fatal:nnx	8724	\msg_kernel_error:nnxxx	8885 , 8912
\msg_fatal:nnxx	8724	\msg_kernel_error:nnxxxx	150 , 8885 , 8885 , 8913 , 8915 , 8917 , 8919
\msg_fatal:nnxxx	8724		\msg_kernel_fatal:nn
\msg_fatal:nnxxxx	146 , 8724		8865 , 8883
\msg_fatal_text:n 145 , 8690 , 8690 , 8727 , 8868	\msg_kernel_fatal:nnx .	8865 , 8881 , 13275
\msg_generic_new:nn	9091 , 9093	\msg_kernel_fatal:nnxx	8865 , 8879
\msg_generic_new:nnn	9091 , 9092	\msg_kernel_fatal:nnxxx	8865 , 8877
\msg_generic_set:nn	9091 , 9095	\msg_kernel_fatal:nnxxxx	150 , 8865 , 8865 , 8878 , 8880 , 8882 , 8884
\msg_generic_set:nnn	9091 , 9094	\msg_kernel_info:nn	8920 , 8962
\msg_gset:nnn	8527 , 8554	\msg_kernel_info:nnx	8920 , 8960
\msg_gset:nnnn ...	8527 , 8534 , 8547 , 8555	\msg_kernel_info:nnxx	8920 , 8958
\msg_if_more_text:c	8713	\msg_kernel_info:nnxxx	8920 , 8956
\msg_if_more_text:cTF	8748 , 8887	\msg_kernel_info:nnxxxx	150 , 8920 , 8943 , 8957 , 8959 , 8961 , 8963
\msg_if_more_text:N	8713 , 8713	\msg_kernel_new:nnn	8857 , 8859
\msg_if_more_text:NF	8722	\msg_kernel_new:nnnn	150 , 7986 , 7994 ,
\msg_if_more_text:NT	8721		7997 , 8270 , 8277 , 8857 , 8857 , 8964 ,
\msg_if_more_text:NTF	8723		8973 , 8981 , 8988 , 8995 , 9003 , 9012 ,
\msg_if_more_text_p:N	8720		9019 , 9026 , 9033 , 9208 , 9681 , 9684 ,
\msg_info:nn	8776		9690 , 9697 , 9706 , 9712 , 9718 , 9725 ,
\msg_info:nnx	8776		9732 , 9738 , 13200 , 13208 , 13216 , 13222
\msg_info:nnxx	8776	\msg_kernel_set:nnn	8857 , 8863
\msg_info:nnxxx	8776	\msg_kernel_set:nnnn ...	150 , 8857 , 8861
\msg_info:nnxxxx	147 , 8776	\msg_kernel_warning:nn	8920 , 8940
\msg_info_text:n	146 , 8690 , 8694 , 8780 , 8949	\msg_kernel_warning:nnx	8920 , 8938
\msg_interrupt:xxx 149 , 8604 , 8604 , 8726 , 8737 ,	\msg_kernel_warning:nnxx	8920 , 8936
	8750 , 8759 , 8867 , 8889 , 8902 , 9042	\msg_kernel_warning:nnxxx ...	8920 , 8934
\msg_interrupt_aux:	8604 , 8610 , 8660	\msg_kernel_warning:nnxxxx	150 , 8920 , 8921 , 8935 , 8937 , 8939 , 8941
\msg_interrupt_details:xxx 8604 , 8609 , 8627	\msg_line_context	145
\msg_interrupt_more_text:n 8604 , 8623 , 8631 , 8637	\msg_line_context: 1200 , 1200 , 1219 , 8590 , 8591
\msg_interrupt_no_details:xx 8604 , 8608 , 8619	\msg_line_number	145
\msg_interrupt_text:n 8604 , 8625 , 8633 , 8635	\msg_line_number:	8590 , 8590 , 8595 , 9209
\msg_kernel_bug:x	9040 , 9040	\msg_log:nn	8784 , 9089
		\msg_log:nnx	8784 , 9088
		\msg_log:nnxx	8784 , 9087

<code>\msg_log:nnxxx</code>	8784, 9086	<code>\muskip_add:Nn</code>	78, 4208, 4208, 4210, 4211
<code>\msg_log:nnxxxx</code>	147, 8784, 9085	<code>\muskip_eval:n</code>	79, 4218, 4218
<code>\msg_log:x</code>	149, 8675, 8675, 8778, 8786, 8947	<code>\muskip_gadd:cn</code>	4208
<code>\msg_new:nnn</code>	8527, 8536, 8860	<code>\muskip_gadd:Nn</code>	79, 4208, 4210, 4212
<code>\msg_new:nnnn</code>	144, 8527, 8527, 8537, 8858	<code>\muskip_gset:cn</code>	4197
<code>\msg_newline</code>	148	<code>\muskip_gset:Nn</code>	79, 4197, 4199, 4201
<code>\msg_newline:</code>	8588, 8588, 8648	<code>\muskip_gset_eq:cc</code>	4202
<code>\msg_no_more_text:xxxx</code>	8713, 8715, 8719	<code>\muskip_gset_eq:cN</code>	4202
<code>\msg_none:nn</code>	8789	<code>\muskip_gset_eq:Nc</code>	4202
<code>\msg_none:nnx</code>	8789	<code>\muskip_gset_eq:NN</code>	79, 4202, 4205–4207
<code>\msg_none:nnxx</code>	8789	<code>\muskip_gsub:cn</code>	4208
<code>\msg_none:nnxxx</code>	8789	<code>\muskip_gsub:Nn</code>	79, 4208, 4215, 4217
<code>\msg_none:nnxxxx</code>	147, 8789	<code>\muskip_gzero:c</code>	4192
<code>\msg_redirect_class:nn</code>	148, 8851, 8851	<code>\muskip_gzero:N</code>	78, 4192, 4194, 4196
<code>\msg_redirect_module:nnn</code>	148, 8853, 8853	<code>\muskip_new:c</code>	4184
<code>\msg_redirect_name:nnn</code>	148, 8855, 8855	<code>\muskip_new:N</code>	78, 4184, 4185, 4191
<code>\msg_see_documentation_text:n</code>	8695, 8695, 8730, 8741, 8754, 8763, 8872, 8894, 8907, 9045	<code>\muskip_set:cn</code>	4197
<code>\msg_set:nnn</code>	8527, 8545, 8864	<code>\muskip_set:Nn</code>	79, 4197, 4197, 4199, 4200
<code>\msg_set:nnnn</code>	144, 8527, 8538, 8546, 8862	<code>\muskip_set_eq:cc</code>	4202
<code>\msg_term:x</code>	149, 8675, 8682, 8770, 8925	<code>\muskip_set_eq:cN</code>	4202
<code>\msg_trace:nn</code>	9084, 9089	<code>\muskip_set_eq:Nc</code>	4202
<code>\msg_trace:nnx</code>	9084, 9088	<code>\muskip_set_eq:NN</code>	79, 4202, 4202–4204
<code>\msg_trace:nnxx</code>	9084, 9087	<code>\muskip_show:c</code>	4222
<code>\msg_trace:nnxxx</code>	9084, 9086	<code>\muskip_show:N</code>	80, 4222, 4222, 4223
<code>\msg_trace:nnxxxx</code>	9084, 9085	<code>\muskip_sub:cn</code>	4208
<code>\msg_two_newlines</code>	148	<code>\muskip_sub:Nn</code>	79, 4208, 4213, 4215, 4216
<code>\msg_two_newlines:</code>	8588, 8589	<code>\muskip_use:c</code>	4220
<code>\msg_use:nnnnxxxx</code>	8703, 8794, 8794, 8923, 8945	<code>\muskip_use:N</code>	80, 4219, 4220, 4220, 4221
<code>\msg_use_aux:nn</code>	8794, 8827, 8829	<code>\muskip_zero:c</code>	4192
<code>\msg_use_aux:nnn</code>	8794, 8818, 8821	<code>\muskip_zero:N</code>	78, 4192, 4192, 4194, 4195
<code>\msg_use_code:</code>	8794, 8796, 8836, 8844, 8848	<code>\muskipdef</code>	358
<code>\msg_use_loop:n</code>	8794, 8801, 8849, 8850	<code>\mutoglu</code>	718
<code>\msg_use_loop:o</code>	8794, 8845		
<code>\msg_use_loop_check:nn</code>	8794, 8826, 8832, 8835, 8839		
<code>\msg_warning:nn</code>	8768		
<code>\msg_warning:nnx</code>	8768		
<code>\msg_warning:nnxx</code>	8768		
<code>\msg_warning:nnxxx</code>	8768		
<code>\msg_warning:nnxxxx</code>	147, 8768		
<code>\msg_warning_text:n</code>	146, 8690, 8693, 8772, 8927		
<code>\mskip</code>	463		
<code>\muexpr</code>	712		
<code>\multiply</code>	364		
<code>\muskip</code>	660		
<code>\muskip_add:cn</code>	4208		

<code>\noexpand</code>	35, 39, 40, 166, 169, 172, 174, 175, 184, 187–189, 191, 193, 194, 203, 205–209, 268, 270, 275, 277, 375	<code>\parshapeindent</code>	706
<code>\noindent</code>	543	<code>\parshapelength</code>	707
<code>\nolimits</code>	497	<code>\parskip</code>	565
<code>\nonscript</code>	477	<code>\patterns</code>	648
<code>\nonstopmode</code>	440	<code>\pausing</code>	435
<code>\nulldelimiterspace</code>	510	<code>\pdf@strcmp</code>	59
<code>\nullfont</code>	628	<code>\pdfcolorstack</code>	738
<code>\number</code>	637	<code>\pdfcompresslevel</code>	739
<code>\numexpr</code>	709	<code>\pdfcreationdate</code>	737
O			
<code>\O</code>	1838, 2724	<code>\pdfdecimaldigits</code>	740
<code>\omit</code>	383	<code>\pdfhorigin</code>	741
<code>\openin</code>	409	<code>\pdfinfo</code>	742
<code>\openout</code>	410	<code>\pdflastxform</code>	743
<code>\or</code>	22, 69, 406	<code>\pdfliteral</code>	744
<code>\or:</code>	785, 787, 1326–1334, 1526, 3625–3649, 11668, 11670, 11672, 11674, 11754, 11756, 11758, 11760	<code>\pdfminorversion</code>	745
<code>\outer</code>	369	<code>\pdfobjcompresslevel</code>	746
<code>\output</code>	584	<code>\pdfoutput</code>	747
<code>\outputpenalty</code>	594	<code>\pdfpkresolution</code>	752
<code>\over</code>	471	<code>\pdfrefxform</code>	748
<code>\overfullrule</code>	622	<code>\pdfrestore</code>	749
<code>\overline</code>	502	<code>\pdfsave</code>	750
<code>\overwithdelims</code>	472	<code>\pdfsetmatrix</code>	751
P			
<code>\P</code>	1836, 2724	<code>\pdfstrcmp</code>	33, 59, 230, 235, 238, 252, 756
<code>\package_check_loaded_expl:</code>	783, 1534, 1890, 2401, 2491, 3211, 3919, 4242, 5044, 5581, 6044, 6365, 6909, 8009, 8029, 8522, 9105, 9754, 9871, 13235	<code>\pdftex_if_engine:</code>	1444
<code>\PackageError</code>	219, 235	<code>\pdftex_if_engine:F</code>	1448, 1459, 1473
<code>\pagedepth</code>	586	<code>\pdftex_if_engine:T</code>	1447, 1458, 1472
<code>\pagediscards</code>	727	<code>\pdftex_if_engine:TF</code>	1449, 1460, 1474
<code>\pagefilllstretch</code>	590	<code>\pdftex_if_engine_p:</code>	1454, 1464, 1476, 1522
<code>\pagefillstretch</code>	589	<code>\pdftex_if_engineTF</code>	21
<code>\pagefilstretch</code>	588	<code>\pdftex_pdfcolorstack:D</code>	738
<code>\pagegoal</code>	592	<code>\pdftex_pdfcompresslevel:D</code>	739
<code>\pageshrink</code>	591	<code>\pdftex_pdfcreationdate:D</code>	737
<code>\pagestretch</code>	587	<code>\pdftex_pdfdecimaldigits:D</code>	740
<code>\pagetotal</code>	593	<code>\pdftex_pdfhorigin:D</code>	741
<code>\par</code>	542	<code>\pdftex_pdfinfo:D</code>	742
<code>\parfillskip</code>	573	<code>\pdftex_pdflastxform:D</code>	743
<code>\parindent</code>	566	<code>\pdftex_pdfliteral:D</code>	744
<code>\parshape</code>	558	<code>\pdftex_pdfminorversion:D</code>	745
<code>\parshapedimen</code>	708	<code>\pdftex_pdfobjcompresslevel:D</code>	746
		<code>\pdftex_pdfoutput:D</code>	747
		<code>\pdftex_pdfpkresolution:D</code>	752
		<code>\pdftex_pdfrefxform:D</code>	748
		<code>\pdftex_pdfrestore:D</code>	749
		<code>\pdftex_pdfsave:D</code>	750
		<code>\pdftex_pdfsetmatrix:D</code>	751
		<code>\pdftex_pdftextrevision:D</code>	753
		<code>\pdftex_pdfvorigin:D</code>	754
		<code>\pdftex_pdfxform:D</code>	755

- \pdfstrcmp:D 756,
1481, 1487, 2426, 2433, 2464, 2473,
2696, 4146, 4797, 4836, 9968, 9979
- \pdfxrevision 753
- \pdfvorigin 754
- \pdfxform 755
- \peek_after:NN 3137, 3138
- \peek_after:Nw
54, 2880, 2880, 2905, 2923, 2979, 3138
- \peek_catcode:N 2997
- \peek_catcode:NTF 54
- \peek_catcode_ignore_spaces:N ... 2997
- \peek_catcode_ignore_spaces:NTF ... 54
- \peek_catcode_remove:N 2997
- \peek_catcode_remove:NTF 55
- \peek_catcode_remove_ignore_spaces:N
..... 2997
- \peek_catcode_remove_ignore_spaces:NTF
..... 55
- \peek_charcode:N 3013
- \peek_charcode:NTF 55
- \peek_charcode_ignore_spaces:N .. 3013
- \peek_charcode_ignore_spaces:NTF ... 55
- \peek_charcode_remove:N 3013
- \peek_charcode_remove:NTF 55
- \peek_charcode_remove_ignore_spaces:N
..... 3013
- \peek_charcode_remove_ignore_spaces:NTF
..... 55
- \peek_def:nnnn 2980, 2981,
2997, 3001, 3005, 3009, 3013, 3017,
3021, 3025, 3029, 3033, 3037, 3041
- \peek_def_aux:nnnnn 2980, 2983–2985, 2987
- \peek_execute_branches: 2976, 2992
- \peek_execute_branches_catcode:
.. 2929, 2929, 3000, 3002, 3008, 3010
- \peek_execute_branches_charcode: ...
.. 2946, 2946, 3016, 3018, 3024, 3026
- \peek_execute_branches_charcode:NN 2946
- \peek_execute_branches_charcode_aux:NN
..... 2956, 2960
- \peek_execute_branches_meaning:
.. 2929, 2938, 3032, 3034, 3040, 3042
- \peek_execute_branches_N_type:
..... 3096, 3096, 3108, 3110, 3112
- \peek_false:w 2876, 2878, 2899,
2917, 2935, 2943, 2954, 2965, 3104
- \peek_gafter:NN 3137, 3139
- \peek_gafter:Nw 54, 2882, 3139
- \peek_ignore_spaces_execute_branches:
..... 2969, 2969, 2979,
3004, 3012, 3020, 3028, 3036, 3044
- \peek_ignore_spaces_execute_branches_aux:
..... 2969, 2973, 2978
- \peek_meaning:N 3029
- \peek_meaning:NTF 56
- \peek_meaning_ignore_spaces:N ... 3029
- \peek_meaning_ignore_spaces:NTF ... 56
- \peek_meaning_remove:N 3029
- \peek_meaning_remove:NTF 56
- \peek_meaning_remove_ignore_spaces:N
..... 3029
- \peek_meaning_remove_ignore_spaces:NTF
..... 56
- \peek_N_type: 3096
- \peek_N_type:F 3111
- \peek_N_type:T 3109
- \peek_N_type:TF 3107
- \peek_N_typeTF 58
- \peek_tmp:w 2876, 2879, 2888, 2974
- \peek_token_generic:NN 2890
- \peek_token_generic:NNF 2909, 3112
- \peek_token_generic:NNT 2907, 3110
- \peek_token_generic:NNTF
..... 2890, 2908, 2910, 3108
- \peek_token_remove_generic:NN ... 2911
- \peek_token_remove_generic:NNF .. 2927
- \peek_token_remove_generic:NNT .. 2925
- \peek_token_remove_generic:NNTF
..... 2911, 2926, 2928
- \peek_true:w 2876, 2876,
2894, 2915, 2933, 2941, 2963, 3105
- \peek_true_aux:w . 2876, 2877, 2887, 2916
- \peek_true_remove:w 2884, 2884, 2915
- \penalty 643
- \postdisplaypenalty 490
- \predisplaydirection 734
- \predisdisplaypenalty 489
- \predisplaysize 488
- \pretolerance 569
- \prevdepth 616
- \prevgraf 575
- \prg_case_dim:nnn 39, 2127, 2127
- \prg_case_dim_aux:nnn .. 2127, 2130, 2132
- \prg_case_dim_aux:nw 2127, 2133, 2134, 2138
- \prg_case_end:nw 2113,
2113, 2124, 2137, 2148, 2160, 2171
- \prg_case_int:nnn 38, 2114, 2114, 3514, 3520
- \prg_case_int_aux:nnn .. 2114, 2117, 2119

- \prg_case_int_aux:nw [2114](#), [2120](#), [2121](#), [2125](#)
- \prg_case_str:nnn [39](#), [2140](#), [2140](#), [2151](#), [4981](#)
- \prg_case_str:onn [2140](#)
- \prg_case_str:xxn [2140](#), [2152](#)
- \prg_case_str_aux:nw [2140](#), [2143](#), [2145](#), [2149](#)
- \prg_case_str_x_aux:nw
..... [2140](#), [2155](#), [2157](#), [2161](#)
- \prg_case_tl:cnm [2163](#)
- \prg_case_tl:Nnn ... [39](#), [2163](#), [2163](#), [2174](#)
- \prg_case_tl_aux:Nw [2163](#), [2166](#), [2168](#), [2172](#)
- \prg_conditional_form_F:nnn [1055](#)
- \prg_conditional_form_p:nnn [1052](#)
- \prg_conditional_form_T:nnn [1054](#)
- \prg_conditional_form_TF:nnn [1053](#)
- \prg_define_quicksort:nnn [2314](#), [2314](#), [2389](#)
- \prg_do_nothing [9](#)
- \prg_do_nothing: [1478](#), [1478](#), [4455](#), [4460](#),
[4612](#), [4799](#), [5527](#), [5530](#), [5533](#), [5547](#),
[5554](#), [5977](#), [5981](#), [5988](#), [9974](#), [9985](#)
- \prg_generate_conditional_aux:nnNNnnnn
..... [898](#),
[906](#), [913](#), [921](#), [929](#), [938](#), [946](#), [955](#), [966](#)
- \prg_generate_conditional_aux:nnw ..
..... [968](#), [974](#), [980](#)
- \prg_generate_conditional_parm_aux:nnNNnnnn
..... [966](#)
- \prg_generate_conditional_parm_aux:nw
..... [966](#)
- \prg_generate_F_form_count:Nnnnn
..... [1011](#), [1027](#)
- \prg_generate_F_form_parm:Nnnnn [982](#), [998](#)
- \prg_generate_p_form_count:Nnnnn
..... [1011](#), [1011](#)
- \prg_generate_p_form_parm:Nnnnn [982](#), [982](#)
- \prg_generate_T_form_count:Nnnnn
..... [1011](#), [1019](#)
- \prg_generate_T_form_parm:Nnnnn [982](#), [990](#)
- \prg_generate_TF_form_count:Nnnnn ..
..... [1011](#), [1035](#)
- \prg_generate_TF_form_parm:Nnnnn
..... [982](#), [1006](#)
- \prg_get_count_aux:nn
..... [927](#), [936](#), [945](#), [953](#), [964](#), [964](#)
- \prg_get_parm_aux:nw
..... [896](#), [904](#), [912](#), [919](#), [964](#), [965](#)
- \prg_new_conditional:Nnn [925](#),
[934](#), [1894](#), [2442](#), [2450](#), [2462](#), [2471](#), [8483](#)
- \prg_new_conditional:Npnn
..... [32](#), [894](#), [902](#), [1417](#),
[1479](#), [1485](#), [1894](#), [1922](#), [1936](#), [2269](#),
[2271](#), [2273](#), [2275](#), [2607](#), [2612](#), [2617](#),
[2622](#), [2629](#), [2635](#), [2640](#), [2645](#), [2650](#),
[2655](#), [2660](#), [2665](#), [2670](#), [2675](#), [2689](#),
[2703](#), [2708](#), [2729](#), [2736](#), [2743](#), [2754](#),
[2765](#), [2776](#), [2787](#), [2796](#), [2803](#), [2821](#),
[3357](#), [3427](#), [3435](#), [3443](#), [3983](#), [3988](#),
[4143](#), [4153](#), [4476](#), [4498](#), [4519](#), [4521](#),
[4711](#), [4727](#), [4743](#), [4777](#), [4779](#), [4794](#),
[4847](#), [4849](#), [6164](#), [6176](#), [6437](#), [6439](#),
[6449](#), [9626](#), [9667](#), [9673](#), [12928](#), [12936](#)
- \prg_new_eq_conditional:NN [34](#)
- \prg_new_eq_conditional:NNn
..... [960](#), [962](#), [1894](#),
[5136](#), [5138](#), [5782](#)–[5787](#), [6355](#)–[6358](#)
- \prg_new_map_functions:Nn ... [2392](#), [2393](#)
- \prg_new_protected_conditional:Nnn ..
..... [925](#), [951](#), [1894](#), [9821](#)
- \prg_new_protected_conditional:Npnn
..... [33](#), [894](#), [917](#),
[1894](#), [4533](#), [4554](#), [5140](#), [5367](#), [5375](#),
[5388](#), [5395](#), [5402](#), [5409](#), [5788](#), [5792](#),
[6205](#), [6279](#), [6285](#), [12944](#), [12961](#), [13148](#)
- \prg_quicksort:n [42](#), [2389](#)
- \prg_quicksort_compare:nnTF
..... [42](#), [2390](#), [2391](#)
- \prg_quicksort_function:n [42](#), [2390](#), [2390](#)
- \prg_replicate:nn [39](#),
[2175](#), [2175](#), [8457](#), [10292](#), [10328](#), [10398](#)
- \prg_replicate_ [2175](#), [2186](#)
- \prg_replicate_0:n [2175](#)
- \prg_replicate_1:n [2175](#)
- \prg_replicate_2:n [2175](#)
- \prg_replicate_3:n [2175](#)
- \prg_replicate_4:n [2175](#)
- \prg_replicate_5:n [2175](#)
- \prg_replicate_6:n [2175](#)
- \prg_replicate_7:n [2175](#)
- \prg_replicate_8:n [2175](#)
- \prg_replicate_9:n [2175](#)
- \prg_replicate_aux:N [2175](#), [2182](#), [2183](#), [2185](#)
- \prg_replicate_first_~:n [2175](#)
- \prg_replicate_first_0:n [2175](#)
- \prg_replicate_first_1:n [2175](#)
- \prg_replicate_first_2:n [2175](#)
- \prg_replicate_first_3:n [2175](#)
- \prg_replicate_first_4:n [2175](#)
- \prg_replicate_first_5:n [2175](#)
- \prg_replicate_first_6:n [2175](#)
- \prg_replicate_first_7:n [2175](#)
- \prg_replicate_first_8:n [2175](#)

`\prg_replicate_first_9:n` [2175](#)
`\prg_replicate_first_aux:N`
 [2175](#), [2178](#), [2184](#)
`\prg_return_false` [34](#)
`\prg_return_false:`
 [890](#), [892](#), [1098](#), [1103](#), [1116](#),
 [1121](#), [1129](#), [1146](#), [1420](#), [1483](#), [1488](#),
 [1894](#), [1927](#), [1941](#), [2270](#), [2272](#), [2274](#),
 [2276](#), [2447](#), [2455](#), [2468](#), [2477](#), [2610](#),
 [2615](#), [2620](#), [2625](#), [2632](#), [2638](#), [2643](#),
 [2648](#), [2653](#), [2658](#), [2663](#), [2668](#), [2673](#),
 [2678](#), [2699](#), [2706](#), [2713](#), [2715](#), [2735](#),
 [2742](#), [2746](#), [2753](#), [2757](#), [2764](#), [2775](#),
 [2779](#), [2786](#), [2795](#), [2802](#), [2811](#), [2824](#),
 [2843](#), [2860](#), [2869](#), [3376](#), [3384](#), [3390](#),
 [3400](#), [3408](#), [3414](#), [3422](#), [3432](#), [3440](#),
 [3446](#), [3986](#), [3994](#), [4150](#), [4161](#), [4491](#),
 [4503](#), [4516](#), [4526](#), [4543](#), [4558](#), [4720](#),
 [4740](#), [4755](#), [4763](#), [4773](#), [4789](#), [4803](#),
 [4840](#), [5153](#), [5363](#), [5802](#), [6169](#), [6194](#),
 [6209](#), [6283](#), [6289](#), [6438](#), [6440](#), [6450](#),
 [8493](#), [8716](#), [9631](#), [9671](#), [9677](#), [9825](#),
 [12933](#), [12941](#), [12978](#), [12992](#), [12996](#),
 [13000](#), [13004](#), [13016](#), [13020](#), [13035](#),
 [13050](#), [13068](#), [13077](#), [13085](#), [13100](#),
 [13109](#), [13117](#), [13162](#), [13168](#), [13174](#),
 [13180](#), [13186](#), [13192](#), [13197](#), [13198](#)
`\prg_return_true` [34](#)
`\prg_return_true:` ... [890](#), [890](#), [1101](#),
 [1118](#), [1126](#), [1131](#), [1144](#), [1149](#), [1420](#),
 [1483](#), [1488](#), [1894](#), [1925](#), [1939](#), [2270](#),
 [2272](#), [2274](#), [2276](#), [2445](#), [2453](#), [2466](#),
 [2475](#), [2610](#), [2615](#), [2620](#), [2625](#), [2632](#),
 [2638](#), [2643](#), [2648](#), [2653](#), [2658](#), [2663](#),
 [2668](#), [2673](#), [2678](#), [2697](#), [2706](#), [2713](#),
 [2735](#), [2742](#), [2753](#), [2764](#), [2775](#), [2786](#),
 [2795](#), [2802](#), [2811](#), [2841](#), [2867](#), [3374](#),
 [3382](#), [3392](#), [3398](#), [3406](#), [3416](#), [3424](#),
 [3430](#), [3438](#), [3448](#), [3986](#), [3992](#), [4148](#),
 [4160](#), [4489](#), [4501](#), [4514](#), [4524](#), [4540](#),
 [4558](#), [4718](#), [4738](#), [4753](#), [4771](#), [4791](#),
 [4801](#), [4838](#), [5156](#), [5371](#), [5379](#), [5392](#),
 [5399](#), [5406](#), [5413](#), [5802](#), [6167](#), [6192](#),
 [6214](#), [6295](#), [6438](#), [6440](#), [6450](#), [8488](#),
 [8491](#), [8497](#), [8717](#), [9630](#), [9670](#), [9676](#),
 [9826](#), [12931](#), [12939](#), [12989](#), [13023](#),
 [13032](#), [13046](#), [13064](#), [13072](#), [13082](#),
 [13096](#), [13104](#), [13114](#), [13162](#), [13168](#),
 [13174](#), [13180](#), [13186](#), [13192](#), [13198](#)
`\prg_set_conditional:Nnn` . [925](#), [925](#), [1894](#)
`\prg_set_conditional:Npnn`
 [32](#), [894](#), [894](#), [1095](#),
 [1107](#), [1123](#), [1135](#), [1894](#), [4486](#), [8713](#)
`\prg_set_eq_conditional:NN` [34](#)
`\prg_set_eq_conditional:NNn`
 [960](#), [960](#), [1894](#)
`\prg_set_eq_conditional_aux:NNNn` ...
 [961](#), [963](#), [1040](#), [1040](#)
`\prg_set_eq_conditional_aux:NNNw` ...
 [1040](#), [1041](#), [1042](#), [1050](#)
`\prg_set_map_functions:Nn` ... [2392](#), [2394](#)
`\prg_set_protected_conditional:Nnn` .
 [925](#), [944](#), [1894](#)
`\prg_set_protected_conditional:Npnn`
 [33](#), [894](#), [910](#), [1894](#)
`\prg_stepwise_aux:NNnnnn`
 [2245](#), [2247](#), [2253](#), [2262](#)
`\prg_stepwise_function:nnnN`
 [40](#), [2215](#), [2215](#), [2266](#)
`\prg_stepwise_function_decr:nnnN` ...
 [2215](#), [2222](#), [2235](#), [2240](#)
`\prg_stepwise_function_incr:nnnN` ...
 [2215](#), [2221](#), [2226](#), [2231](#)
`\prg_stepwise_inline:nnnn`
 [40](#), [2245](#), [2245](#), [13364](#), [13369](#)
`\prg_stepwise_variable:nnnNn`
 [40](#), [2245](#), [2251](#)
`\prg_variable_get_scope:N` [41](#), [2282](#), [2288](#)
`\prg_variable_get_scope_aux:w`
 [2282](#), [2290](#), [2293](#)
`\prg_variable_get_type:N` . [41](#), [2282](#), [2302](#)
`\prg_variable_get_type:w` [2282](#)
`\prg_variable_get_type_aux:w`
 [2304](#), [2307](#), [2311](#)
`\prop_clear:c` [6050](#), [6051](#), [7365](#)
`\prop_clear:N` . [115](#), [6050](#), [6050](#), [6055](#), [7863](#)
`\prop_clear_new:c` [6054](#), [6988](#), [6989](#), [8701](#)
`\prop_clear_new:N` . [115](#), [6054](#), [6054](#), [6056](#)
`\prop_del:cn` [6086](#)
`\prop_del:cV` [6086](#)
`\prop_del:Nn` [117](#), [6086](#),
 [6086](#), [6092](#), [6093](#), [7858](#), [7861](#), [7866](#)
`\prop_del:NV` [6086](#)
`\prop_del_aux:NNnnn` [6086](#), [6087](#), [6089](#), [6090](#)
`\prop_display:c` [6331](#), [6333](#)
`\prop_display:N` [6331](#), [6332](#)
`\prop_gclear:c` [6050](#), [6053](#), [6059](#)
`\prop_gclear:N` ... [115](#), [6050](#), [6052](#), [6058](#)
`\prop_gclear_new:c` [6054](#), [6059](#)
`\prop_gclear_new:N` [115](#), [6054](#), [6057](#)

\prop_gdel:cn	6086	\prop_gput:Noo	6127
\prop_gdel:cV	6086	\prop_gput:NVn	6127, 8097, 8110
\prop_gdel:Nn	117, 6086, 6088, 6094, 6095	\prop_gput:NVV	6127
\prop_gdel:NV	6086, 8214, 8227	\prop_gput_if_new:cn	6149
\prop_get:cn	6317	\prop_gput_if_new:Nnn	116, 6149, 6151, 6163
\prop_get:cnN	6096, 6205, 8841	\prop_gset_eq:cc	6060, 6067, 7132, 7134
\prop_get:cnNF	7108	\prop_gset_eq:cN	6060, 6066, 6990, 6992
\prop_get:coN	6205	\prop_gset_eq:Nc	6060, 6065
\prop_get:cVN	6096, 6205	\prop_gset_eq:NN	116, 6060, 6064
\prop_get:Nn	120, 6317, 6317, 6330	\prop_if_empty:c	6164
\prop_get:NnN	117, 6096, 6096, 6104, 6105, 6205, 6205, 7811, 7815, 7895, 7899	\prop_if_empty:N	6164, 6164
\prop_get:NnNF	6217, 6220	\prop_if_empty:NF	6175
\prop_get:NnNT	6216, 6219	\prop_if_empty:NT	6174
\prop_get:NnNTF	118, 6218, 6221	\prop_if_empty:NTF	118, 6173, 6253, 8236, 8256
\prop_get:NoN	6096, 6205	\prop_if_empty_p:N	6172
\prop_get:NVN	6096, 6205	\prop_if_eq:cc	6354, 6358
\prop_get_aux:Nnnn	6096, 6099, 6102	\prop_if_eq:cN	6354, 6356
\prop_get_aux_true:Nnnn	6205, 6208, 6211	\prop_if_eq:Nc	6354, 6357
\prop_get_gdel:NnN	6343, 6344	\prop_if_eq:NN	6354, 6355
\prop_get_Nn_aux:nwn	6317, 6319, 6324, 6328	\prop_if_in:cc	6346
\prop_gget:cnN	6335	\prop_if_in:cn	6176
\prop_gget:cVN	6335	\prop_if_in:cnTF	8831, 8834
\prop_gget:NnN	6335, 6336, 6340, 6341	\prop_if_in:co	6176
\prop_gget:NVN	6335	\prop_if_in:cV	6176
\prop_gget_aux:Nnnn	6335, 6337, 6338	\prop_if_in:Nn	6176, 6176
\prop_gpop:cnN	6106, 6279	\prop_if_in:NnF	6201, 6202, 6348, 8118, 8126
\prop_gpop:coN	6106	\prop_if_in:NnT	6199, 6200, 6347
\prop_gpop:NnN	117, 6106, 6112, 6125, 6126, 6285, 6344	\prop_if_in:NnTF	118, 6203, 6204, 6349, 8825
\prop_gpop:NnNF	6301	\prop_if_in:No	6176
\prop_gpop:NnNT	6300	\prop_if_in:NV	6176
\prop_gpop:NnNTF	120, 6302	\prop_if_in:NVT	8162, 8199
\prop_gpop:NoN	6106	\prop_if_in_aux:Nw	6176, 6186, 6189
\prop_gput:ccx	6351	\prop_if_in_aux:nwn	6176, 6178, 6183, 6187
\prop_gput:cn	6127	\prop_if_in_p:Nn	6197, 6198
\prop_gput:cno	6127	\prop_map_break	119
\prop_gput:cnV	6127	\prop_map_break:	6230, 6248, 6248, 6311
\prop_gput:cnx	6127	\prop_map_break:n	119, 6249, 6249
\prop_gput:con	6127	\prop_map_function:cc	6222
\prop_gput:coo	6127	\prop_map_function:cN	6222, 7967
\prop_gput:cVn	6127	\prop_map_function:Nc	6222, 6243
\prop_gput:cVV	6127	\prop_map_function:NN	119, 6222, 6222, 6235, 6236, 6265, 8244, 8264
\prop_gput:Nnn	116, 6127, 6128, 6145, 6147, 6352	\prop_map_function_aux:Nwn	6222, 6224, 6227, 6233
\prop_gput:Nno	6127	\prop_map_inline:cn	6238, 7436, 7455, 7524, 7526, 7546, 7548, 7611, 7665, 7667, 7671, 7673
\prop_gput:NnV	6127	\prop_map_inline:Nn	119, 6238, 6238, 6247, 7529, 7624, 7864, 7873
\prop_gput:Nnx	6127		
\prop_gput:Non	6127		

- \prop_map_tokens:cn [6303](#)
 - \prop_map_tokens:Nn [120](#), [6303](#), [6303](#), [6316](#)
 - \prop_map_tokens_aux:nwn
..... [6303](#), [6305](#), [6308](#), [6314](#)
 - \prop_new:c [6048](#), [6049](#)
 - \prop_new:N [115](#), [6048](#), [6048](#), [6055](#), [6058](#),
[6915](#), [6920](#), [7511](#), [7736](#), [7777](#), [7785](#),
[8056](#), [8057](#), [8697](#), [8698](#), [8920](#), [8942](#)
 - \prop_pop:cnN [6106](#), [6279](#)
 - \prop_pop:coN [6106](#)
 - \prop_pop:NnN
[117](#), [6106](#), [6106](#), [6123](#), [6124](#), [6279](#), [6279](#)
 - \prop_pop:NnNF [6298](#)
 - \prop_pop:NnNT [6297](#)
 - \prop_pop:NnNTF [118](#), [6299](#)
 - \prop_pop:NoN [6106](#)
 - \prop_pop_aux:NNNnnn [6106](#), [6109](#), [6115](#), [6118](#)
 - \prop_pop_aux_true:NNNnnn
..... [6279](#), [6282](#), [6288](#), [6291](#)
 - \prop_put:cnn [6127](#), [7186](#), [8852](#), [8854](#)
 - \prop_put:cno [6127](#)
 - \prop_put:cnV [6127](#)
 - \prop_put:cnx
... [6127](#), [7192](#), [7194](#), [7196](#), [7198](#),
[7203](#), [7208](#), [7213](#), [7220](#), [7227](#), [7460](#),
[7573](#), [7631](#), [7639](#), [7706](#), [7720](#), [7727](#)
 - \prop_put:con [6127](#)
 - \prop_put:coo [6127](#)
 - \prop_put:cVn [6127](#)
 - \prop_put:cVV [6127](#)
 - \prop_put:Nnn .. [116](#), [6127](#), [6127](#), [6141](#),
[6143](#), [6916](#)–[6919](#), [7737](#), [7739](#), [7741](#),
[7743](#), [7745](#), [7747](#), [7749](#), [7751](#), [7753](#),
[7755](#), [7757](#), [7759](#), [7761](#), [7763](#), [7765](#),
[7767](#), [7769](#), [7771](#), [8059](#)–[8062](#), [8856](#)
 - \prop_put:Nno [6127](#), [6922](#)–[6924](#), [6926](#)–[6931](#)
 - \prop_put:NnV [6127](#)
 - \prop_put:Nnx
.. [6127](#), [7554](#), [7556](#), [7559](#), [7561](#), [7567](#)
 - \prop_put:Non [6127](#)
 - \prop_put:Noo [6127](#)
 - \prop_put:NVn [6127](#)
 - \prop_put:NVV [6127](#)
 - \prop_put_aux:NNnn [6127](#)–[6129](#)
 - \prop_put_aux:NNnnnn .. [6127](#), [6131](#), [6133](#)
 - \prop_put_if_new:cnn [6149](#)
 - \prop_put_if_new:Nnn [116](#), [6149](#), [6149](#), [6162](#)
 - \prop_put_if_new_aux:NNnn [6150](#), [6152](#), [6153](#)
 - \prop_set_eq:cc [6060](#), [6063](#), [7125](#), [7127](#), [7395](#)
 - \prop_set_eq:cN .. [6060](#), [6062](#), [7118](#), [7120](#)
 - \prop_set_eq:Nc [6060](#), [6061](#), [7853](#)
 - \prop_set_eq:NN [116](#), [6060](#), [6060](#)
 - \prop_show:c [6251](#), [6333](#)
 - \prop_show:N .. [120](#), [6251](#), [6251](#), [6278](#), [6332](#)
 - \prop_show_aux:n [6251](#)
 - \prop_show_aux:nn [6265](#), [6270](#)
 - \prop_show_aux:w
..... [6251](#), [6267](#), [6277](#), [8246](#), [8266](#)
 - \prop_split:Nnn [121](#), [6080](#), [6080](#), [6131](#), [6337](#)
 - \prop_split:NnTF [121](#),
[6068](#), [6068](#), [6082](#), [6087](#), [6089](#), [6098](#),
[6108](#), [6114](#), [6155](#), [6207](#), [6281](#), [6287](#)
 - \prop_split_aux:nnnn ... [6068](#), [6074](#), [6078](#)
 - \prop_split_aux:NnTF ... [6068](#), [6069](#), [6070](#)
 - \prop_split_aux:w [6068](#), [6072](#), [6075](#), [6079](#)
 - \protect [235](#)
 - \protected [68](#),
[82](#), [98](#), [104](#), [126](#), [133](#), [141](#), [143](#), [147](#),
[152](#), [157](#), [213](#), [266](#), [273](#), [292](#), [325](#), [736](#)
 - \protected@edef [8371](#), [8643](#)
 - \ProvidesClass [154](#)
 - \ProvidesExplClass [6](#), [146](#), [152](#)
 - \ProvidesExplFile [6](#), [146](#), [157](#)
 - \ProvidesExplPackage
..... [6](#), [146](#), [147](#), [333](#), [781](#), [1532](#),
[1888](#), [2399](#), [2489](#), [3209](#), [3917](#), [4240](#),
[5042](#), [5579](#), [6042](#), [6363](#), [6907](#), [8007](#),
[8027](#), [8520](#), [9103](#), [9752](#), [9869](#), [13233](#)
 - \ProvidesFile [159](#)
 - \ProvidesPackage [47](#), [149](#)
- Q**
- \q [1076](#), [2026](#), [2031](#)
 - \q_mark [43](#), [1850](#), [1852](#), [1856](#),
[2404](#), [2405](#), [3367](#), [3369](#), [4439](#), [4447](#),
[4451](#), [4462](#), [4648](#), [4651](#), [4652](#), [4658](#),
[4662](#), [4664](#), [4666](#), [4692](#), [4693](#), [5625](#),
[5626](#), [5640](#), [5653](#), [5657](#), [5759](#), [5765](#),
[5778](#), [5877](#), [5884](#), [6073](#), [6075](#), [6076](#)
 - \q_nil [875](#), [878](#), [2317](#), [2321](#),
[2404](#), [2404](#), [2444](#), [2465](#), [3738](#), [3760](#),
[4500](#), [4512](#), [4513](#), [4650](#), [4654](#), [4671](#),
[4674](#), [4677](#), [4716](#), [4732](#), [4752](#), [5624](#),
[5628](#), [5635](#), [5703](#), [5712](#), [9129](#), [9160](#),
[9180](#), [9187](#), [9192](#), [13154](#), [13161](#),
[13167](#), [13173](#), [13179](#), [13185](#), [13191](#)
 - \q_no_value [43](#), [2055](#), [2404](#), [2406](#),
[2452](#), [2474](#), [6084](#), [6100](#), [6110](#), [6116](#),
[9129](#), [9137](#), [9142](#), [9159](#), [9165](#), [9396](#)

- `\q_prop` [120](#), [6046](#), [6046](#), [6047](#), [6073](#), [6074](#),
[6076](#), [6138](#), [6159](#), [6180](#), [6183](#), [6191](#),
[6225](#), [6227](#), [6306](#), [6308](#), [6321](#), [6324](#)
 - `\q_recursion_stop` [44](#), [877](#),
[880](#), [972](#), [1041](#), [1789](#), [2113](#), [2120](#),
[2133](#), [2143](#), [2155](#), [2166](#), [2408](#), [2409](#),
[4564](#), [4568](#), [4583](#), [4593](#), [4598](#), [4635](#),
[4636](#), [4639](#), [5641](#), [5818](#), [5857](#), [5877](#),
[5933](#), [6181](#), [6189](#), [6225](#), [6306](#), [6322](#)
 - `\q_recursion_tail`
..... [2408](#), [2408](#), [2412](#), [2418](#),
[2427](#), [2434](#), [4564](#), [4568](#), [4583](#), [4593](#),
[4598](#), [4635](#), [5641](#), [5818](#), [5857](#), [5877](#),
[5933](#), [6181](#), [6225](#), [6229](#), [6306](#), [6310](#)
 - `\q_stop` . [43](#), [876](#), [879](#), [1076](#), [1078](#), [1086](#),
[1088](#), [1315](#), [1319](#), [1811](#), [1853](#), [1856](#),
[2055](#), [2058](#), [2291](#), [2293](#), [2305](#), [2307](#),
[2311](#), [2317](#), [2321](#), [2383](#), [2404](#), [2407](#),
[2692](#), [2694](#), [2732](#), [2734](#), [2739](#), [2741](#),
[2749](#), [2752](#), [2760](#), [2763](#), [2771](#), [2774](#),
[2782](#), [2785](#), [2791](#), [2794](#), [2799](#), [2801](#),
[2807](#), [2810](#), [2827](#), [2830](#), [2833](#), [2855](#),
[3048](#), [3055](#), [3064](#), [3073](#), [3358](#), [3359](#),
[3367](#), [3371](#), [3379](#), [3387](#), [3395](#), [3403](#),
[3411](#), [3419](#), [3806](#), [3843](#), [4080](#), [4085](#),
[4447](#), [4462](#), [4656](#), [4677](#), [4687](#), [4688](#),
[4690](#), [4692](#), [4693](#), [4700](#), [4708](#), [4710](#),
[4716](#), [4732](#), [4752](#), [5035](#), [5037](#), [5166](#),
[5169](#), [5179](#), [5182](#), [5370](#), [5391](#), [5398](#),
[5479](#), [5481](#), [5486](#), [5630](#), [5635](#), [5693](#),
[5694](#), [5703](#), [5705](#), [5765](#), [5959](#), [5992](#),
[6073](#), [6076](#), [8380](#), [8471](#), [9076](#), [9078](#),
[9141](#), [9146](#), [9148](#), [9159](#), [9164](#), [9166](#),
[9189](#), [9192](#), [9260](#), [9263](#), [9269](#), [9278](#),
[9288](#), [9944](#), [9945](#), [9964](#), [9969](#), [9974](#),
[9980](#), [9985](#), [9991](#), [9997](#), [9999](#), [10000](#),
[10003](#), [10007](#), [10011](#), [10047](#), [10052](#),
[10269](#), [10271](#), [10286](#), [10288](#), [10289](#),
[10295](#), [10302](#), [10304](#), [10307](#), [10309](#),
[10310](#), [10312](#), [10314](#), [10316](#), [10318](#),
[10320](#), [10322](#), [10324](#), [10325](#), [10334](#),
[10336](#), [10346](#), [10348](#), [10359](#), [10366](#),
[10368–10370](#), [10372](#), [10374](#), [10376](#),
[10378](#), [10380](#), [10382](#), [10384](#), [10386](#),
[10395](#), [10401](#), [10403](#), [10413](#), [10415](#),
[10422](#), [10424–10426](#), [10432](#), [10437](#),
[10442](#), [10447](#), [10452](#), [10457](#), [10462](#),
[10467](#), [10477](#), [10482](#), [10483](#), [10494](#),
[10496](#), [10515](#), [10535](#), [11005](#), [11010](#),
[11122](#), [11175](#), [11352](#), [11357](#), [11364](#),
[11367](#), [13154](#), [13158](#), [13161](#), [13164](#),
[13167](#), [13170](#), [13173](#), [13176](#), [13179](#),
[13182](#), [13185](#), [13188](#), [13191](#), [13194](#)
 - `\q_tl_act_mark`
..... [95](#), [2484](#), [2484](#), [4855](#), [4858](#), [4875](#)
 - `\q_tl_act_stop` [95](#),
[2484](#), [2485](#), [4855](#), [4858](#), [4862](#), [4871](#),
[4873](#), [4879](#), [4883](#), [4886](#), [4890](#), [4893](#)
 - `\quark_if_nil:N` [2442](#), [2442](#)
 - `\quark_if_nil:n` [2462](#), [2462](#)
 - `\quark_if_nil:nF` [2483](#)
 - `\quark_if_nil:nT` [2324](#), [2328](#), [2482](#)
 - `\quark_if_nil:NTF` .. [44](#), [3741](#), [3763](#), [9184](#)
 - `\quark_if_nil:nTF` [44](#), [2332](#), [2341](#), [2350](#),
[2359](#), [2481](#), [5708](#), [13160](#), [13166](#),
[13172](#), [13178](#), [13184](#), [13190](#), [13196](#)
 - `\quark_if_nil:o` [2462](#)
 - `\quark_if_nil:oF` [9139](#)
 - `\quark_if_nil:V` [2462](#)
 - `\quark_if_nil_p:n` [2480](#)
 - `\quark_if_no_value:c` [2442](#)
 - `\quark_if_no_value:cF` [9616](#)
 - `\quark_if_no_value:N` [2450](#)
 - `\quark_if_no_value:n` [2462](#), [2471](#)
 - `\quark_if_no_value:N.` [2442](#)
 - `\quark_if_no_value:NF` [2460](#)
 - `\quark_if_no_value:NT` [2459](#), [13308](#)
 - `\quark_if_no_value:NTF`
[44](#), [2060](#), [2461](#), [7813](#), [7817](#), [7897](#), [7901](#)
 - `\quark_if_no_value:nTF` [44](#)
 - `\quark_if_no_value_p:N` [2458](#)
 - `\quark_if_recursion_tail_aux:w` .. [2424](#)
 - `\quark_if_recursion_tail_stop:N`
..... [44](#), [2410](#), [2410](#), [4604](#)
 - `\quark_if_recursion_tail_stop:n`
..... [45](#), [2424](#), [2424](#),
[2440](#), [4572](#), [5650](#), [5823](#), [5862](#), [5881](#)
 - `\quark_if_recursion_tail_stop:o` .. [2424](#)
 - `\quark_if_recursion_tail_stop_do:Nn`
..... [45](#), [2410](#), [2416](#)
 - `\quark_if_recursion_tail_stop_do:nn`
..... [45](#), [2424](#), [2431](#), [2441](#), [4638](#)
 - `\quark_if_recursion_tail_stop_do:on`
..... [2424](#)
 - `\quark_new:N`
[43](#), [2403](#), [2403–2409](#), [2484](#), [2485](#), [6046](#)
- R**
- `\R` [1837](#), [2724](#)
 - `\radical` [464](#)

<code>\raise</code>	604	<code>\scriptspace</code>	516	
<code>\read</code>	411	<code>\scriptstyle</code>	475	
<code>\readline</code>	686	<code>\scrollmode</code>	441	
<code>\relax</code>	3–6, 9, 13, 62, 70–80, 84–93, 96, 101, 131, 133, 141, 143, 229, 233, 249, 281–289, 446	<code>\seq_break</code>	104	
<code>\relpenalty</code>	507	<code>\seq_break:</code> ...	5204, 5236, <u>5241</u> , 5241, 5243, 5250, 5364, 5372, 5379, 5392, 5399, 5406, 5413, 5450, 5470, 5478	
<code>\RequirePackage</code>	57, 58	<code>\seq_break:n</code>	<u>105</u> , 5153, 5156, <u>5241</u> , 5242, 5244, 5458	
<code>\reverse_if:N</code> <u>22</u> , <u>785</u> , 790, 4014–4016, 4707		<code>\seq_break_point:n</code> ..	<u>105</u> , 5154, 5167, 5180, 5193, 5221, 5241, 5242, <u>5245</u> , 5245, 5257, 5292, 5303, 5373, 5380, 5393, 5400, 5407, 5414, 5452, 5471	
<code>\right</code>	505	<code>\seq_clear:c</code>	<u>5055</u> , 5056	
<code>\righthyphenmin</code>	561	<code>\seq_clear:N</code> ...	<u>96</u> , <u>5055</u> , 5055, 5100, 8798	
<code>\rightskip</code>	563	<code>\seq_clear_new:c</code>	<u>5059</u> , 5060	
<code>\romannumeral</code>	638	<code>\seq_clear_new:N</code>	<u>96</u> , <u>5059</u> , 5059	
<code>\rule</code>	7795, 7850	<code>\seq_concat:ccc</code>	<u>5071</u>	
S				
<code>\S</code>	2724	<code>\seq_concat:NNN</code> <u>97</u> , <u>5071</u> , 5071, 5075, 9802		
<code>\savecatcodetable</code>	763	<code>\seq_display:c</code>	<u>5572</u> , 5574	
<code>\savingshyphcodes</code>	725	<code>\seq_display:N</code>	<u>5572</u> , 5573	
<code>\savingvdiscards</code>	726	<code>\seq_gclear:c</code>	<u>5055</u> , 5058	
<code>\scan_align_safe_stop</code>	41	<code>\seq_gclear:N</code>	<u>96</u> , <u>5055</u> , 5057	
<code>\scan_align_safe_stop:</code> . <u>2281</u> , <u>2281</u> , 3879		<code>\seq_gclear_new:c</code>	<u>5059</u> , 5062	
<code>\scan_stop</code>	9	<code>\seq_gclear_new:N</code>	<u>96</u> , <u>5059</u> , 5061	
<code>\scan_stop:</code>	308, 322, <u>810</u> , 810, 1044, 1068, 1097, 1115, 1125, 1143, 1179, 1573, 1834–1842, 2283, 2297, 2628, 2705, 3057, 3066, 3075, 3108, 3110, 3112, 4123, 4134, 4139, 4164, 4169, 4172, 4198, 4209, 4214, 4219, 4233, 4234, 4349–4352, 4708, 6476, 6512, 6513, 7791, 7846, 8098, 8111, 9952–9954, 9994, 10005, 10013, 10018, 10054, 10055, 10071, 10079, 10118, 10127, 10143, 10152, 10215, 10710, 10722, 10723, 10855, 10859, 10871, 10875, 10888, 10892, 10934, 10995, 10999, 11006–11008, 11016, 11027, 11077, 11179, 11254, 11262, 11304, 11318, 11322, 11360, 11361, 11370, 11371, 11388, 11396, 11404, 11420, 11434, 11447, 11802, 12125, 12135, 12179, 12255–12258, 12279, 12480, 12506, 12543, 12551, 12559, 12641, 12643, 12645, 12680, 12688, 12892, 12894, 12896, 12898, 12900, 12914, 13310		<code>\seq_gconcat:ccc</code>	<u>5071</u>
<code>\scantokens</code>	684	<code>\seq_gconcat:NNN</code> <u>97</u> , <u>5071</u> , 5073, 5076, 9863		
<code>\scriptfont</code>	630	<code>\seq_get:cN</code>	5327, 5328	
<code>\scriptscriptfont</code>	631	<code>\seq_get:NN</code>	101, <u>5327</u> , 5327	
<code>\scriptscriptstyle</code>	476	<code>\seq_get_left:cN</code> .	<u>5163</u> , 5328, <u>5367</u> , 5570	
		<code>\seq_get_left:NN</code>	<u>98</u> , <u>5163</u> , 5163, 5171, 5327, <u>5367</u> , 5367, 5569	
		<code>\seq_get_left:NNF</code>	5383	
		<code>\seq_get_left:NNT</code>	5382	
		<code>\seq_get_left:NNTF</code>	102, 5384	
		<code>\seq_get_left_aux:NnwN</code> .	<u>5163</u> , 5166, 5169	
		<code>\seq_get_left_aux:Nw</code>	5370	
		<code>\seq_get_right:cN</code>	<u>5189</u> , <u>5367</u>	
		<code>\seq_get_right:NN</code>	98, <u>5189</u> , 5189, 5212, <u>5367</u> , 5375	
		<code>\seq_get_right:NNF</code>	5386	
		<code>\seq_get_right:NNT</code>	5385	
		<code>\seq_get_right:NNTF</code>	102, 5387	
		<code>\seq_get_right_aux:NN</code>	<u>5189</u> , 5192, 5195, 5378	
		<code>\seq_get_right_loop:nn</code>	<u>5189</u> , 5198, 5207, 5210, 5227	
		<code>\seq_gpop:cN</code>	<u>5327</u> , 5332	
		<code>\seq_gpop:NN</code> .	101, <u>5327</u> , 5331, 9842, 13307	
		<code>\seq_gpop_left:cN</code>	<u>5172</u> , 5332, <u>5388</u>	

\seq_gpop_left:NN	5420	\seq_gset_eq:Nc	5063, 5068
98, 5172, 5174, 5188, 5331, 5388, 5395		\seq_gset_eq:NN	97, 5063, 5067, 5097
\seq_gpop_left:NNF	5420	\seq_gset_from_clist:cc	5490
\seq_gpop_left:NNT	5419	\seq_gset_from_clist:cN	5490
\seq_gpop_left:NNTF	102, 5421	\seq_gset_from_clist:cn	5490
\seq_gpop_right:cn	5213, 5388	\seq_gset_from_clist:Nc	5490
\seq_gpop_right:NN	98, 5213, 5215, 5240, 5388, 5409	\seq_gset_from_clist:NN	104, 5490, 5500, 5514, 5515
\seq_gpop_right:NNF	5426	\seq_gset_from_clist:Nn	5490, 5505, 5516
\seq_gpop_right:NNT	5425	\seq_gset_reverse:N	104, 5517, 5521
\seq_gpop_right:NNTF	103, 5427	\seq_gset_split:Nnn	104, 5536, 5538
\seq_gpush:cn	5307, 5322	\seq_if_empty:c	5136, 5138
\seq_gpush:co	5307, 5325	\seq_if_empty:N	5136, 5136
\seq_gpush:cV	5307, 5323	\seq_if_empty:NTF	99, 5336, 6004
\seq_gpush:cv	5307, 5324	\seq_if_empty_break_return_false:N	5360, 5360, 5369, 5377, 5390, 5397, 5404, 5411
\seq_gpush:cx	5307, 5326	\seq_if_empty_err_break:N	104, 5165, 5178, 5191, 5219, 5246, 5246
\seq_gpush:Nn	102, 5307, 5317	\seq_if_in:cn	5140
\seq_gpush:No	5307, 5320	\seq_if_in:co	5140
\seq_gpush:Nv	5307, 5318, 9839	\seq_if_in:cV	5140
\seq_gpush:Nv	5307, 5319	\seq_if_in:cv	5140
\seq_gpush:Nx	5307, 5321, 13296	\seq_if_in:cx	5140
\seq_gput_left:cn	5085, 5322	\seq_if_in:Nn	5140, 5140
\seq_gput_left:co	5085, 5325	\seq_if_in:NnF	5103, 5159, 5160, 9847
\seq_gput_left:cV	5085, 5323	\seq_if_in:NnT	5157, 5158
\seq_gput_left:cv	5085, 5324	\seq_if_in:NnTF	99, 5161, 5162, 8803
\seq_gput_left:cx	5085, 5326	\seq_if_in:No	5140
\seq_gput_left:Nn	97, 5085, 5085, 5089, 5090, 5317	\seq_if_in:Nv	5140
\seq_gput_left:No	5085, 5320	\seq_if_in:Nv	5140
\seq_gput_left:Nv	5085, 5318	\seq_if_in:Nx	5140
\seq_gput_left:Nv	5085, 5319	\seq_if_in_aux:	5140, 5149, 5156
\seq_gput_left:Nx	5085, 5321	\seq_item:cn	5438
\seq_gput_right:cn	5085	\seq_item:n	104, 5046, 5046, 5078, 5080, 5086, 5088, 5128, 5145, 5169, 5182, 5225, 5269, 5274, 5279, 5285, 5510, 5525, 5526, 5528, 5534, 5567
\seq_gput_right:co	5085	\seq_item:Nn	103, 5438, 5438, 5461
\seq_gput_right:cV	5085	\seq_item_aux:nnn	5438, 5440, 5454, 5459
\seq_gput_right:cv	5085	\seq_length:c	5428
\seq_gput_right:cx	5085	\seq_length:N	103, 5428, 5428, 5437, 5445
\seq_gput_right:Nn	97, 5085, 5087, 5091, 5092	\seq_length_aux:n	5428, 5433, 5436
\seq_gput_right:No	5085	\seq_map_break	100
\seq_gput_right:Nv	5085, 9773	\seq_map_break:	5243, 5243, 5256, 9812
\seq_gput_right:Nv	5085	\seq_map_break:n	101, 5243, 5244
\seq_gput_right:Nx	5085, 9834	\seq_map_function:cn	5253
\seq_gremove_all:cn	5110	\seq_map_function:NN	4, 100, 5253, 5253, 5265, 5348, 5433, 5462
\seq_gremove_all:Nn	99, 5110, 5112, 5135		
\seq_gremove_duplicates:c	5094		
\seq_gremove_duplicates:N	99, 5094, 5096, 5109		
\seq_gset_eq:cc	5063, 5070		
\seq_gset_eq:cn	5063, 5069		

<code>\seq_map_function_aux:NNn</code>	<code>\seq_push:cv</code>	5314
..... 5253 , 5255 , 5259 , 5263	<code>\seq_push:cx</code>	5307 , 5316
<code>\seq_map_inline:cn</code>	<code>\seq_push:Nn</code>	102 , 5307 , 5307
<code>\seq_map_inline:Nn</code>	<code>\seq_push:No</code>	5307 , 5310
100 , 5101 , 5288 , 5288 , 5294 , 9806 , 9856	<code>\seq_push:Nv</code>	5307 , 5308
<code>\seq_map_variable:ccn</code>	<code>\seq_push:Nv</code>	5307 , 5309
<code>\seq_map_variable:cNn</code>	<code>\seq_push:Nx</code>	5307 , 5311
<code>\seq_map_variable:Ncn</code>	<code>\seq_push_item_def:n</code>	104 , 5116 ,
<code>\seq_map_variable:NNn</code>	5197 , 5225 , 5266 , 5266 , 5290 , 6007	
..... 100 , 5295 , 5295 , 5305 , 5306	<code>\seq_push_item_def:x</code> ...	5266 , 5271 , 5297
<code>\seq_mapthread_function:ccN</code>	<code>\seq_push_item_def_aux:</code>	
<code>\seq_mapthread_function:cNn</code> 5266 , 5268 , 5273 , 5276	
<code>\seq_mapthread_function:NcN</code>	<code>\seq_put_left:cn</code>	5077 , 5312
<code>\seq_mapthread_function:NNN</code>	<code>\seq_put_left:co</code>	5077 , 5315
..... 103 , 5464 , 5464 , 5488 , 5489	<code>\seq_put_left:cV</code>	5077 , 5313
<code>\seq_mapthread_function_aux:NN</code>	<code>\seq_put_left:cv</code>	5077 , 5314
..... 5464 , 5466 , 5473	<code>\seq_put_left:cx</code>	5077 , 5316
<code>\seq_mapthread_function_aux:Nnnwnn</code> .	<code>\seq_put_left:Nn</code>	
..... 5464 , 5475 , 5481 , 5486 97 , 5077 , 5077 , 5081 , 5082 , 5307	
<code>\seq_new:c</code>	<code>\seq_put_left:No</code>	5077 , 5310
<code>\seq_new:N</code> 4 , 96 , 5053 , 5053 , 5093 , 8790 ,	<code>\seq_put_left:Nv</code>	5077 , 5308
9767 , 9768 , 9777 , 9779 , 9782 , 13263	<code>\seq_put_left:Nx</code>	5077 , 5309
<code>\seq_pop:cN</code>	<code>\seq_put_right:cn</code>	5077 , 5311
<code>\seq_pop:NN</code>	<code>\seq_put_right:co</code>	5077
<code>\seq_pop_item_def</code>	<code>\seq_put_right:cV</code>	5077
<code>\seq_pop_item_def:</code>	<code>\seq_put_right:cv</code>	5077
5132 , 5203 ,	<code>\seq_put_right:cx</code>	5077
5235 , 5266 , 5282 , 5292 , 5303 , 6015	<code>\seq_put_right:Nn</code>	97 , 5077 ,
<code>\seq_pop_left:cN</code>	5079 , 5083 , 5084 , 5104 , 8806 , 9848	
5172 , 5330 , 5388	<code>\seq_put_right:No</code>	5077
<code>\seq_pop_left:NN</code>	<code>\seq_put_right:Nv</code>	5077
98 , 5172 , 5172 , 5187 , 5329 , 5388 , 5388	<code>\seq_put_right:Nv</code>	5077
<code>\seq_pop_left:NNF</code>	<code>\seq_put_right:Nx</code>	5077
5417	<code>\seq_remove_all:cn</code>	5110
<code>\seq_pop_left:NNT</code>	<code>\seq_remove_all:Nn</code>	
5416 99 , 5110 , 5110 , 5134 , 9851	
<code>\seq_pop_left:NNTF</code>	<code>\seq_remove_all_aux:NNn</code>	
102 , 5418 5110 , 5111 , 5113 , 5114	
<code>\seq_pop_left_aux:NNN</code>	<code>\seq_remove_duplicates:c</code>	5094
..... 5172 , 5173 , 5175 , 5176	<code>\seq_remove_duplicates:N</code>	
<code>\seq_pop_left_aux:NnnNNN</code> 99 , 5094 , 5094 , 5108 , 9854	
..... 5172 , 5179 , 5182 , 5391 , 5398	<code>\seq_remove_duplicates_aux:NN</code>	
<code>\seq_pop_right:cN</code> 5094 , 5095 , 5097 , 5098	
5213 , 5388	<code>\seq_reverse_aux:NN</code>	5520 , 5522 , 5523
<code>\seq_pop_right:NN</code>	<code>\seq_reverse_aux_item:w</code>	5526 , 5530
..... 98 , 5213 , 5213 , 5239 , 5388 , 5402	<code>\seq_set_eq:cc</code>	5063 , 5066
<code>\seq_pop_right:NNF</code>	<code>\seq_set_eq:cN</code>	5063 , 5065
5423	<code>\seq_set_eq:Nc</code>	5063 , 5064
<code>\seq_pop_right:NNT</code>		
5422		
<code>\seq_pop_right:NNTF</code>		
103 , 5424		
<code>\seq_pop_right_aux:NNN</code>		
..... 5213 , 5214 , 5216 , 5217		
<code>\seq_pop_right_aux_ii:NNN</code>		
..... 5213 , 5220 , 5223 , 5405 , 5412		
<code>\seq_push:cn</code>		
5307 , 5312		
<code>\seq_push:co</code>		
5307 , 5315		
<code>\seq_push:cV</code>		
5307 , 5313		

<code>\seq_set_eq:NN</code>	<code>\skip_gadd:Nn</code>
.... 96 , 5063 , 5063 , 5095 , 9800 , 9817	76 , 4133 , 4135 , 4137
<code>\seq_set_from_clist:cc</code>	<code>\skip_gset:cn</code>
5490	4122
<code>\seq_set_from_clist:cN</code>	<code>\skip_gset:Nn</code>
5490	76 , 4122 , 4124 , 4126
<code>\seq_set_from_clist:cn</code>	<code>\skip_gset_eq:cc</code>
5490	4127
<code>\seq_set_from_clist:Nc</code>	<code>\skip_gset_eq:cN</code>
5490	4127
<code>\seq_set_from_clist:NN</code>	<code>\skip_gset_eq:Nc</code>
103 , 5490 , 5490 , 5511 , 5512 , 9801 , 9862	4127
<code>\seq_set_from_clist:Nn</code> .	<code>\skip_gset_eq:NN</code> ...
5490 , 5495 , 5513	76 , 4127 , 4130 – 4132
<code>\seq_set_reverse:N</code>	<code>\skip_gsub:cn</code>
104 , 5517 , 5519	4133
<code>\seq_set_split:Nnn</code>	<code>\skip_gsub:Nn</code>
104 , 5536 , 5536	76 , 4133 , 4140 , 4142
<code>\seq_set_split_aux:NNnn</code>	<code>\skip_gzero:c</code>
5536 , 5537 , 5539 , 5540	4118
<code>\seq_set_split_aux_end:</code>	<code>\skip_gzero:N</code>
.. 5536 , 5549 , 5553 , 5560 , 5564 , 5566	75 , 4118 , 4119 , 4121
<code>\seq_set_split_aux_i:w</code>	<code>\skip_horizontal:c</code>
5536 , 5547 , 5554 , 5560	4167
<code>\seq_set_split_aux_ii:w</code> 5536 , 5562 , 5566	<code>\skip_horizontal:N</code>
<code>\seq_show:c</code> 80 , 4167 , 4167 , 4169 , 4173
5334 , 5574	<code>\skip_horizontal:n</code>
<code>\seq_show:N</code> ... 102 , 5334 , 5334 , 5359 , 5573	4167 , 4168 , 6855 , 6885
<code>\seq_show_aux:n</code>	<code>\skip_if_eq:nn</code>
5334 , 5348 , 5353	4143 , 4143
<code>\seq_show_aux:w</code>	<code>\skip_if_eq:nnTF</code>
5334 , 5350 , 5358	77
<code>\seq_tmp:w</code>	<code>\skip_if_infinite_glue:n</code> ...
5517 , 5525 , 5528	4153 , 4153
<code>\seq_top:cN</code>	<code>\skip_if_infinite_glue:nTF</code> ...
5568 , 5570	77 , 4226
<code>\seq_top:NN</code>	<code>\skip_new:c</code>
5568 , 5569	4110
<code>\seq_use:c</code>	<code>\skip_new:N</code>
5462	75 , 4110 , 4111 , 4117 , 4179 – 4183 , 10257
<code>\seq_use:N</code>	<code>\skip_set:cn</code>
103 , 5462 , 5462 , 5463	4122
<code>\seq_wrap_item:n</code>	<code>\skip_set:Nn</code> ...
.. 5490 , 5493 , 5498 , 5503 , 5508 , 5510	76 , 4122 , 4122 , 4124 , 4125
<code>\set@color</code>	<code>\skip_set_eq:cc</code>
8022	4127
<code>\setbox</code>	<code>\skip_set_eq:cN</code>
612	4127
<code>\setlanguage</code>	<code>\skip_set_eq:Nc</code>
370	4127
<code>\sfcode</code>	<code>\skip_set_eq:NN</code> ...
667	76 , 4127 , 4127 – 4129
<code>\sffamily</code>	<code>\skip_show:c</code>
7783	4175
<code>\shipout</code>	<code>\skip_show:N</code>
577	77 , 4175 , 4175 , 4176
<code>\show</code>	<code>\skip_split_finite_else_action:nnNN</code>
420 81 , 4224 , 4224
<code>\showbox</code>	<code>\skip_sub:cn</code>
422	4133
<code>\showboxbreadth</code>	<code>\skip_sub:Nn</code> ...
436	76 , 4133 , 4138 , 4140 , 4141
<code>\showboxdepth</code>	<code>\skip_use:c</code>
437	4165
<code>\showgroups</code>	<code>\skip_use:N</code> ...
697	77 , 4164 , 4165 , 4165 , 4166
<code>\showifs</code>	<code>\skip_vertical:c</code>
698	4167
<code>\showlists</code>	<code>\skip_vertical:N</code> 80 , 4167 , 4170 , 4172 , 4174
423	<code>\skip_vertical:n</code>
<code>\showthe</code>	4167 , 4171
421	<code>\skip_zero:c</code>
<code>\showtokens</code>	4118
685	<code>\skip_zero:N</code>
<code>\skewchar</code>	75 , 4118 , 4118 – 4120
634	<code>\skipdef</code>
<code>\skip</code>	357
658	<code>\space</code>
<code>\skip_add:cn</code>	49 , 205
4133	<code>\spacefactor</code>
<code>\skip_add:Nn</code> ... 76 , 4133 , 4133 , 4135 , 4136	576
<code>\skip_eval:n</code>	<code>\spaceskip</code>
77 , 4146 , 4163 , 4163	571
<code>\skip_gadd:cn</code>	<code>\span</code>
4133	384
	<code>\special</code>
	646
	<code>\splitbotmark</code>
	455
	<code>\splitbotmarks</code>
	681
	<code>\splitdiscards</code>
	728

<code>\tex_day:D</code>	651	<code>\tex_fi:D</code>	405, 789
<code>\tex_deadcycles:D</code>	585	<code>\tex_finalhyphendemerits:D</code>	554
<code>\tex_def:D</code>	350, 819–823	<code>\tex_firstmark:D</code>	452
<code>\tex_defaultthyphenchar:D</code>	635	<code>\tex_floatingpenalty:D</code>	599
<code>\tex_defaultskewchar:D</code>	636	<code>\tex_font:D</code>	365
<code>\tex_delcode:D</code>	666	<code>\tex_fontdimen:D</code>	632
<code>\tex_delimiter:D</code>	460	<code>\tex_fontname:D</code>	456
<code>\tex_delimiterfactor:D</code>	509	<code>\tex_futurelet:D</code>	361, 2881, 2883
<code>\tex_delimitershortfall:D</code>	508	<code>\tex_gdef:D</code>	352, 837
<code>\tex_dimen:D</code>	657	<code>\tex_global:D</code>	336, 341, 343, 367, 816, 816, 1284, 1289, 1919, 2883, 3306, 3317, 3323, 3331, 3333, 3343, 3345, 3352, 3933, 3938, 3944, 3950, 3954, 3961, 3966, 4119, 4124, 4130, 4135, 4140, 4194, 4199, 4205, 4210, 4215, 5001, 6398, 6404, 6458, 6478, 6484, 6490, 6518, 6524, 6530, 6536, 8083, 8215, 8228, 8502, 8506, 13272
<code>\tex_dimendef:D</code>	356, 2767	<code>\tex_globaldefs:D</code>	371
<code>\tex_discretionary:D</code>	520	<code>\tex_halign:D</code>	378
<code>\tex_displayindent:D</code>	485	<code>\tex_hangafter:D</code>	556
<code>\tex_displaylimits:D</code>	495	<code>\tex_hangingindent:D</code>	557
<code>\tex_displaystyle:D</code>	473	<code>\tex_hbadness:D</code>	618
<code>\tex_displaywidowpenalty:D</code>	484	<code>\tex_hbox:D</code>	613, 6476, 6477, 6482, 6488, 6502, 6503
<code>\tex_displaywidth:D</code>	486	<code>\tex_hfil:D</code>	521
<code>\tex_divide:D</code>	363, 10073, 10120, 10145, 10714, 10982, 11173, 11238, 11282, 11327, 11330, 11332, 11334, 11398, 11440, 12553, 12603–12605	<code>\tex_hfill:D</code>	523
<code>\tex_doublehyphendemerits:D</code>	553	<code>\tex_hfilneg:D</code>	522
<code>\tex_dp:D</code>	664, 6408	<code>\tex_hfuzz:D</code>	620
<code>\tex_dump:D</code>	647	<code>\tex_hoffset:D</code>	595
<code>\tex_edef:D</code>	351, 824	<code>\tex_holdinginserts:D</code>	598
<code>\tex_else:D</code>	404, 788	<code>\tex_hruler:D</code>	534
<code>\tex_emergencystretch:D</code>	568	<code>\tex_hsize:D</code>	559, 7019, 7062
<code>\tex_end:D</code>	442, 766, 1194, 8733, 8875	<code>\tex_hskip:D</code>	524, 4167
<code>\tex_endcsname:D</code>	444, 807	<code>\tex_hss:D</code>	525, 6505, 6507, 6839
<code>\tex_endgroup:D</code>	377, 764, 812	<code>\tex_ht:D</code>	663, 6407
<code>\tex_endinput:D</code>	416, 8744	<code>\tex_hyphen:D</code>	348, 769
<code>\tex_endlinechar:D</code>	307, 308, 322, 458, 4367, 4393, 4406	<code>\tex_hyphenation:D</code>	649
<code>\tex_eqno:D</code>	478	<code>\tex_hyphenchar:D</code>	633
<code>\tex_errhelp:D</code>	424, 8651	<code>\tex_hyphenpenalty:D</code>	551
<code>\tex_errmessage:D</code>	418, 1186, 8671	<code>\tex_if:D</code>	387, 791, 792
<code>\tex_errorcontextlines:D</code>	425, 8689	<code>\tex_ifcase:D</code>	388, 3218
<code>\tex_errorstopmode:D</code>	439	<code>\tex_ifcat:D</code>	389, 793
<code>\tex_escapechar:D</code>	457, 8320, 8356, 8362	<code>\tex_ifdim:D</code>	392, 3921
<code>\tex_everycr:D</code>	386	<code>\tex_ifeof:D</code>	393, 8031
<code>\tex_everydisplay:D</code>	487, 767	<code>\tex_iffalse:D</code>	398, 786
<code>\tex_everyhbox:D</code>	626	<code>\tex_ifhbox:D</code>	394, 6434
<code>\tex_everyjob:D</code>	655, 4816, 4818, 9758, 9760, 9770, 9772	<code>\tex_ifhmode:D</code>	400, 796
<code>\tex_everymath:D</code>	511, 768	<code>\tex_ifinner:D</code>	403, 798
<code>\tex_everypar:D</code>	574	<code>\tex_ifmmode:D</code>	401, 795
<code>\tex_everyvbox:D</code>	627		
<code>\tex_exhyphenpenalty:D</code>	550		
<code>\tex_expandafter:D</code>	374, 801		
<code>\tex_fam:D</code>	366		

<code>\tex_ifnum:D</code>	390, 813, 3216	<code>\tex_mathchardef:D</code>	359, 1179, 3306, 13272
<code>\tex_ifodd:D</code> ..	391, 1211, 1892, 1893, 3217	<code>\tex_mathchoice:D</code>	459
<code>\tex_iftrue:D</code>	399, 785	<code>\tex_mathclose:D</code>	492
<code>\tex_ifvbox:D</code>	395, 6435	<code>\tex_mathcode:D</code>	670, 2564, 2566, 2568, 3115
<code>\tex_ifvmode:D</code>	402, 797	<code>\tex_mathinner:D</code>	493
<code>\tex_ifvoid:D</code>	396, 6436	<code>\tex_mathop:D</code>	494
<code>\tex_ifx:D</code>	397, 794	<code>\tex_mathopen:D</code>	498
<code>\tex_ignorespaces:D</code>	445	<code>\tex_mathord:D</code>	499
<code>\tex_immediate:D</code>	407, 1181, 1183, 8111, 8289	<code>\tex_mathpunct:D</code>	500
<code>\tex_indent:D</code>	541	<code>\tex_mathrel:D</code>	501
<code>\tex_input:D</code>	415, 770, 9841	<code>\tex_mathsurround:D</code>	512
<code>\tex_inputlineno:D</code> .	417, 1201, 1829, 8590	<code>\tex_maxdeadcycles:D</code>	582
<code>\tex_insert:D</code>	597	<code>\tex_maxdepth:D</code>	583
<code>\tex_insertpenalties:D</code>	600	<code>\tex_meaning:D</code>	642, 804, 808
<code>\tex_interlinepenalty:D</code>	579	<code>\tex_medmuskip:D</code>	513
<code>\tex_italic_correction:D</code>	771	<code>\tex_message:D</code>	419
<code>\tex_italiccor:D</code>	347	<code>\tex_mkern:D</code>	466
<code>\tex_jobname:D</code>	654, 4826, 9761	<code>\tex_month:D</code>	652
<code>\tex_kern:D</code>	532, 6607, 6837, 7357, 7362, 7428, 7429, 7536, 7537, 7940, 7941	<code>\tex_moveleft:D</code>	602, 6427
<code>\tex_language:D</code>	449	<code>\tex_moveright:D</code>	603, 6429
<code>\tex_lastbox:D</code>	606, 6456, 6903	<code>\tex_mskip:D</code>	463
<code>\tex_lastkern:D</code>	539	<code>\tex_multiply:D</code>	364, 10912, 11119, 11333, 11349, 12327, 12916
<code>\tex_lastpenalty:D</code>	645	<code>\tex_muskip:D</code>	660
<code>\tex_lastskip:D</code>	540	<code>\tex_muskipdef:D</code>	358
<code>\tex_lccode:D</code>	668, 1068, 1834, 1835, 2283, 2297, 2570, 2572, 2574, 3116, 4349, 4350	<code>\tex_newlinechar:D</code> .	414, 4368, 4394, 4407
<code>\tex_leaders:D</code>	536	<code>\tex_noalign:D</code>	382
<code>\tex_left:D</code>	504	<code>\tex_noboundary:D</code>	517
<code>\tex_lefthyphenmin:D</code>	560	<code>\tex_noexpand:D</code>	375, 802
<code>\tex_leftskip:D</code>	562	<code>\tex_noindent:D</code>	543
<code>\tex_leqno:D</code>	479	<code>\tex_nolimits:D</code>	497
<code>\tex_let:D</code>	337, 341, 343, 349, 766–776, 785–818, 823, 824, 837, 838, 1168, 1277, 1528, 1892, 1893	<code>\tex_nonscript:D</code>	477
<code>\tex_limits:D</code>	496	<code>\tex_nonstopmode:D</code>	440
<code>\tex_linepenalty:D</code>	552	<code>\tex_nulldelimiterspace:D</code>	510
<code>\tex_lineskip:D</code>	546	<code>\tex_nullfont:D</code>	628, 2840
<code>\tex_lineskiplimit:D</code>	547	<code>\tex_number:D</code>	637, 3213
<code>\tex_long:D</code>	368, 816, 817, 819, 826, 828, 834, 836, 840, 842, 848, 850	<code>\tex_omit:D</code>	383
<code>\tex_looseness:D</code>	564	<code>\tex_openin:D</code>	409, 8098
<code>\tex_lower:D</code>	601, 6433	<code>\tex_openout:D</code>	410, 8111
<code>\tex_lowercase:D</code>	640, 1070, 1843, 4353, 4413	<code>\tex_or:D</code>	406, 787
<code>\tex_mag:D</code>	448	<code>\tex_outer:D</code>	369
<code>\tex_mark:D</code>	450	<code>\tex_output:D</code>	584
<code>\tex_mathaccent:D</code>	461	<code>\tex_outputpenalty:D</code>	594
<code>\tex_mathbin:D</code>	491	<code>\tex_over:D</code>	471
<code>\tex_mathchar:D</code>	462	<code>\tex_overfullrule:D</code>	622
		<code>\tex_overline:D</code>	502
		<code>\tex_overwithdelims:D</code>	472
		<code>\tex_pagedepth:D</code>	586
		<code>\tex_pagefilllstretch:D</code>	590
		<code>\tex_pagefillstretch:D</code>	589

<code>\tex_pagefilstretch:D</code>	588	<code>\tex_showboxbreadth:D</code>	436
<code>\tex_pagegoal:D</code>	592	<code>\tex_showboxdepth:D</code>	437
<code>\tex_pageshrink:D</code>	591	<code>\tex_showlists:D</code>	423
<code>\tex_pagestretch:D</code>	587	<code>\tex_showthe:D</code>	
<code>\tex_pagetotal:D</code>	593		421, 1437, 2498, 2568, 2574, 2580,
<code>\tex_par:D</code>	542, 8014		2586, 3123, 3126, 3129, 3132, 3135
<code>\tex_parfillskip:D</code>	573	<code>\tex_skewchar:D</code>	634
<code>\tex_parindent:D</code>	566	<code>\tex_skip:D</code>	658
<code>\tex_parshape:D</code>	558	<code>\tex_skipdef:D</code>	357, 2756
<code>\tex_parskip:D</code>	565	<code>\tex_space:D</code>	346
<code>\tex_patterns:D</code>	648	<code>\tex_spacefactor:D</code>	576
<code>\tex_pausing:D</code>	435	<code>\tex_spaceskip:D</code>	571
<code>\tex_penalty:D</code>	643	<code>\tex_span:D</code>	384
<code>\tex_postdisplaypenalty:D</code>	490	<code>\tex_special:D</code>	646
<code>\tex_predisplayskip:D</code>	489	<code>\tex_splitbotmark:D</code>	455
<code>\tex_predisplaysize:D</code>	488	<code>\tex_splitfirstmark:D</code>	454
<code>\tex_pretolerance:D</code>	569	<code>\tex_splitmaxdepth:D</code>	624
<code>\tex_prevdepth:D</code>	616	<code>\tex_splittopskip:D</code>	625
<code>\tex_prevgraf:D</code>	575	<code>\tex_string:D</code>	639, 805
<code>\tex_protected:D</code> ..	816, 818, 825, 827, 829–832, 834, 836, 844, 846, 848, 850	<code>\tex_tabskip:D</code>	385
<code>\tex_radical:D</code>	464	<code>\tex_textfont:D</code>	629
<code>\tex_raise:D</code>	604, 6431	<code>\tex_textstyle:D</code>	474
<code>\tex_read:D</code>	411, 8500, 8502	<code>\tex_the:D</code>	
<code>\tex_relax:D</code>	446, 810, 3215, 3923		. 308, 447, 1201, 1579, 1583, 1829,
<code>\tex_relpemalty:D</code>	507		2496, 2566, 2572, 2578, 2584, 3121,
<code>\tex_right:D</code>	505		3124, 3127, 3130, 3133, 3355, 4092,
<code>\tex_righthyphenmin:D</code>	561		4165, 4220, 4818, 9760, 9772, 13296
<code>\tex_rightskip:D</code>	563	<code>\tex_thickmuskip:D</code>	515
<code>\tex_romannumeral:D</code> ..	638, 814, 1549, 1561, 1567, 1609, 1613, 1618, 1624, 1630, 1636, 1648, 1653, 1655, 1662, 1716, 1723, 1728, 1731, 1733, 1736, 1741, 1747, 1759, 1769, 1773, 1778, 2116, 2129, 2142, 2154, 2165, 4855, 4907, 4926, 4966, 4968, 4988, 9069	<code>\tex_thinmuskip:D</code>	514
<code>\tex_scriptfont:D</code>	630	<code>\tex_time:D</code>	650
<code>\tex_scriptscriptfont:D</code>	631	<code>\tex_toks:D</code>	659
<code>\tex_scriptscriptstyle:D</code>	476	<code>\tex_toksdef:D</code>	360, 2778
<code>\tex_scriptspace:D</code>	516	<code>\tex_tolerance:D</code>	570
<code>\tex_scriptstyle:D</code>	475	<code>\tex_topmark:D</code>	451
<code>\tex_scrollmode:D</code>	441	<code>\tex_topskip:D</code>	581
<code>\tex_setbox:D</code>		<code>\tex_tracingcommands:D</code>	426
	612, 6396, 6402, 6456, 6477, 6482, 6488, 6517, 6522, 6528, 6534, 6552	<code>\tex_tracinglostchars:D</code>	427
<code>\tex_setlanguage:D</code>	370	<code>\tex_tracingmacros:D</code>	428
<code>\tex_sfcode:D</code> ..	667, 2582, 2584, 2586, 3118	<code>\tex_tracingonline:D</code>	429
<code>\tex_shipout:D</code>	577	<code>\tex_tracingoutput:D</code>	430
<code>\tex_show:D</code>	420, 809	<code>\tex_tracingpages:D</code>	431
<code>\tex_showbox:D</code>	422, 6474	<code>\tex_tracingparagraphs:D</code>	432
		<code>\tex_tracingrestores:D</code>	433
		<code>\tex_tracingstats:D</code>	434
		<code>\tex_uccode:D</code> ..	669, 2576, 2578, 2580, 3117
		<code>\tex_uchyph:D</code>	567
		<code>\tex_undefined:D</code>	336, 343
		<code>\tex_underline:D</code>	503, 772
		<code>\tex_unhbox:D</code>	608, 6509
		<code>\tex_unhcopy:D</code>	609, 6508

<code>\tex_unkern:D</code>	533	<code>\tl_act_case_space:n</code> ...	4965 , 4974 , 4976
<code>\tex_unpenalty:D</code>	644	<code>\tl_act_end:w</code>	4855
<code>\tex_unskip:D</code>	531	<code>\tl_act_end:wn</code>	4876 , 4882
<code>\tex_unvbox:D</code>	610, 6548	<code>\tl_act_group:nwnNNN</code> ...	4855 , 4868 , 4883
<code>\tex_unvcopy:D</code>	611, 6547	<code>\tl_act_group_recurse:Nnn</code>	4855 , 4900 , 4920
<code>\tex_uppercase:D</code>	641, 4414	<code>\tl_act_length_group:nn</code>	4939 , 4945 , 4953
<code>\tex_vadjust:D</code>	544	<code>\tl_act_length_normal:nN</code>	4939 , 4944 , 4951
<code>\tex_valign:D</code>	379	<code>\tl_act_length_space:n</code> .	4939 , 4946 , 4952
<code>\tex_vbadness:D</code>	619	<code>\tl_act_loop:w</code>	
<code>\tex_vbox:D</code> 614 , 6512 , 6515–6517 , 6528 , 6534		..	4855 , 4858 , 4862 , 4879 , 4886 , 4893
<code>\tex_vcenter:D</code>	465	<code>\tl_act_normal:NwnNNN</code> ..	4855 , 4865 , 4873
<code>\tex_vfil:D</code>	526	<code>\tl_act_output:n</code>	
<code>\tex_vfill:D</code>	528	4855 , 4896 , 4976 , 4979 , 4987
<code>\tex_vfilneg:D</code>	527	<code>\tl_act_result:n</code> ..	4860 , 4882 , 4896–4899
<code>\tex_vfuzz:D</code>	621	<code>\tl_act_reverse_group:nn</code>	4905 , 4910 , 4918
<code>\tex_voffset:D</code>	596	<code>\tl_act_reverse_group_preserve:nn</code> ..	
<code>\tex_vrule:D</code>	535, 7791 , 7846	4929 , 4933
<code>\tex_vsize:D</code>	578	<code>\tl_act_reverse_normal:nN</code>	
<code>\tex_vskip:D</code>	529, 4170	4905 , 4909 , 4916 , 4928
<code>\tex_vsplit:D</code>	607, 6552	<code>\tl_act_reverse_output:n</code>	
<code>\tex_vss:D</code>	530	..	4855 , 4898 , 4915 , 4917 , 4921 , 4934
<code>\tex_vtop:D</code>	615, 6513 , 6522	<code>\tl_act_reverse_space:n</code>	
<code>\tex_wd:D</code>	662, 6409	4905 , 4911 , 4914 , 4930
<code>\tex_widowpenalty:D</code>	549	<code>\tl_act_space:wwnNNN</code> ...	4855 , 4869 , 4890
<code>\tex_write:D</code>	412 , 1181 , 1183 , 8284	<code>\tl_clear:c</code>	4262 , 5056 , 5588
<code>\tex_xdef:D</code>	353, 838	<code>\tl_clear:N</code>	82 , 4262 , 4262 , 4266 , 4269 , 4370 , 5055 , 5587 , 8351 , 8353 , 8354 , 8444 , 9123 , 9127 , 9796 , 10700 , 10978 , 10996 , 11231 , 11255 , 11275 , 11305 , 11319
<code>\tex_xleaders:D</code>	538	<code>\tl_clear_new:c</code> 4268 , 5060 , 5592 , 9395 , 9397	
<code>\tex_xspaceskip:D</code>	572	<code>\tl_clear_new:N</code>	
<code>\tex_year:D</code>	653	83 , 4268 , 4268 , 4272 , 5059 , 5591
<code>\textasteriskcentered</code>	3903, 3909	<code>\tl_const:cn</code>	
<code>\textbardbl</code>	3908	4250 , 12032–12071 , 12378–12389
<code>\textdagger</code>	3904, 3910	<code>\tl_const:cx</code>	4250 , 8325 , 12468
<code>\textdaggerdbl</code>	3905, 3911	<code>\tl_const:Nn</code>	82 , 2403 , 2605 , 4250 , 4250 , 4260 , 4356 , 4827 , 4828 , 4955 , 4960 , 6047 , 8036 , 8317 , 8525 , 8526 , 8556 , 8561 , 8563 , 8565 , 8567 , 8569 , 8574 , 8575 , 8582 , 8600 , 9049 , 9051 , 9214–9218 , 9890–9894
<code>\textfont</code>	629	<code>\tl_const:Nx</code> .	4250 , 4255 , 4261 , 4826 , 8321
<code>\textparagraph</code>	3907	<code>\tl_elt_count:c</code>	5025 , 5030
<code>\textsection</code>	3906	<code>\tl_elt_count:N</code>	5025 , 5029
<code>\textstyle</code>	474	<code>\tl_elt_count:n</code>	5025 , 5026
<code>\texttt</code>	8616	<code>\tl_elt_count:o</code>	5025 , 5028
<code>\TeXETstate</code>	729	<code>\tl_elt_count:V</code>	5025 , 5027
<code>\the</code>	70–79, 447	<code>\tl_expandable_lowercase:n</code>	95 , 4965 , 4967
<code>\thickmuskip</code>	515	<code>\tl_expandable_uppercase:n</code>	95 , 4965 , 4965
<code>\thinmuskip</code>	514		
<code>\time</code>	650		
<code>\tiny</code>	7783		
<code>\tl_act:NNNnn</code>	4855 , 4855		
<code>\tl_act_aux:NNNnn</code>	4855 , 4855 , 4856 , 4908 , 4927 , 4943 , 4971		
<code>\tl_act_case_aux:nn</code>	4966 , 4968 , 4969 , 4988		
<code>\tl_act_case_group:nn</code> ..	4965 , 4973 , 4985		
<code>\tl_act_case_normal:nN</code> .	4965 , 4972 , 4977		

\tl_gclear:c	4262, 5058, 5590	\tl_gset:Nn	83, 4282, 4288,
\tl_gclear:N	83, 4262, 4264, 4267, 4271, 5057, 5589		4297, 4299, 4362, 4388, 4994, 5175,
\tl_gclear_new:c	4268, 5062, 5594		5398, 6089, 6115, 6288, 6339, 9840,
\tl_gclear_new:N	83, 4268, 4270, 4273, 5061, 5593		10175, 10629, 10664, 10745, 10770,
\tl_gput_left:cn	4300		10796, 10899, 10917, 11030, 11582,
\tl_gput_left:co	4300		11679, 11880, 12073, 12391, 12725
\tl_gput_left:cV	4300	\tl_gset:No	4282, 4290
\tl_gput_left:cx	4300	\tl_gset:Nv	4282
\tl_gput_left:Nn	84, 4300, 4308, 4320, 5086	\tl_gset:Nx	4282, 4292, 4298, 4419,
\tl_gput_left:No	4300, 4312, 4322		4423, 4684, 5074, 5113, 5216, 5412,
\tl_gput_left:Nv	4300, 4310, 4321		5502, 5507, 5522, 5539, 5606, 5663,
\tl_gput_left:Nx	4300, 4314, 4323, 5669		5753, 6001, 6128, 9761, 10211, 12375
\tl_gput_right:cn	4324	\tl_gset_eq:cc	4274, 4281, 5070, 5602, 6067, 10265
\tl_gput_right:co	4324	\tl_gset_eq:cN	4274, 4279, 5069, 5601, 6066, 10263
\tl_gput_right:cV	4324	\tl_gset_eq:Nc	4274, 4280, 5068, 5600, 6065, 10264
\tl_gput_right:cx	4324	\tl_gset_eq:NN	83,
\tl_gput_right:Nn	84, 4324, 4332, 4344, 5088		4265, 4274, 4278, 5067, 5599, 5669,
\tl_gput_right:No	4324, 4336, 4346		5687, 6064, 9765, 10159, 10171, 10262
\tl_gput_right:Nv	4324, 4334, 4345	\tl_gset_rescan:cnn	4359
\tl_gput_right:Nx	4324, 4338, 4347, 5687, 6152	\tl_gset_rescan:cno	4359
\tl_gremove_all:cn	4470, 5023	\tl_gset_rescan:cnx	4385
\tl_gremove_all:Nn	85, 4470, 4472, 4475, 5022	\tl_gset_rescan:Nnn	86, 4359, 4361, 4383, 4384
\tl_gremove_all_in:cn	5015, 5023	\tl_gset_rescan:Nno	4359
\tl_gremove_all_in:Nn	5015, 5022	\tl_gset_rescan:Nnx	4385, 4387, 4401
\tl_gremove_in:cn	5015, 5019	\tl_gtrim_spaces:c	4642
\tl_gremove_in:Nn	5015, 5018	\tl_gtrim_spaces:N	91, 4642, 4683, 4686
\tl_gremove_once:cn	4464, 5019	\tl_head:f	4687
\tl_gremove_once:Nn	85, 4464, 4466, 4469, 5018	\tl_head:n	91, 4687, 4689, 4694, 5033
\tl_greplace_all:cnn	4416, 5013	\tl_head:V	4687
\tl_greplace_all:Nnn	84, 4416, 4422, 4427, 4473, 5012	\tl_head:v	4687
\tl_greplace_all_in:cnn	5005, 5013	\tl_head:w	91, 4687, 4687, 4690, 4703,
\tl_greplace_all_in:Nnn	5005, 5012		4716, 4732, 4752, 5034, 9969, 9980
\tl_greplace_in:cnn	5005, 5009	\tl_head_i:n	5032, 5033
\tl_greplace_in:Nnn	5005, 5008	\tl_head_i:w	5032, 5034
\tl_greplace_once:cnn	4416, 5009	\tl_head_iii:f	5032
\tl_greplace_once:Nnn	84, 4416, 4418, 4425, 4467, 5008	\tl_head_iii:n	5032, 5035, 5036
\tl_gset:cf	4282	\tl_head_iii:w	5032, 5035, 5037
\tl_gset:cn	4282	\tl_if_blank:n	4476, 4476
\tl_gset:co	4282	\tl_if_blank:nF	3802, 4480, 4484, 5936
\tl_gset:cx	4282, 11649, 11735, 12016	\tl_if_blank:nT	4479, 4483
\tl_gset:Nc	4999, 5000	\tl_if_blank:nTF	86, 4481, 4485, 5980
\tl_gset:Nf	4282, 5700	\tl_if_blank:o	4476
		\tl_if_blank:oTF	9136
		\tl_if_blank:V	4476
		\tl_if_blank_p:n	4478, 4482

<code>\tl_if_blank_p_aux:NNw</code>	4476	<code>\tl_if_head_N_type:nTF</code>	
<code>\tl_if_empty:c</code>	4486 , 5138 , 5783	93 , 4715 , 4731 , 4745 , 4851 , 4864
<code>\tl_if_empty:N</code>	4486 , 4486 , 5136 , 5782	<code>\tl_if_head_space:n</code>	4794 , 4794
<code>\tl_if_empty:n</code>	4498 , 4498	<code>\tl_if_head_space:nTF</code>	93
<code>\tl_if_empty:NF</code>	4496 , 5676 , 9831	<code>\tl_if_head_space_aux:w</code>	4794 , 4799 , 4806
<code>\tl_if_empty:nF</code>	2991 , 4509 , 5882	<code>\tl_if_in:cn</code>	4548
<code>\tl_if_empty:NT</code>	4495	<code>\tl_if_in:Nn</code>	4548
<code>\tl_if_empty:nT</code>	4508	<code>\tl_if_in:nn</code>	4554 , 4554
<code>\tl_if_empty:NTF</code> ...	87 , 4497 , 9181 , 9824	<code>\tl_if_in:NnF</code>	4549 , 4552
<code>\tl_if_empty:nTF</code>		<code>\tl_if_in:nnF</code>	4549 , 4561
.....	87 , 2735 , 2742 , 2753 , 2764 ,	<code>\tl_if_in:NnT</code>	4548 , 4551
	2775 , 2786 , 2795 , 2802 , 2811 , 3844 ,	<code>\tl_if_in:nnT</code>	4548 , 4560
	4430 , 4507 , 5542 , 5651 , 8607 , 9289	<code>\tl_if_in:NnTF</code>	87 , 4550 , 4553
<code>\tl_if_empty:o</code>	4510 , 4519	<code>\tl_if_in:nnTF</code>	
<code>\tl_if_empty:oTF</code> .	2832 , 4557 , 5800 , 6010	87 , 4550 , 4562 , 7443 , 9259 , 9266
<code>\tl_if_empty:V</code>	4498	<code>\tl_if_in:no</code>	4554
<code>\tl_if_empty_p:N</code>	4494	<code>\tl_if_in:on</code>	4554
<code>\tl_if_empty_p:n</code>	4506	<code>\tl_if_in:Vn</code>	4554
<code>\tl_if_empty_return:o</code>		<code>\tl_if_single:N</code>	4843
.....	4477 , 4510 , 4510 , 4520	<code>\tl_if_single:n</code>	4847 , 4847
<code>\tl_if_eq:cc</code>	4521 , 5787 , 6358	<code>\tl_if_single:NF</code>	4845
<code>\tl_if_eq:ccTF</code>	9628	<code>\tl_if_single:nF</code>	4845
<code>\tl_if_eq:cN</code>	4521 , 5786 , 6356	<code>\tl_if_single:NT</code>	4844
<code>\tl_if_eq:Nc</code>	4521 , 5785 , 6357	<code>\tl_if_single:nT</code>	4844
<code>\tl_if_eq:NN</code>	4521 , 4521 , 5784 , 6355	<code>\tl_if_single:NTF</code>	87 , 4846
<code>\tl_if_eq:nn</code>	4533 , 4533	<code>\tl_if_single:nTF</code>	88 , 4846
<code>\tl_if_eq:NNF</code>	4532	<code>\tl_if_single_p:N</code>	4843
<code>\tl_if_eq:NNT</code>	4531 , 5125 , 7857 , 7860	<code>\tl_if_single_p:n</code>	4843
<code>\tl_if_eq:NNTF</code> .	87 , 2170 , 4530 , 8388 , 8842	<code>\tl_if_single_token:n</code>	4849 , 4849
<code>\tl_if_eq:nnTF</code>	87	<code>\tl_if_single_token:nTF</code>	88
<code>\tl_if_eq_p:NN</code>	4529	<code>\tl_length:c</code>	4621 , 5030
<code>\tl_if_head_eq_catcode:nN</code> ...	4711 , 4727	<code>\tl_length:N</code> ...	90 , 4621 , 4626 , 4633 , 5029
<code>\tl_if_head_eq_catcode:nNTF</code>	92	<code>\tl_length:n</code> ...	90 , 4621 , 4621 , 4632 , 5026
<code>\tl_if_head_eq_charcode:nN</code> ..	4711 , 4711	<code>\tl_length:o</code>	4621 , 5028
<code>\tl_if_head_eq_charcode:nNF</code>	4726	<code>\tl_length:V</code>	4621 , 5027
<code>\tl_if_head_eq_charcode:nNT</code>	4725	<code>\tl_length_aux:n</code> .	4621 , 4624 , 4629 , 4631
<code>\tl_if_head_eq_charcode:nNTF</code>		<code>\tl_length_tokens:n</code> .	95 , 4939 , 4939 , 4954
.....	92 , 3707 , 3720 , 4724	<code>\tl_map_break</code>	89
<code>\tl_if_head_eq_charcode_p:nN</code>	4723	<code>\tl_map_break:</code> ...	4608 , 4608 , 8121 , 8129
<code>\tl_if_head_eq_meaning:nN</code> ...	4711 , 4743	<code>\tl_map_function:cN</code>	4563
<code>\tl_if_head_eq_meaning:nNTF</code> ...	93 , 9165	<code>\tl_map_function:NN</code>	
<code>\tl_if_head_eq_meaning_aux_normal:nN</code>		88 , 4563 , 4565 , 4575 , 4629 , 8092 , 8105
.....	4746 , 4750	<code>\tl_map_function:nN</code> .	88 , 4563 , 4563 , 4624
<code>\tl_if_head_eq_meaning_aux_special:nN</code>		<code>\tl_map_function_aux:NN</code>	4563
.....	4747 , 4758	<code>\tl_map_function_aux:Nn</code>	
<code>\tl_if_head_group:n</code>	4779 , 4779	..	4564 , 4567 , 4570 , 4573 , 4581 , 4591
<code>\tl_if_head_group:nTF</code> 93 , 4734 , 4768 , 4867		<code>\tl_map_inline:cn</code>	4576
<code>\tl_if_head_N_type:n</code>	4777 , 4777	<code>\tl_map_inline:Nn</code> ..	88 , 4576 , 4586 , 4596
		<code>\tl_map_inline:nn</code> ..	88 , 2724 , 4576 , 4576
		<code>\tl_map_inline_aux:n</code>	4576

`\tl_map_variable:cNn` [4597](#)
`\tl_map_variable:NNn` [88](#), [4597](#), [4599](#), [4607](#)
`\tl_map_variable:nNn` [89](#), [4597](#), [4597](#), [4600](#)
`\tl_map_variable_aux:NnN` [4597](#)
`\tl_map_variable_aux:Nnn` [4598](#), [4601](#), [4605](#)
`\tl_new:c` [4244](#),
[5054](#), [5586](#), [9377](#), [11648](#), [11734](#), [12015](#)
`\tl_new:cn` [4990](#)
`\tl_new:N` [82](#), [4244](#), [4244](#), [4249](#),
[4269](#), [4271](#), [4358](#), [4415](#), [4546](#), [4547](#),
[4829](#)–[4832](#), [4993](#), [5051](#)–[5053](#), [5333](#),
[5583](#), [5585](#), [6250](#), [6914](#), [6940](#), [6941](#),
[7778](#), [8308](#)–[8312](#), [8524](#), [8598](#), [8603](#),
[8791](#)–[8793](#), [9108](#)–[9111](#), [9220](#)–[9222](#),
[9224](#)–[9227](#), [9756](#), [9776](#), [9895](#), [9925](#),
[9932](#), [9935](#), [9938](#), [10158](#), [12374](#), [13324](#)
`\tl_new:Nn` [4990](#), [4991](#), [4996](#), [4997](#)
`\tl_new:Nx` [4990](#)
`\tl_put_left:cn` [4300](#)
`\tl_put_left:co` [4300](#)
`\tl_put_left:cV` [4300](#)
`\tl_put_left:cx` [4300](#)
`\tl_put_left:Nn` [83](#), [4300](#), [4300](#), [4316](#), [5078](#)
`\tl_put_left:No` [4300](#), [4304](#), [4318](#)
`\tl_put_left:NV` [4300](#), [4302](#), [4317](#)
`\tl_put_left:Nx` .. [4300](#), [4306](#), [4319](#), [5667](#)
`\tl_put_right:cn` [4324](#)
`\tl_put_right:co` [4324](#)
`\tl_put_right:cV` [4324](#)
`\tl_put_right:cx` [4324](#)
`\tl_put_right:Nn`
..... [84](#), [4324](#), [4324](#), [4340](#), [5080](#), [9182](#)
`\tl_put_right:No` . [4324](#), [4328](#), [4342](#), [8669](#)
`\tl_put_right:NV` [4324](#), [4326](#), [4341](#)
`\tl_put_right:Nx` ... [4324](#), [4330](#), [4343](#),
[5685](#), [6150](#), [8407](#), [8413](#), [8420](#), [8441](#),
[8450](#), [8461](#), [9151](#), [9173](#), [9186](#), [9195](#)
`\tl_remove_all:cn` [4470](#), [5021](#)
`\tl_remove_all:Nn` [85](#), [4470](#), [4470](#), [4474](#), [5020](#)
`\tl_remove_all_in:cn` [5015](#), [5021](#)
`\tl_remove_all_in:Nn` [5015](#), [5020](#)
`\tl_remove_in:cn` [5015](#), [5017](#)
`\tl_remove_in:Nn` [5015](#), [5016](#)
`\tl_remove_once:cn` [4464](#), [5017](#)
`\tl_remove_once:Nn`
..... [85](#), [4464](#), [4464](#), [4468](#), [5016](#)
`\tl_replace_all:cn` [4416](#), [5011](#)
`\tl_replace_all:Nnn` .. [84](#), [4416](#), [4420](#),
[4426](#), [4471](#), [5010](#), [5551](#), [9125](#), [9126](#)
`\tl_replace_all_aux:`
..... [4416](#), [4421](#), [4423](#), [4452](#), [4455](#)
`\tl_replace_all_in:cn` [5005](#), [5011](#)
`\tl_replace_all_in:Nnn` [5005](#), [5010](#)
`\tl_replace_aux:NNNnn`
.. [4416](#), [4417](#), [4419](#), [4421](#), [4423](#), [4428](#)
`\tl_replace_aux_ii:w` [4416](#), [4451](#), [4454](#), [4459](#)
`\tl_replace_in:cn` [5005](#), [5007](#)
`\tl_replace_in:Nnn` [5005](#), [5006](#)
`\tl_replace_once:cn` [4416](#), [5007](#)
`\tl_replace_once:Nnn`
..... [84](#), [4416](#), [4416](#), [4424](#), [4465](#), [5006](#)
`\tl_replace_once_aux:`
..... [4416](#), [4417](#), [4419](#), [4457](#)
`\tl_replace_once_aux_end:w`
..... [4416](#), [4460](#), [4462](#)
`\tl_rescan:nn` [86](#), [4402](#), [4402](#)
`\tl_rescan_aux:w` . [4359](#), [4371](#), [4377](#), [4409](#)
`\tl_reverse:c` [4936](#)
`\tl_reverse:N` [90](#), [4936](#), [4936](#), [4938](#)
`\tl_reverse:n` [90](#), [4924](#), [4924](#), [4935](#)
`\tl_reverse:o` [4924](#), [4937](#)
`\tl_reverse:V` [4924](#)
`\tl_reverse_group_preserve:nn` ... [4924](#)
`\tl_reverse_items:n` [90](#), [4634](#), [4634](#)
`\tl_reverse_items_aux:nN` [4634](#)
`\tl_reverse_items_aux:nw` [4635](#), [4636](#), [4639](#)
`\tl_reverse_tokens:n` [94](#), [4905](#), [4905](#), [4922](#)
`\tl_set:cf` [4282](#)
`\tl_set:cn` [4282](#), [9396](#), [9400](#)
`\tl_set:co` [4282](#)
`\tl_set:cx` [4282](#), [9380](#)
`\tl_set:Nc` [4999](#), [5001](#), [5002](#)
`\tl_set:Nf` [4282](#), [5698](#)
`\tl_set:Nn` [83](#), [2258](#), [2893](#),
[2914](#), [4282](#), [4282](#), [4294](#), [4296](#), [4360](#),
[4379](#), [4386](#), [4536](#), [4537](#), [4603](#), [5121](#),
[5130](#), [5144](#), [5147](#), [5170](#), [5173](#), [5185](#),
[5200](#), [5231](#), [5299](#), [5391](#), [5545](#), [5695](#),
[5707](#), [5863](#), [6087](#), [6100](#), [6103](#), [6109](#),
[6110](#), [6116](#), [6120](#), [6213](#), [6282](#), [6293](#),
[6921](#), [6925](#), [7113](#), [7444](#), [7445](#), [7780](#),
[7783](#), [8387](#), [8636](#), [8823](#), [8824](#), [9124](#),
[9234](#), [9271](#), [9338](#), [9364](#), [9432](#), [9556](#),
[9569](#), [9613](#), [10174](#), [10626](#), [10661](#),
[10744](#), [10769](#), [10795](#), [10898](#), [10916](#),
[11029](#), [11581](#), [11678](#), [11879](#), [12072](#),
[12390](#), [12724](#), [12782](#), [12794](#), [13309](#)
`\tl_set:No` [4282](#), [4284](#), [4937](#), [5003](#)
`\tl_set:NV` [4282](#)

- \tl_set:Nv [4282](#)
- \tl_set:Nx
 - [4282](#), [4286](#), [4295](#), [4396](#), [4417](#), [4421](#),
 - [4682](#), [5072](#), [5111](#), [5214](#), [5347](#), [5405](#),
 - [5492](#), [5497](#), [5520](#), [5537](#), [5556](#), [5604](#),
 - [5661](#), [5672](#), [5751](#), [5869](#), [5901](#), [5999](#),
 - [6127](#), [6264](#), [7965](#), [8243](#), [8263](#), [8368](#),
 - [8427](#), [8456](#), [8640](#), [9168](#), [9179](#), [9194](#),
 - [9232](#), [9258](#), [9265](#), [9268](#), [9554](#), [9564](#),
 - [9586](#), [9587](#), [9791](#), [9811](#), [9958](#), [9971](#),
 - [9982](#), [10074](#), [10121](#), [10146](#), [10209](#),
 - [10729](#), [10732](#), [10778](#), [11026](#), [11390](#),
 - [11399](#), [11422](#), [11441](#), [11594](#), [11691](#),
 - [11892](#), [12086](#), [12169](#), [12202](#), [12429](#),
 - [12545](#), [12554](#), [12682](#), [13126](#), [13151](#)
- \tl_set_eq:cc [4274](#),
 - [4277](#), [5066](#), [5598](#), [6063](#), [9442](#), [10261](#)
- \tl_set_eq:cN
 - [4274](#), [4275](#), [5065](#), [5597](#), [6062](#), [10259](#)
- \tl_set_eq:Nc [4274](#),
 - [4276](#), [5064](#), [5596](#), [6061](#), [9620](#), [10260](#)
- \tl_set_eq:NN [83](#), [4263](#), [4274](#), [4274](#), [5063](#),
 - [5595](#), [5667](#), [5685](#), [6060](#), [10169](#), [10258](#)
- \tl_set_rescan:cmn [4359](#)
- \tl_set_rescan:cno [4359](#)
- \tl_set_rescan:cnx [4385](#)
- \tl_set_rescan:Nnn
 - [85](#), [4359](#), [4359](#), [4381](#), [4382](#)
- \tl_set_rescan:Nno [4359](#), [9959](#)
- \tl_set_rescan:Nnx [4385](#), [4385](#), [4400](#)
- \tl_set_rescan_aux:NNnn
 - [4359](#), [4360](#), [4362](#), [4363](#)
- \tl_set_rescan_aux:NNnx
 - [4385](#), [4386](#), [4388](#), [4389](#)
- \tl_show:c [4812](#), [10267](#)
- \tl_show:N ... [93](#), [4812](#), [4812](#), [4813](#), [10266](#)
- \tl_show:n [94](#), [4814](#), [4814](#), [5339](#),
 - [5893](#), [6256](#), [6266](#), [7976](#), [8239](#), [8259](#)
- \tl_tail:f [4687](#)
- \tl_tail:n [92](#), [4687](#), [4691](#), [4695](#)
- \tl_tail:V [4687](#)
- \tl_tail:v [4687](#)
- \tl_tail:w [92](#), [4687](#), [4688](#), [9974](#), [9985](#)
- \tl_tail_aux:w [4692](#), [4693](#)
- \tl_tmp:w [4436](#),
 - [4455](#), [4460](#), [4556](#), [4557](#), [4642](#), [4680](#)
- \tl_to_lowercase:n [86](#),
 - [2285](#), [2299](#), [2686](#), [2726](#), [2819](#), [3086](#),
 - [4413](#), [4413](#), [8317](#), [8657](#), [9063](#), [9117](#)
- \tl_to_str:c [4610](#)
- \tl_to_str:N [89](#), [4610](#), [4610](#), [4611](#), [8377](#), [8378](#)
- \tl_to_str:n
 - [89](#), [3048](#), [3999](#), [4085](#), [4433](#), [4500](#),
 - [4513](#), [4609](#), [4609](#), [4699](#), [4708](#), [6069](#),
 - [6138](#), [6159](#), [6179](#), [6180](#), [6320](#), [6321](#),
 - [7809](#), [7893](#), [8322](#), [8470](#), [9232](#), [9265](#),
 - [9554](#), [9564](#), [9586](#), [9664](#), [9680](#), [10251](#)
- \tl_to_uppercase:n [86](#), [4413](#), [4414](#)
- \tl_trim_spaces:c [4642](#)
- \tl_trim_spaces:N ... [91](#), [4642](#), [4681](#), [4685](#)
- \tl_trim_spaces:n ... [90](#), [4642](#), [4644](#),
 - [4682](#), [4684](#), [5563](#), [5995](#), [9169](#), [9194](#)
- \tl_trim_spaces_aux:i:w
 - [4642](#), [4647](#), [4658](#), [4661](#), [5622](#)
- \tl_trim_spaces_aux_ii:w [4652](#), [4666](#), [5626](#)
- \tl_trim_spaces_aux_ii:w\tl_trim_spaces_aux_iii:w
 - [4642](#)
- \tl_trim_spaces_aux_iii:w
 - [4653](#), [4668](#), [4671](#), [4675](#), [5627](#)
- \tl_trim_spaces_aux_iv:w [4642](#), [4655](#), [4677](#)
- \tl_use:c [4612](#), [4613](#), [5732](#)
- \tl_use:N [89](#), [4612](#), [4612](#), [5731](#)
- \token_get_arg_spec:N ... [57](#), [3046](#), [3059](#)
- \token_get_prefix_arg_replacement_aux:wN
 - [3046](#), [3047](#), [3054](#), [3063](#), [3072](#)
- \token_get_prefix_spec:N . [57](#), [3046](#), [3050](#)
- \token_get_replacement_spec:N [3046](#), [3068](#)
- \token_get_replacement_text:N [57](#)
- \token_if_active:N [2660](#), [2660](#)
- \token_if_active:Nf [3203](#)
- \token_if_active:NT [3202](#)
- \token_if_active:NTF [52](#), [3204](#)
- \token_if_active_char:N [3188](#)
- \token_if_active_char:Nf [3203](#)
- \token_if_active_char:NT [3202](#)
- \token_if_active_char:NTF [3204](#)
- \token_if_active_char_p:N [3201](#)
- \token_if_active_p:N [3201](#)
- \token_if_alignment:N [2622](#), [2622](#)
- \token_if_alignment:Nf [3191](#)
- \token_if_alignment:NT [3190](#)
- \token_if_alignment:NTF [51](#), [3192](#)
- \token_if_alignment_p:N [3189](#)
- \token_if_alignment_tab:N [3188](#)
- \token_if_alignment_tab:Nf [3191](#)
- \token_if_alignment_tab:NT [3190](#)
- \token_if_alignment_tab:NTF [3192](#)
- \token_if_alignment_tab_p:N [3189](#)
- \token_if_chardef:N [2717](#), [2729](#)
- \token_if_chardef:NTF [53](#)

\token_if_chardef_aux:w	2731, 2734	\token_if_math_toggle:N	51, 3196
\token_if_chardef_p_aux:w	2717	\token_if_math_toggle_p:N	3193
\token_if_cs:N	2703, 2703	\token_if_mathchardef:N	2717, 2736
\token_if_cs:N	52	\token_if_mathchardef:N	53
\token_if_dim_register:N	2717, 2765	\token_if_mathchardef_aux:w .	2738, 2741
\token_if_dim_register:N	53	\token_if_mathchardef_p_aux:w . . .	2717
\token_if_dim_register_aux:w	2770, 2774	\token_if_other:N	2655, 2655
\token_if_dim_register_p_aux:w . .	2717	\token_if_other:N	3199
\token_if_eq_catcode:NN	2670, 2670	\token_if_other:NT	3198
\token_if_eq_catcode:NNTF	52	\token_if_other:N	52, 3200
\token_if_eq_catcode_p:NN	2950, 2951, 3100, 3101	\token_if_other_char:N	3188
\token_if_eq_charcode:NN	2675, 2675	\token_if_other_char:N	3199
\token_if_eq_charcode:NNTF	52	\token_if_other_char:NT	3198
\token_if_eq_meaning:NN	2665, 2665	\token_if_other_char:N	3197
\token_if_eq_meaning:NNT	2294	\token_if_other_p:N	3197
\token_if_eq_meaning:NNTF	52, 2309, 2971	\token_if_parameter:N	2627, 2629
\token_if_eq_meaning_p:NN	2952, 3102	\token_if_parameter:N	51
\token_if_expandable:N	2708, 2708	\token_if_primitive:N	2813, 2821
\token_if_expandable:N	53	\token_if_primitive:N	54
\token_if_group_begin:N	2607, 2607	\token_if_primitive_aux:NNw	2813, 2826, 2830
\token_if_group_begin:N	51	\token_if_primitive_aux_loop:N	2813, 2833, 2846, 2852
\token_if_group_end:N	2612, 2612	\token_if_primitive_aux_nullfont:N .	2813, 2834, 2838
\token_if_group_end:N	51	\token_if_primitive_aux_space:w	2813, 2832, 2837
\token_if_int_register:N	2717, 2743	\token_if_primitive_aux_undefined:N .	2813, 2858, 2864
\token_if_int_register:N	53	\token_if_primitive_auxii:Nw	2813, 2849, 2855
\token_if_int_register_aux:w	2748, 2752	\token_if_protected_long_macro:N . . .	2717, 2803
\token_if_int_register_p_aux:w . . .	2717	\token_if_protected_long_macro:N . . .	2717, 2803
\token_if_letter:N	2650, 2650	\token_if_protected_long_macro:N . . .	53
\token_if_letter:N	52	\token_if_protected_long_macro_aux:w .	2806, 2809
\token_if_long_macro:N	2717, 2796	\token_if_protected_long_macro_p_aux:w .	2717
\token_if_long_macro:N	53	\token_if_protected_macro:N	2717, 2787
\token_if_long_macro_aux:w	2798, 2801	\token_if_protected_macro:N	53
\token_if_long_macro_p_aux:w	2717	\token_if_protected_macro:N	2790, 2793
\token_if_macro:N	2680, 2689	\token_if_protected_macro_p_aux:w .	2717
\token_if_macro:N	52, 2823, 3052, 3061, 3070	\token_if_protected_macro:N	2717, 2787
\token_if_macro_p_aux:w	2680, 2691, 2694	\token_if_protected_macro:N	53
\token_if_math_shift:N	3188	\token_if_protected_macro_aux:w	2790, 2793
\token_if_math_shift:N	3195	\token_if_protected_macro_p_aux:w .	2717
\token_if_math_shift:NT	3194	\token_if_skip_register:N	2717, 2754
\token_if_math_shift:N	3196	\token_if_skip_register:N	53
\token_if_math_shift_p:N	3193	\token_if_skip_register_aux:w	2759, 2763
\token_if_math_subscript:N	2640, 2640	\token_if_skip_register_p_aux:w . .	2717
\token_if_math_subscript:N	52	\token_if_space:N	2645, 2645
\token_if_math_superscript:N	2635, 2635	\token_if_space:N	52
\token_if_math_superscript:N	51		
\token_if_math_toggle:N	2617, 2617		
\token_if_math_toggle:N	3195		
\token_if_math_toggle:NT	3194		

<code>\token_if_toks_register:N</code> . . .	2717, 2776	<code>\unpenalty</code>	644
<code>\token_if_toks_register:NTF</code>	53	<code>\unskip</code>	531
<code>\token_if_toks_register_aux:w</code>	2781, 2785	<code>\unvbox</code>	610
<code>\token_if_toks_register_p_aux:w</code>	2717	<code>\unvcopy</code>	611
<code>\token_new:Nn</code>	50, 2587, 2587, 2592, 2594–2596, 2598–2601	<code>\uppercase</code>	641
<code>\token_to_meaning:N</code>	50, 804, 804, 1207, 1217, 1230, 1849, 2692, 2732, 2739, 2749, 2760, 2771, 2782, 2791, 2799, 2807, 2827, 3055, 3064, 3073	<code>\use:c</code>	16, 851, 851, 979, 1160, 1162, 1164, 1166, 1951, 1961, 2033, 2034, 3370, 3666, 3676, 3819, 3828, 3830, 3832, 3833, 3837, 4002, 8433, 8729, 8740, 8753, 8756, 8762, 8773, 8781, 8787, 8809, 8870, 8892, 8897, 8905, 8928, 8950, 9279, 9286, 9448, 9657, 9963, 9993, 9996, 10013, 10016, 10017, 10020, 10023, 10307, 10369, 10425, 10475, 11627, 11714, 11925, 12112, 12448, 12975, 13038, 13053, 13055, 13146
<code>\token_to_str:c</code>	820, 821	<code>\use:n</code>	17, 861, 861, 995, 1024, 1298, 1445, 1447, 1451, 1459, 1461, 1469, 1473, 4761, 4778, 5245, 5462, 6084, 6313, 9075
<code>\token_to_str:N</code>	5, 51, 804, 805, 821, 1060, 1063, 1207, 1217, 1219, 1230, 1350, 1440, 1827, 2305, 4785, 5249, 5338, 5344, 5892, 5898, 6255, 6261, 6968, 6973, 7112, 7959, 7964, 8357–8361, 13279	<code>\use:nn</code>	861, 862, 1552, 3046, 3997, 5854
<code>\toks</code>	659	<code>\use:nnn</code>	861, 863
<code>\toksdef</code>	360	<code>\use:nnnn</code>	861, 864
<code>\tolerance</code>	570	<code>\use:x</code>	18, 852, 852, 4082, 4375, 8374
<code>\topmark</code>	451	<code>\use_i:nn</code>	17, 865, 865, 891, 1081, 1110, 1138, 1309, 1449, 1463, 1471, 10063, 10109, 10134, 10702, 11021, 11378, 11411, 12204, 12532, 12671
<code>\topmarks</code>	677	<code>\use_i:nnn</code>	17, 867, 867, 1092, 1337, 3055, 12171
<code>\topskip</code>	581	<code>\use_i:nnnn</code>	18, 867, 871
<code>\TotalHeight</code>	7137, 7141, 7145, 7149, 7156, 7658, 7685, 7686	<code>\use_i_after_else:nw</code>	1524, 1525
<code>\tracingassigns</code>	687	<code>\use_i_after_fi:nw</code>	1524, 1524
<code>\tracingcommands</code>	426	<code>\use_i_after_or:nw</code>	1524, 1526
<code>\tracinggroups</code>	694	<code>\use_i_after_orelse:nw</code>	1524, 1527
<code>\tracingifs</code>	690	<code>\use_i_delimit_by_q_nil:nw</code>	19, 878, 878
<code>\tracinglostchars</code>	427	<code>\use_i_delimit_by_q_recursion_stop:nw</code>	19, 45, 878, 880, 2419, 2435, 5887, 6249, 6327
<code>\tracingmacros</code>	428	<code>\use_i_delimit_by_q_stop:nw</code>	19, 878, 879, 1799, 5964
<code>\tracingnesting</code>	689	<code>\use_i_ii:nnn</code>	18, 867, 870, 1577
<code>\tracingonline</code>	429	<code>\use_ii:nn</code>	17, 865, 866, 893, 1083, 1112, 1140, 1311, 1446, 1452, 1460, 1474, 6076, 9139
<code>\tracingoutput</code>	430	<code>\use_ii:nnn</code>	17, 867, 868, 1094, 3064, 9187
<code>\tracingpages</code>	431	<code>\use_ii:nnnn</code>	18, 867, 872
<code>\tracingparagraphs</code>	432	<code>\use_iii:nnn</code>	17, 867, 869, 3073
<code>\tracingrestores</code>	433	<code>\use_iii:nnnn</code>	18, 867, 873
<code>\tracingscantokens</code>	688		
<code>\tracingstats</code>	434		
U			
<code>\U</code>	2724		
<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>\use_iv:nnnn</code>	18, 867, 874	<code>\vbox_gset_top:cn</code>	6521
<code>\use_none:n</code>	18, 881, 881, 995, 1024, 1063, 1065, 1300, 1444, 1448, 1450, 1458, 1462, 1470, 1472, 1809, 1873, 2421, 2437, 2861, 3597, 3712, 3716, 3721, 4477, 4638, 4678, 4764, 4782, 4810, 4852, 5049, 5199, 5228, 5261, 5456, 5483, 5484, 5636, 5772, 6014, 8297, 8299, 9136, 9169, 10077, 10124, 10149, 10198, 10240, 10652, 10688, 10761, 10787, 10841, 10962, 11105, 11425, 11444, 11603, 11658, 11700, 11744, 11901, 12025, 12095, 12317, 12434, 12477, 12861	<code>\vbox_gset_top:Nn</code>	130, 6521, 6523, 6526
<code>\use_none:nn</code>	881, 882, 1808, 4848, 5126, 12925	<code>\vbox_set:cn</code>	6517
<code>\use_none:nnn</code>	881, 883, 1807, 6156, 9180	<code>\vbox_set:cw</code>	6533, 6542
<code>\use_none:nnnn</code>	881, 884, 1806, 10034	<code>\vbox_set:Nn</code>	130, 6517, 6517–6519, 7017
<code>\use_none:nnnnn</code>	881, 885, 1805	<code>\vbox_set:Nw</code>	130, 6533, 6533, 6536, 6537, 6541, 7061
<code>\use_none:nnnnnn</code>	881, 886, 1804	<code>\vbox_set_end</code>	130
<code>\use_none:nnnnnnn</code>	881, 887, 1803	<code>\vbox_set_end:</code>	6533, 6539, 6543, 7067
<code>\use_none:nnnnnnnn</code>	881, 888, 1802	<code>\vbox_set_inline_begin:c</code>	6541, 6542
<code>\use_none:nnnnnnnnn</code>	881, 889, 1315, 1800, 1801	<code>\vbox_set_inline_begin:N</code>	6541, 6541
<code>\use_none_delimit_by_q_nil:w</code> 18, 875, 875		<code>\vbox_set_inline_end:</code>	6541, 6543
<code>\use_none_delimit_by_q_recursion_stop:w</code>	18, 45, 875, 877, 977, 1045, 1793, 2413, 2428, 4608, 5886, 6248	<code>\vbox_set_split_to_ht:NNn</code> 131, 6551, 6551	
<code>\use_none_delimit_by_q_stop:w</code>	18, 875, 876, 2324, 2328, 4440, 5759, 5951, 5957, 8463, 8477	<code>\vbox_set_to_ht:cnn</code>	6527
<code>\usepackage</code>	223	<code>\vbox_set_to_ht:Nnn</code>	130, 6527, 6527, 6530, 6531
V			
<code>\vadjust</code>	544	<code>\vbox_set_top:cn</code>	6521
<code>\valign</code>	379	<code>\vbox_set_top:Nn</code>	130, 6521, 6521, 6524, 6525, 7028, 7071
<code>\vbadness</code>	619	<code>\vbox_to_ht:nn</code>	130, 6514, 6514
<code>\vbox</code>	614	<code>\vbox_to_zero:n</code>	130, 6514, 6516
<code>\vbox:n</code>	129, 6512, 6512	<code>\vbox_top:n</code>	129, 6512, 6513
<code>\vbox_gset:cn</code>	6517	<code>\vbox_unpack:c</code>	6547
<code>\vbox_gset:cw</code>	6533, 6545	<code>\vbox_unpack:N</code>	131, 6547, 6547, 6549, 7028, 7071
<code>\vbox_gset:Nn</code>	130, 6517, 6518, 6520	<code>\vbox_unpack_clear:c</code>	6547
<code>\vbox_gset:Nw</code>	131, 6533, 6535, 6538, 6544	<code>\vbox_unpack_clear:N</code> 131, 6547, 6548, 6550	
<code>\vbox_gset_end</code>	131	<code>\vcenter</code>	465
<code>\vbox_gset_end:</code>	6533, 6540, 6546	<code>\vcoffin_set:cnn</code>	7013
<code>\vbox_gset_inline_begin:c</code>	6541, 6545	<code>\vcoffin_set:cw</code>	7057
<code>\vbox_gset_inline_begin:N</code>	6541, 6544	<code>\vcoffin_set:Nnn</code>	134, 7013, 7013, 7039
<code>\vbox_gset_inline_end:</code>	6541, 6546	<code>\vcoffin_set:Nnw</code>	134, 7057, 7057, 7086
<code>\vbox_gset_to_ht:cnn</code>	6527	<code>\vcoffin_set_end</code>	134
<code>\vbox_gset_to_ht:Nnn</code> 130, 6527, 6529, 6532		<code>\vcoffin_set_end:</code>	7057, 7064, 7085
W			
<code>\wd</code>	662	<code>\vfil</code>	526
<code>\widowpenalties</code>	722	<code>\vfill</code>	528
		<code>\vfildneg</code>	527
		<code>\vfuzz</code>	621
		<code>\voffset</code>	596
		<code>\voidb@x</code>	6462
		<code>\vrule</code>	535
		<code>\vsize</code>	578
		<code>\vskip</code>	529
		<code>\vsplit</code>	607
		<code>\vss</code>	530
		<code>\vtop</code>	615

<code>\widowpenalty</code>	549	<code>\xetex_if_engineTF</code>	4, 22
<code>\Width</code>	7137, 7142, 7146, 7150, 7157, 7655, 7688, 7689	<code>\xetex_XeTeXversion:D</code>	757, 1456
<code>\write</code>	412	<code>\XeTeXversion</code>	757
		<code>\xleaders</code>	538
		<code>\xspaceskip</code>	572
X			
<code>\X</code>	2720, 2724	Y	
<code>\xdef</code>	353	<code>\Y</code>	2721, 2724
<code>\xetex_if_engine:</code>	1444	<code>\year</code>	653
<code>\xetex_if_engine:F</code>	1451, 1462	Z	
<code>\xetex_if_engine:T</code>	1450, 1461	<code>\Z</code>	1834, 1842, 2722, 2724
<code>\xetex_if_engine:TF</code>	1452, 1463	<code>\z@</code>	4102
<code>\xetex_if_engine_p:</code>	1455, 1465, 1523		