

The L^AT_EX3 Sources

The L^AT_EX3 Project*

October 9, 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
V	The <code>l3expan</code> package: Argument expansion	24
12	Defining new variants	24
13	Methods for defining variants	25
14	Introducing the variants	25
15	Manipulating the first argument	26
16	Manipulating two arguments	27
17	Manipulating three arguments	28
18	Unbraced expansion	28
19	Preventing expansion	29
20	Internal functions and variables	30
VI	The <code>l3prg</code> package: Control structures	32
21	Defining a set of conditional functions	32
22	The boolean data type	34
23	Boolean expressions	36
24	Logical loops	37
25	Switching by case	38
26	Producing n copies	39
27	Detecting <code>TeX</code> 's mode	40
28	Internal programming functions	41
29	Experimental programmings functions	42
VII	The <code>l3quark</code> package: Quarks	43
30	Defining quarks	43

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

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

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

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

120	Viewing property lists	119
121	Experimental property list functions	119
122	Internal property list functions	119
XV	The l3box package: Boxes	121
123	Creating and initialising boxes	121
124	Using boxes	122
125	Measuring and setting box dimensions	123
126	Affine transformations	123
127	Box conditionals	125
128	The last box inserted	125
129	Constant boxes	125
130	Scratch boxes	125
131	Viewing box contents	126
132	Horizontal mode boxes	126
133	Vertical mode boxes	127
134	Primitive box conditionals	130
XVI	The l3coffins package: Coffin code layer	131
135	Creating and initialising coffins	131
136	Setting coffin content and poles	131
137	Coffin transformations	132
138	Joining and using coffins	133
139	Coffin diagnostics	134
XVII	The l3color package: Colour support	135

140	Colour in boxes	135
XVIII	The l3io package: Input–output operations	136
141	Managing streams	136
142	Writing to files	137
143	Wrapping lines in output	139
144	Reading from files	140
145	Internal input–output functions	141
XIX	The l3msg package: Messages	142
146	Creating new messages	142
147	Contextual information for messages	143
148	Issuing messages	144
149	Redirecting messages	145
150	Low-level message functions	146
151	Kernel-specific functions	147
152	Expandable errors	148
XX	The l3keys package: Key–value interfaces	150
153	Creating keys	151
154	Sub-dividing keys	155
155	Choice and multiple choice keys	156
156	Setting keys	158
157	Setting known keys only	158
158	Utility functions for keys	159
159	Low-level interface for parsing key–val lists	159

XXI	The <code>l3file</code> package: File operations	161
160	File operation functions	161
161	Internal file functions	162
XXII	The <code>l3fp</code> package: Floating-point operations	163
162	Floating-point variables	163
163	Conversion of floating point values to other formats	165
164	Rounding floating point values	165
165	Floating-point conditionals	166
166	Unary floating-point operations	167
167	Floating-point arithmetic	167
168	Floating-point power operations	168
169	Exponential and logarithm functions	169
170	Trigonometric functions	169
171	Constant floating point values	170
172	Notes on the floating point unit	170
XXIII	The <code>l3luatex</code> package: LuaTeX-specific functions	172
173	Breaking out to Lua	172
174	Category code tables	173
XXIV	Implementation	174
175	Bootstrap code	174
175.1	Format-specific code	174
175.2	Package-specific code	175
175.3	Dealing with package-mode meta-data	177
175.4	The <code>\pdfstrcmp</code> primitive in X _Y TeX	180
175.5	Engine requirements	180
175.6	The L ^A T _E X3 code environment	181

176	l3names implementation	182
177	l3basics implementation	192
177.1	Renaming some \TeX primitives (again)	193
177.2	Defining functions	194
177.3	Selecting tokens	195
177.4	Gobbling tokens from input	196
177.5	Conditional processing and definitions	197
177.6	Dissecting a control sequence	202
177.7	Exist or free	204
177.8	Defining and checking (new) functions	205
177.9	More new definitions	207
177.10	Copying definitions	209
177.11	Undefining functions	210
177.12	Defining functions from a given number of arguments	210
177.13	Using the signature to define functions	212
177.14	Checking control sequence equality	214
177.15	Diagnostic wrapper functions	214
177.16	Engine specific definitions	214
177.17	Doing nothing functions	215
177.18	String comparisons	215
177.19	Deprecated functions	216
178	l3expan implementation	217
178.1	General expansion	217
178.2	Hand-tuned definitions	221
178.3	Definitions with the automated technique	223
178.4	Last-unbraced versions	224
178.5	Preventing expansion	226
178.6	Defining function variants	226
178.7	Variants which cannot be created earlier	229
179	l3prg implementation	229
179.1	Defining a set of conditional functions	229
179.2	The boolean data type	230
179.3	Boolean expressions	231
179.4	Logical loops	237
179.5	Switching by case	238
179.6	Producing n copies	239
179.7	Detecting \TeX 's mode	242
179.8	Internal programming functions	243
179.9	Experimental programmings functions	244
179.10	Deprecated functions	247
180	l3quark implementation	247

181	l3token implementation	250
181.1	Character tokens	250
181.2	Generic tokens	252
181.3	Token conditionals	253
181.4	Peeking ahead at the next token	261
181.5	Decomposing a macro definition	266
181.6	Experimental token functions	267
181.7	Deprecated functions	268
182	l3int implementation	271
182.1	Integer expressions	271
182.2	Creating and initialising integers	273
182.3	Setting and incrementing integers	274
182.4	Using integers	275
182.5	Integer expression conditionals	275
182.6	Integer expression loops	278
182.7	Formatting integers	279
182.8	Converting from other formats to integers	284
182.9	Viewing integer	288
182.10	Constant integers	288
182.11	Scratch integers	289
182.12	Registers for earlier modules	289
182.13	Deprecated functions	289
183	l3skip implementation	290
183.1	Length primitives renamed	290
183.2	Creating and initialising dim variables	291
183.3	Setting dim variables	291
183.4	Utilities for dimension calculations	292
183.5	Dimension expression conditionals	292
183.6	Dimension expression loops	294
183.7	Using dim expressions and variables	295
183.8	Viewing dim variables	296
183.9	Constant dimensions	296
183.10	Scratch dimensions	296
183.11	Creating and initialising skip variables	296
183.12	Setting skip variables	297
183.13	Skip expression conditionals	298
183.14	Using skip expressions and variables	298
183.15	Inserting skips into the output	299
183.16	Viewing skip variables	299
183.17	Constant skips	299
183.18	Scratch skips	299
183.19	Creating and initialising muskip variables	299
183.20	Setting muskip variables	300
183.21	Using muskip expressions and variables	301

183.2	Viewing <code>muskip</code> variables	301
183.2	Experimental skip functions	301
184	<code>l3tl</code> implementation	302
184.1	Functions	302
184.2	Adding to token list variables	303
184.3	Reassigning token list category codes	305
184.4	Reassigning token list character codes	307
184.5	Modifying token list variables	307
184.6	Token list conditionals	309
184.7	Mapping to token lists	312
184.8	Using token lists	313
184.9	Working with the contents of token lists	314
184.10	The first token from a token list	316
184.1	Viewing token lists	320
184.1	Constant token lists	320
184.1	Scratch token lists	321
184.1	Experimental functions	321
184.1	Deprecated functions	326
185	<code>l3seq</code> implementation	328
185.1	Allocation and initialisation	329
185.2	Appending data to either end	330
185.3	Modifying sequences	330
185.4	Sequence conditionals	331
185.5	Recovering data from sequences	332
185.6	Mapping to sequences	335
185.7	Sequence stacks	337
185.8	Viewing sequences	338
185.9	Experimental functions	338
185.1	Deprecated interfaces	344
186	<code>l3clist</code> implementation	344
186.1	Allocation and initialisation	345
186.2	Removing spaces around items	346
186.3	Adding data to comma lists	347
186.4	Comma lists as stacks	348
186.5	Using comma lists	349
186.6	Modifying comma lists	349
186.7	Comma list conditionals	351
186.8	Mapping to comma lists	352
187	Viewing comma lists	354
187.1	Scratch comma lists	355
187.2	Experimental functions	355
187.3	Deprecated interfaces	358

188	l3prop implementation	359
	188.1 Allocation and initialisation	359
	188.2 Accessing data in property lists	360
	188.3 Property list conditionals	363
	188.4 Recovering values from property lists with branching	364
	188.5 Mapping to property lists	365
	188.6 Viewing property lists	366
	188.7 Experimental functions	367
	188.8 Deprecated interfaces	368
189	l3box implementation	369
	189.1 Creating and initialising boxes	370
	189.2 Measuring and setting box dimensions	371
	189.3 Using boxes	371
	189.4 Box conditionals	372
	189.5 The last box inserted	372
	189.6 Constant boxes	373
	189.7 Scratch boxes	373
	189.8 Viewing box contents	373
	189.9 Horizontal mode boxes	373
	189.10 Vertical mode boxes	375
	189.11 Affine transformations	377
190	l3coffins Implementation	385
	190.1 Coffins: data structures and general variables	385
	190.2 Basic coffin functions	387
	190.3 Coffins: handle and pole management	391
	190.4 Coffins: calculation of pole intersections	394
	190.5 Aligning and typesetting of coffins	398
	190.6 Rotating coffins	402
	190.7 Resizing coffins	407
	190.8 Coffin diagnostics	410
	190.9 Messages	416
191	l3color Implementation	416
192	l3io implementation	417
	192.1 Primitives	417
	192.2 Variables and constants	417
	192.3 Stream management	419
	192.4 Deferred writing	424
	192.5 Immediate writing	424
	192.6 Special characters for writing	425
	192.7 Hard-wrapping lines based on length	425
	192.8 Reading input	430
	192.9 Deprecated functions	431

193	l3msg implementation	432
194	Creating messages	432
194.1	Messages: support functions and text	433
194.2	Showing messages: low level mechanism	434
194.3	Displaying messages	436
194.4	Kernel-specific functions	441
194.5	Expandable errors	446
194.6	Deprecated functions	447
195	l3keys Implementation	447
195.1	Low-level interface	447
195.2	Constants and variables	451
195.3	The key defining mechanism	452
195.4	Turning properties into actions	454
195.5	Creating key properties	458
195.6	Setting keys	461
195.7	Utilities	464
195.8	Messages	465
195.9	Deprecated functions	466
196	l3file implementation	466
197	l3fp Implementation	470
197.1	Constants	470
197.2	Variables	471
197.3	Parsing numbers	474
197.4	Internal utilities	477
197.5	Operations for fp variables	478
197.6	Transferring to other types	483
197.7	Rounding numbers	490
197.8	Unary functions	492
197.9	Basic arithmetic	494
197.10	Arithmetic for internal use	503
197.11	Trigonometric functions	509
197.12	Exponent and logarithm functions	522
197.13	Tests for special values	544
197.14	Floating-point conditionals	544
197.15	Messages	550
198	l3luatex implementation	551
198.1	Category code tables	552
	Index	555

Part I

Introduction to expl3 and this document

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

1 Naming functions and variables

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

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

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

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

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

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

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

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

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

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

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

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

skip “Rubber” lengths.

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

tl Token list variables: placeholder for a token list.

1.1 Terminological inexactitude

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

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

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

2 Documentation conventions

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

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

`\ExplSyntaxOn`
`\ExplSyntaxOff`

`\ExplSyntaxOn ... \ExplSyntaxOff`

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

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

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

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

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

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

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

`\cs_to_str:N` ☆

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

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

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

`\seq_map_function:NN` ☆

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

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

`\xetex_if_engineTF` ☆

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

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

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

`\l_tmpa_tl`

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

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

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

The normal description text.

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

3 Formal language conventions which apply generally

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

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

Part II

The l3bootstrap package

Bootstrap code

4 Using the L^AT_EX3 modules

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

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

`\ExplSyntaxOn`
`\ExplSyntaxOff`

Updated: 2011-08-13

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

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

`\ExplSyntaxNamesOn`
`\ExplSyntaxNamesOff`

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

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

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

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

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

`\GetIdInfo`

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

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

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

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

Part III

The l3names package Namespace for primitives

5 Setting up the L^AT_EX3 programming language

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

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

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

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

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

`\pdftex_...` Introduced by pdfT_EX;

`\xetex_...` Introduced by X_YT_EX;

`\luatex_...` Introduced by LuaT_EX.

Part IV

The l3basics package

Basic definitions

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

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

6 No operation functions

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

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

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

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

7 Grouping material

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

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

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

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

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

8 Control sequences and functions

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

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

8.1 Defining functions

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

8.2 Defining new functions using primitive parameter text

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

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

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

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

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

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

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

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

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

<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 if 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 if 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_predicate:w</code>	★	<code>\if_predicate:w <predicate> <true code> \else: <false code> \fi:</code>
------------------------------	---	---

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

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

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

<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 <code><true code></code> if currently in horizontal mode, otherwise execute <code><false code></code> . 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 <code><cs></code> 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 <code><cs></code> is free according to the criteria for <code>\cs_if_free_p:N</code> , and if not raises a kernel-level error.
--------------------------------	--

Part V

The l3expan package

Argument expansion

This module provides generic methods for expanding \TeX 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 \LaTeX 3 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.

12 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.

13 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.

14 Introducing the variants

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

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

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

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

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

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

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

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

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

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

```
\cs_set_nopar:Npn \tl_set:Nf { \exp_args:Nnf \tl_set:Nn }
```

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

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

16 Manipulating two arguments

<hr/> <code>\exp_args:NNo</code> ★	<code>\exp_args:NNo</code> $\langle token1 \rangle$ $\langle token2 \rangle$ $\{\langle tokens \rangle\}$
<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/> <code>\exp_args:Nno</code> ★	<code>\exp_args:Nno</code> $\langle token \rangle$ $\{\langle tokens_1 \rangle\}$ $\{\langle tokens_2 \rangle\}$
<code>\exp_args:(NnV Nnf Noo 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.

17 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.*

18 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.

19 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).
<hr/> <hr/>	Updated: 2011-06-03		

20 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 LaTeX3 approach as this makes them more readily visible in the log and so forth.

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

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

Part VI

The l3prg package

Control structures

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

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

21 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.

22 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/> <hr/>	
<code>\bool_gset:Nn</code>	<code>\bool_gset:Nn <boolean> {\<boolexpr>}</code>
<code>\bool_gset:cn</code>	Evaluates the <i><boolean expression></i> as described for <code>\bool_if:n(TF)</code> , and sets the <i><boolean></i> variable to the logical truth of this evaluation. This assignment is global.
<hr/> <hr/>	
<code>\bool_if_p:N</code> ★	<code>\bool_if_p:N {\<boolean>}</code>
<code>\bool_if_p:c</code> ★	<code>\bool_if:NTF {\<boolean>} {\<true code>} {\<false code>}</code>
<code>\bool_if:NTF</code> ★	Tests the current truth of <i><boolean></i> , and continues expansion based on this result.
<code>\bool_if:cTF</code> ★	
<hr/> <hr/>	
<code>\l_tmpa_bool</code>	A scratch boolean for local assignment. It is never used by the kernel code, and so is safe for use with any L ^A T _E X3-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.
<hr/> <hr/>	
<code>\g_tmpa_bool</code>	A scratch boolean for global assignment. It is never used by the kernel code, and so is safe for use with any L ^A T _E X3-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.

23 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.

24 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`.

25 Switching by case

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

`\prg_case_int:nnn` ☆

Updated: 2011-09-17

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

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

As an example of `\prg_case_int:nnn`:

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

will leave “Medium” in the input stream.

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

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

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

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

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

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

26 Producing *n* copies

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

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>*.

27 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\textit{TF}</code>	★	<code>\mode_if_vertical:TF {\langle true code \rangle} {\langle false code \rangle}</code>

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

28 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$'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.

$\text{T}_{\text{E}}\text{X}$ hackers note: This is a protected version of `\prg_do_nothing:`, which therefore stops $\text{T}_{\text{E}}\text{X}$'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.*)

29 Experimental programmings functions

<code>\prg_quicksort:n</code>	<code>\prg_quicksort:n { {\langle item_1 \rangle} {\langle item_2 \rangle} ... {\langle item n \rangle} }</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 {\langle element \rangle}</code>
<code>\prg_quicksort_compare:nnTF</code>	<code>\prg_quicksort_compare:nnTF {\langle element_1 \rangle} {\langle element_2 \rangle}</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_`.

30 Defining quarks

<u><code>\quark_new:N</code></u>	<code>\quark_new:N <quark></code> Creates a new <code><quark></code> which expands only to <code><quark></code> . The <code><quark></code> will be defined globally, and an error message will be raised if the name was already taken.
<u><code>\q_stop</code></u>	Used as a marker for delimited arguments, such as $\cs_set:Npn \tmp:w #1#2 \q_stop {#1}$
<u><code>\q_mark</code></u>	Used as a marker for delimited arguments when <code>\q_stop</code> is already in use. Quark to mark a null value in structured variables or functions. Used as an end delimiter when this may itself may need to be tested (in contrast to <code>\q_stop</code> , which is only ever used as a delimiter).
<u><code>\q_no_value</code></u>	A canonical value for a missing value, when one is requested from a data structure. This is therefore used as a “return” value by functions such as <code>\prop_get:NnN</code> if there is no data to return.

31 Quark tests

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

<code>\quark_if_nil_p:N</code>	★	<code>\quark_if_nil_p:N <token></code>
<code>\quark_if_nil:NTF</code>	★	<code>\quark_if_nil:NTF <token> {\true code} {\false code}</code>

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

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

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

<code>\quark_if_no_value_p:n</code>	★	<code>\quark_if_no_value_p:n {\token list}</code>
<code>\quark_if_no_value:nTF</code>	★	<code>\quark_if_no_value:nTF {\token list} {\true code} {\false code}</code>

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

32 Recursion

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

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

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

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

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

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

Updated: 2011-09-06

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

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

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

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

Updated: 2011-09-06

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

33 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.

34 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
```

35 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 $\langle integer\ expression \rangle$.

`\char_show_value_sfcode:n`

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

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

36 Generic tokens

`\token_new:Nn`

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

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

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

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

`\c_catcode_letter_token`
`\c_catcode_other_token`

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

`\c_catcode_active_tl`

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

37 Converting tokens

`\token_to_meaning:N` ★

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

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

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

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

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

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

38 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:NTF</code>	<code>*</code>	<code>\token_if_math_subscript:NTF</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:NTF</code>	<code>*</code>	<code>\token_if_space:NTF</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:NTF</code>	<code>*</code>	<code>\token_if_letter:NTF</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:NTF</code>	<code>*</code>	<code>\token_if_other:NTF</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:NTF</code>	<code>*</code>	<code>\token_if_active:NTF</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:NNTF</code>	<code>*</code>	<code>\token_if_eq_catcode:NNTF</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:NNTF</code>	<code>*</code>	<code>\token_if_eq_charcode:NNTF</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:NNTF</code>	<code>*</code>	<code>\token_if_eq_meaning:NNTF</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:NTF</code>	<code>*</code>	<code>\token_if_macro:NTF</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:NTF</code>	<code>*</code>	<code>\token_if_cs:NTF</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 a 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.

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

<code>\peek_meaning:NTF</code>	<code>\peek_meaning: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 meaning as the <i><test token></i> (as defined by the test <code>\token_if_eq_meaning: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_meaning_ignore_spaces:NTF</code>	<code>\peek_meaning_ignore_spaces:NTF <test token> {\true code} {\false code}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same meaning as the <i><test token></i> (as defined by the test <code>\token_if_eq_meaning: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_meaning_remove:NTF</code>	<code>\peek_meaning_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 meaning as the <i><test token></i> (as defined by the test <code>\token_if_eq_meaning: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_meaning_remove_ignore_spaces:NTF</code>	<code>\peek_meaning_remove_ignore_spaces:NTF <test token> {\true code} {\false code}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same meaning as the <i><test token></i> (as defined by the test <code>\token_if_eq_meaning: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).

40 Decomposing a macro definition

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

`\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

41 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`”).

42 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/> <hr/>	<code>\int_div_truncate:nn</code> ★	<code>\int_div_truncate:nn {⟨integer expr₁⟩} {⟨integer expr₂⟩}</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/> <hr/>	<code>\int_max:nn</code> ★ <code>\int_min:nn</code> ★	<code>\int_max:nn {⟨integer expr₁⟩} {⟨integer expr₂⟩}</code> <code>\int_min:nn {⟨integer expr₁⟩} {⟨integer expr₂⟩}</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/> <hr/>	<code>\int_mod:nn</code> ★	<code>\int_mod:nn {⟨integer expr₁⟩} {⟨integer expr₂⟩}</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.

43 Creating and initialising integers

<hr/> <hr/>	<code>\int_new:N</code> <code>\int_new:c</code>	<code>\int_new:N ⟨integer⟩</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/> <hr/>	<code>\int_const:Nn</code> <code>\int_const:cn</code>	<code>\int_const:Nn ⟨integer⟩ {⟨integer expression⟩}</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/> <hr/>	<code>\int_zero:N</code> <code>\int_zero:c</code>	<code>\int_zero:N ⟨integer⟩</code>	Sets <i>⟨integer⟩</i> to 0 within the scope of the current T _E X group.
<hr/> <hr/>	<code>\int_gzero:N</code> <code>\int_gzero:c</code>	<code>\int_gzero:N ⟨integer⟩</code>	Sets <i>⟨integer⟩</i> to 0 globally, <i>i.e.</i> not restricted by the current T _E X group level.
<hr/> <hr/>	<code>\int_set_eq:NN</code> <code>\int_set_eq:(cN Nc cc)</code>	<code>\int_set_eq:NN ⟨integer1⟩ ⟨integer2⟩</code>	Sets the content of <i>⟨integer1⟩</i> equal to that of <i>⟨integer2⟩</i> . This assignment is restricted to the current T _E X group level.
<hr/> <hr/>	<code>\int_gset_eq:NN</code> <code>\int_gset_eq:(cN Nc cc)</code>	<code>\int_gset_eq:NN ⟨integer1⟩ ⟨integer2⟩</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 T _E X group level.

44 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> 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> 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> 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.

45 Using integers

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

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.

46 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>	$\{\langle integer expression \rangle\}$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
<code>\int_if_odd_p:n</code>	★		
<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.

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

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

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

48 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>
---------------------------------	--

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

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

Updated: 2011-09-17

`\int_to_alph:n { 1 }`

places `a` in the input stream,

`\int_to_alph:n { 26 }`

is represented as `z` and

`\int_to_alph:n { 27 }`

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

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

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

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

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

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

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

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

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

`\int_to_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.*

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

49 Converting from other formats to integers

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

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

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

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

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

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

50 Viewing integers

<hr/> <code>\int_show:N</code>	<code>\int_show:N \langle integer \rangle</code>
<hr/> <code>\int_show:c</code>	Displays the value of the $\langle integer \rangle$ on the terminal.

51 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.

52 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.

53 Internal functions

<hr/> <code>\int_get_digits:n</code> ★ <hr/>	<code>\int_get_digits:n</code> $\langle value \rangle$ Parses the $\langle value \rangle$ to leave the absolute $\langle value \rangle$ in the input stream. This may therefore be used to remove multiple sign tokens from the $\langle value \rangle$ (which may be symbolic).
<hr/> <code>\int_get_sign:n</code> ☆ <hr/>	<code>\int_get_sign:n</code> $\langle value \rangle$ Parses the $\langle value \rangle$ to leave a single sign token (either + or -) in the input stream. This may therefore be used to sanitise sign tokens from the $\langle value \rangle$ (which may be symbolic).
<hr/> <code>\int_to_letter:n</code> ★ <hr/> <div>Updated: 2011-09-17 ☆</div>	<code>\int_to_letter:n</code> $\langle integer\ value \rangle$ For $\langle integer\ values \rangle$ from 0 to 9, leaves the $\langle value \rangle$ in the input stream unchanged. For $\langle integer\ values \rangle$ from 10 to 35, leaves the appropriate upper case letter (from the standard English alphabet) in the input stream: for example, 10 is converted to A, 11 to B, <i>etc.</i>
<hr/> <code>\int_to_roman:w</code> ★ <hr/>	<code>\int_to_roman:w</code> $\langle integer \rangle$ $\langle space \rangle$ or $\langle non-expandable\ token \rangle$ Converts $\langle integer \rangle$ to it lower case Roman representation. Expansion ends when a space or non-expandable token is found. Note that this function produces a string of letters with category code 12 and that protected functions <i>are</i> expanded by this process. Negative $\langle integer \rangle$ values result in no output, although the function does not terminate expansion until a suitable endpoint is found in the same way as for positive numbers. 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</code> $\langle integer1 \rangle$ $\langle relation \rangle$ $\langle integer2 \rangle$ $\langle true\ code \rangle$ <code>\else:</code> $\langle false\ code \rangle$ <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>	<code><integer></code>	<code><case0></code>
<code>\or</code>	★	<code>\or:</code>	<code><case1></code>	
		<code>\or:</code>	<code>...</code>	
		<code>\else:</code>	<code><default></code>	
		<code>\fi:</code>		

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

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

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

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

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

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

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

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

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

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

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

Part X

The l3skip package

Dimensions and skips

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

54 Creating and initialising `dim` variables

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

`\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.

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

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

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

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

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

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

55 Setting `dim` variables

<code>\dim_add:Nn</code>
<code>\dim_add:cn</code>

`\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.

<code>\dim_gadd:Nn</code>
<code>\dim_gadd:cn</code>

`\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.

<code>\dim_set:Nn</code>
<code>\dim_set:cn</code>

`\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.

<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> <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> <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> <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> <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> <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.

56 Utilities for dimension calculations

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

Parses the two *<dimension expressions>* and converts the ratio of the two to a form suitable for use inside a *<dimension expression>*. This ratio is then left in the input stream, allowing syntax such as

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

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

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

will display 327680/655360 on the terminal.

57 Dimension expression conditionals

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

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

Equal	=
Greater than	>
Less than	<

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

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

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

58 Dimension expression loops

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

59 Using dim expressions and variables

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

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

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

<code>\dim_use:c</code> ☆

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

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

60 Viewing dim variables

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

<code>\dim_show:c</code>

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

61 Constant dimensions

<code>\c_max_dim</code>

The maximum value that can be stored as a dimension or skip (these are equivalent).

<code>\c_zero_dim</code>

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

62 Scratch dimensions

<code>\l_tmpa_dim</code>	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.
<code>\l_tmpb_dim</code>	
<code>\l_tmpc_dim</code>	

<code>\g_tmpa_dim</code>	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.
<code>\g_tmpb_dim</code>	

63 Creating and initialising skip variables

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

<code>\skip_zero:N</code>	<code>\skip_zero:N <skip></code>
<code>\skip_zero:c</code>	Sets <i><skip></i> to 0 pt within the scope of the current T _E X group.

<code>\skip_gzero:N</code>	<code>\skip_gzero:N <skip></code>
<code>\skip_gzero:c</code>	Sets <i><skip></i> to 0 pt globally, <i>i.e.</i> not restricted by the current T _E X group level.

64 Setting skip variables

<code>\skip_add:Nn</code>	<code>\skip_add:Nn <skip> {<skip expression>}</code>
<code>\skip_add:cn</code>	Adds the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is local.

<code>\skip_gadd:Nn</code>	<code>\skip_gadd:Nn <skip> {<skip expression>}</code>
<code>\skip_gadd:cn</code>	Adds the result of the <i><skip expression></i> to the current content of the <i><skip></i> . This assignment is global.

<code>\skip_set:Nn</code>	<code>\skip_set:Nn <skip> {<skip expression>}</code>
<code>\skip_set:cn</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.

<code>\skip_gset:Nn</code>	<code>\skip_gset:Nn <skip> {<skip expression>}</code>
<code>\skip_gset:cn</code>	Sets $\langle skip \rangle$ to the value of $\langle skip expression \rangle$, 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.

<code>\skip_set_eq:NN</code>	<code>\skip_set_eq:NN <skip1> <skip2></code>
<code>\skip_set_eq:(cN Nc cc)</code>	Sets the content of $\langle skip1 \rangle$ equal to that of $\langle skip2 \rangle$. This assignment is restricted to the current T _E X group level.

<code>\skip_gset_eq:NN</code>	<code>\skip_gset_eq:NN <skip1> <skip2></code>
<code>\skip_gset_eq:(cN Nc cc)</code>	Sets the content of $\langle skip1 \rangle$ equal to that of $\langle skip2 \rangle$. This assignment is global and so is not limited by the current T _E X group level.

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

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

65 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 $\langle skip expressions \rangle$ as described for `\skip_eval:n`. The two results are then compared for exact equality, *i.e.* both the fixed and rubber components must be the same for the test to be true.

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

66 Using skip expressions and variables

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

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

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

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

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

67 Viewing skip variables

<code>\skip_show:N</code>	<code>\skip_show:N \langle skip \rangle</code>
<code>\skip_show:c</code>	

Displays the value of the $\langle skip \rangle$ on the terminal.

68 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).

69 Scratch skips

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

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

70 Creating and initialising muskip variables

<hr/> <code>\muskip_new:N</code> <hr/> <code>\muskip_new:c</code> <hr/>	<code>\muskip_new:N <muskip></code> Creates a new $\langle muskip \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle muskip \rangle$ will initially be equal to 0 mu.
<hr/> <code>\muskip_zero:N</code> <hr/> <code>\muskip_zero:c</code> <hr/>	<code>\skip_zero:N <muskip></code> Sets $\langle muskip \rangle$ to 0 mu within the scope of the current TeX group.
<hr/> <code>\muskip_gzero:N</code> <hr/> <code>\muskip_gzero:c</code> <hr/>	<code>\muskip_gzero:N <muskip></code> Sets $\langle muskip \rangle$ to 0 mu globally, <i>i.e.</i> not restricted by the current TeX group level.

71 Setting muskip variables

<hr/> <code>\muskip_add:Nn</code> <hr/> <code>\muskip_add:cn</code> <hr/>	<code>\muskip_add:Nn <muskip> {<muskip expression>}</code> Adds the result of the $\langle muskip \text{ expression} \rangle$ to the current content of the $\langle muskip \rangle$. This assignment is local.
<hr/> <code>\muskip_gadd:Nn</code> <hr/> <code>\muskip_gadd:cn</code> <hr/>	<code>\muskip_gadd:Nn <muskip> {<muskip expression>}</code> Adds the result of the $\langle muskip \text{ expression} \rangle$ to the current content of the $\langle muskip \rangle$. This assignment is global.
<hr/> <code>\muskip_set:Nn</code> <hr/> <code>\muskip_set:cn</code> <hr/>	<code>\muskip_set:Nn <muskip> {<muskip expression>}</code> Sets $\langle muskip \rangle$ to the value of $\langle muskip \text{ expression} \rangle$, which must evaluate to a math length with units and may include a rubber component (for example 1 mu plus 0.5 mu. This assignment is restricted to the current TeX group.
<hr/> <code>\muskip_gset:Nn</code> <hr/> <code>\muskip_gset:cn</code> <hr/>	<code>\muskip_gset:Nn <muskip> {<muskip expression>}</code> Sets $\langle muskip \rangle$ to the value of $\langle muskip \text{ expression} \rangle$, which must evaluate to a math length with units and may include a rubber component (for example 1 mu plus 0.5 mu. This assignment is global and is not limited to the current TeX group level.
<hr/> <code>\muskip_set_eq:NN</code> <hr/> <code>\muskip_set_eq:(cN Nc cc)</code> <hr/>	<code>\muskip_set_eq:NN <muskip1> <muskip2></code> Sets the content of $\langle muskip1 \rangle$ equal to that of $\langle muskip2 \rangle$. This assignment is restricted to the current TeX group level.
<hr/> <code>\muskip_gset_eq:NN</code> <hr/> <code>\muskip_gset_eq:(cN Nc cc)</code> <hr/>	<code>\muskip_gset_eq:NN <muskip1> <muskip2></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 TeX group level.

<hr/> <code>\muskip_sub:Nn</code> <hr/>	<code>\muskip_sub:Nn <muskip> {\muskip expression}</code>
<code>\muskip_sub:cn</code> <hr/>	Subtracts the result of the <i><muskip expression></i> to the current content of the <i><skip></i> . This assignment is local.
<hr/> <code>\muskip_gsub:Nn</code> <hr/>	<code>\muskip_gsub:Nn <muskip> {\muskip expression}</code>
<code>\muskip_gsub:cn</code> <hr/>	Subtracts the result of the <i><muskip expression></i> to the current content of the <i><muskip></i> . This assignment is global.

72 Using muskip expressions and variables

<hr/> <code>\muskip_eval:n</code> ★ <hr/>	<code>\muskip_eval:n {\muskip expression}</code>
	Evaluates the <i><muskip expression></i> , expanding any skips and token list variables within the <i><expression></i> 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 <i><muglue denotation></i> 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 <i><internal muglue></i> .
<hr/> <code>\muskip_use:N</code> ★ <code>\muskip_use:c</code> ★ <hr/>	<code>\muskip_use:N <muskip></code>
	Recovers the content of a <i><skip></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>\muskip_eval:n</code>).

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

73 Inserting skips into the output

<hr/> <code>\skip_horizontal:N</code> <hr/>	<code>\skip_horizontal:N <skip></code>
<code>\skip_horizontal:(c n)</code> <hr/>	<code>\skip_horizontal:n {\skipexpr}</code>
	Inserts a horizontal <i><skip></i> into the current list.
	T_EXhackers note: <code>\skip_horizontal:N</code> is the T _E X primitive <code>\hskip</code> renamed.
<hr/> <code>\skip_vertical:N</code> <hr/>	<code>\skip_vertical:N <skip></code>
<code>\skip_vertical:(c n)</code> <hr/>	<code>\skip_vertical:n {\skipexpr}</code>
	Inserts a vertical <i><skip></i> into the current list.
	T_EXhackers note: <code>\skip_vertical:N</code> is the T _E X primitive <code>\vskip</code> renamed.

74 Viewing muskip variables

<code>\muskip_show:N</code>	<code>\muskip_show:N <muskip></code>
<code>\muskip_show:c</code>	Displays the value of the <code><muskip></code> on the terminal.

75 Internal functions

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

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

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

76 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>
--	--

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

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 direct 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.

77 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 <code><tl var></code> globally.

<code>\tl_clear_new:N</code>	<code>\tl_clear_new:N <tl var></code>
<code>\tl_clear_new:c</code>	If the <code><tl var></code> already exists, clears it within the scope of the current T _E X group. If the <code><tl var></code> 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 <code><tl var></code> 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 <code><tl var></code> already exists, clears it globally. If the <code><tl var></code> 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 <code><tl var1></code> equal to that of <code><tl var2></code> . 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 <code><tl var1></code> equal to that of <code><tl var2></code> . This assignment is global and so is not limited by the current T _E X group level.

78 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 `<tl var>` to contain `<tokens>`, removing any previous content from the variable. This assignment is restricted to the current T_EX 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 `<tl var>` to contain `<tokens>`, removing any previous content from the variable. This assignment is global and is not limited to the current T_EX 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 `<tokens>` to the left side of the current content of `<tl var>`. This modification is restricted to the current T_EX 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.

79 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, <div style="text-align: center;"> <code>\tl_set:Nn \l_tmpa_tl {abbccd} \tl_remove_all:Nn \l_tmpa_tl {bc}</code> </div> 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.

80 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`.

81 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.

82 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.

83 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 *<function>* to every *<item>* stored within the *<token list>*. The *<function>* should consist of code which will receive the *<item>* stored in the *<variable>*. One variable mapping can be nested inside another. See also `\tl_map_inline:nn`.

<code>\tl_map_break</code> ☆	<code>\tl_map_break:</code>
------------------------------	-----------------------------

Used to terminate a `\tl_map...` function before all entries in the *<token list variable>* have been processed. This will normally take place within a conditional statement, for example

```

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

```

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

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

<code>\tl_to_str:n</code> ☆	<code>\tl_to_str:n {<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.

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

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

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

85 Working with the content of token lists

<hr/> <code>\tl_length:n</code> ★ <code>\tl_length:(V o)</code> ★ <hr/> Updated: 2011-08-13	<code>\tl_length:n {⟨tokens⟩}</code> <p>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 $\langle integer denotation \rangle$.</p>
<hr/> <code>\tl_length:N</code> ★ <code>\tl_length:c</code> ★ <hr/> Updated: 2011-08-13	<code>\tl_length:N {⟨tl var⟩}</code> <p>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 $\langle integer denotation \rangle$.</p>
<hr/> <code>\tl_reverse:n</code> ★ <code>\tl_reverse:(V o)</code> ★ <hr/> Updated: 2011-08-13	<code>\tl_reverse:n {⟨token list⟩}</code> <p>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>.</p>
<hr/> <code>\tl_reverse:N</code> <code>\tl_reverse:c</code> <hr/> Updated: 2011-08-13	<code>\tl_reverse:N {⟨tl var⟩}</code> <p>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 $\text{T}_{\text{E}}\text{X}$ group. See also <code>\tl_reverse:n</code>.</p>
<hr/> <code>\tl_reverse_items:n</code> ★ <hr/> New: 2011-08-13	<code>\tl_reverse_items:n {⟨token list⟩}</code> <p>Reverses the order of the $\langle items \rangle$ stored in $\langle tl var \rangle$, so that $\{\langle item_1 \rangle\} \{\langle item_2 \rangle\} \{\langle item_3 \rangle\} \dots \{\langle item_n \rangle\}$ becomes $\{\langle item_n \rangle\} \dots \{\langle item_3 \rangle\} \{\langle item_2 \rangle\} \{\langle item_1 \rangle\}$. 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>.</p>
<hr/> <code>\tl_trim_spaces:n</code> ★ <hr/> New: 2011-07-09 Updated: 2011-08-13	<code>\tl_trim_spaces:n ⟨token list⟩</code> <p>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.</p>

$\text{T}_{\text{E}}\text{X}$ hackers 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/>	

86 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.

`\tl_tail:n` ★ `\tl_tail:n {⟨tokens⟩}`

`\tl_tail:(V|v|f)` ★

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.

`\tl_tail:w` ★ `\tl_tail:w {⟨tokens⟩} \q_stop`

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.

`\str_head:n` ★ `\str_head:n {⟨tokens⟩}`

`\str_tail:n` ★ `\str_tail:n {⟨tokens⟩}`

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.

`\tl_if_head_eq_catcode_p:nN` ★ `\tl_if_head_eq_catcode_p:nN {⟨token list⟩} ⟨test token⟩`

`\tl_if_head_eq_catcode:nNTF` ★ `\tl_if_head_eq_catcode:nNTF {⟨token list⟩} ⟨test token⟩`
`{⟨true code⟩} {⟨false 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.

`\tl_if_head_eq_charcode_p:nN` ★ `\tl_if_head_eq_charcode_p:nN {⟨token list⟩} ⟨test token⟩`

`\tl_if_head_eq_charcode_p:fN` ★ `\tl_if_head_eq_charcode:nNTF {⟨token list⟩} ⟨test token⟩`

`\tl_if_head_eq_charcode:nNTF` ★ `{⟨true code⟩} {⟨false code⟩}`

`\tl_if_head_eq_charcode:fNTF` ★

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:nNTF</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.

87 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`.

88 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.
--------------------------	--

89 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.
--	--

90 Experimental token list functions

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

New: 2011-08-11

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

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

New: 2011-08-11

Counts the number of \TeX tokens in the $\langle tokens \rangle$ and leaves this information in the input stream. Every token, including spaces and braces, contributes one to the total; thus for instance, the length of `a~{bc}` is 6. This function requires three expansions, giving an $\langle integer denotation \rangle$.

<code>\tl_expandable_uppercase:n</code> ★	<code>\tl_expandable_uppercase:n {⟨tokens⟩}</code>
<code>\tl_expandable_lowercase:n</code> ★	<code>\tl_expandable_lowercase:n {⟨tokens⟩}</code>

New: 2011-08-13

The `\tl_expandable_uppercase:n` function works through all of the $\langle tokens \rangle$, 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.

\TeX hackers note: Begin-group and end-group characters are normalized and become `{` and `}`, respectively.

91 Internal functions

<code>\q_tl_act_mark</code>
<code>\q_tl_act_stop</code>

Quarks which are only used for the particular purposes of `\tl_act_...` functions.

Part XII

The l3seq package

Sequences and stacks

L^AT_EX3 implements a “sequence” data type, which contain an ordered list of entries which may contain any *⟨balanced text⟩*. It is possible to map functions to sequences such that the function is applied to every item in the sequence.

Sequences are also used to implement stack functions in L^AT_EX3. This is achieved using a number of dedicated stack functions.

92 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 $\langle sequence1 \rangle$.

93 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.

94 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.

95 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.

96 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 $\langle sequence \rangle$.

97 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.

98 Sequences as stacks

Sequences can be used as stacks, where data is pushed to and popped from the top of the sequence. (The left of a sequence is the top, for performance reasons.) The stack functions for sequences are not intended to be mixed with the general ordered data functions detailed in the previous section: a sequence should either be used as an ordered data type or as a stack, but not in both ways.

`\seq_get:NN`
`\seq_get:cN`

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

Reads the top item from a *⟨sequence⟩* into the *⟨token list variable⟩* without removing it from the *⟨sequence⟩*. The *⟨token list variable⟩* is assigned locally. If *⟨sequence⟩* is empty an error will be raised.

`\seq_pop:NN`
`\seq_pop:cN`

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

Pops the top item from a *⟨sequence⟩* into the *⟨token list variable⟩*. Both of the variables are assigned locally. If *⟨sequence⟩* is empty an error will be raised.

`\seq_gpop:NN`
`\seq_gpop:cN`

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

Pops the top item from a *⟨sequence⟩* into the *⟨token list variable⟩*. The *⟨sequence⟩* is modified globally, while the *⟨token list variable⟩* is assigned locally. If *⟨sequence⟩* is empty an error will be raised.

```
\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.

99 Viewing sequences

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

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

Displays the entries in the $\langle sequence \rangle$ in the terminal.

100 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).

101 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 `\tokens` before continuing reading the input stream. This function is a copy of `\seq_map_break:n`, but is used in situations which are not mappings.

`\seq_break_point:n` ★ `\seq_break_point:n \tokens`

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 `\tokens` 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.

102 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 $\langle comma list1 \rangle$.

103 Adding data to comma lists

<hr/> <code>\clist_set:Nn</code> <code>\clist_set:(NV No Nx cn cV co cx)</code> <hr/> <div>New: 2011-09-06</div>	<code>\clist_set:Nn <comma list> {\langle item1 \rangle, \dots, \langle item_n \rangle}</code> 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/> <div>New: 2011-09-06</div>	<code>\clist_gset:Nn <comma list> {\langle item1 \rangle, \dots, \langle item_n \rangle}</code> 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.

104 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 $\langle token list \rangle$.

105 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).

106 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.

107 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.

108 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

<code>\clist_gpop:Nn</code>	<code>\clist_gpop:Nn <comma list> <token list variable></code>
<code>\clist_gpop:cN</code>	

Pops the left-most item from a *<comma list>* into the *<token list variable>*, *i.e.* removes the item from the comma list and stores it in the *<token list variable>*. The *<comma list>* is modified globally, while the assignment of the *<token list variable>* is local.

<code>\clist_push:Nn</code>	<code>\clist_push:Nn <comma list> {(items)}</code>
<code>\clist_push:(NV No Nx cn cV co cx)</code>	

Adds the *{(items)}* to the top of the *<comma list>*. Spaces are removed from both sides of each item. The assignment is restricted to the current \TeX group.

<code>\clist_gpush:Nn</code>	<code>\clist_gpush:Nn <comma list> {(items)}</code>
<code>\clist_gpush:(NV No Nx cn cV co cx)</code>	

Pushes the *<items>* onto the end of the top of the *<comma list>*. Spaces are removed from both sides of each item. The assignment is global.

109 Viewing comma lists

<code>\clist_show:N</code>	<code>\clist_show:N <comma list></code>
<code>\clist_show:c</code>	

Displays the entries in the *<comma list>* in the terminal.

110 Scratch comma lists

<code>\l_tmpa_clist</code>	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.
<code>\l_tmpb_clist</code>	
<small>New: 2011-09-06</small>	

<code>\g_tmpa_clist</code>	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.
<code>\g_tmpb_clist</code>	
<small>New: 2011-09-06</small>	

111 Experimental comma list functions

This section contains functions which may or may not be retained, depending on how useful they are found to be.

<code>\clist_length:N</code>	<code>\clist_length:N <comma list></code>
<code>\clist_length:(c n)</code>	

Leaves the number of items in the *<comma list>* in the input stream as an *<integer denotation>*. The total number of items in a *<comma list>* will include those which are empty and duplicates, *i.e.* every item in a *<comma list>* 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 T _E X 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.

112 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.

113 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.

114 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.

115 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.

116 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.

117 Property list conditionals

<code>\prop_if_empty_p:N</code>	★	<code>\prop_if_empty_p:N</code>	$\langle property\ list \rangle$
<code>\prop_if_empty_p:c</code>	★	<code>\prop_if_empty:NTF</code>	$\langle property\ list \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>\prop_if_empty:NTF</code>	★	Tests if the $\langle property\ list \rangle$ is empty (containing no entries).	
<code>\prop_if_empty:cTF</code>	★		

<code>\prop_if_in_p:Nn</code>	★	<code>\prop_if_in:NnTF</code>	$\langle property\ list \rangle$ $\{\langle key \rangle\}$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>\prop_if_in_p:(NV No cn cV co)</code>	★		
<code>\prop_if_in:NnTF</code>	★		
<code>\prop_if_in:(NV No cn cV co)TF</code>	★		

Updated: 2011-09-15

Tests if the $\langle key \rangle$ is present in the $\langle property\ list \rangle$, making the comparison using the method described by `\str_if_eq:nNTF`.

T_EXhackers note: This function iterates through every key–value pair in the $\langle property\ list \rangle$ and is therefore slower than using the non-expandable `\prop_get:NnNTF`.

118 Recovering values from property lists with branching

The functions in this section combine tests for the presence of a key in a property list with recovery of the associated valued. This makes them useful for cases where different cases follow dependent on the presence or absence of a key in a property list. They offer increased readability and performance over separate testing and recovery phases.

<code>\prop_get:NnNTF</code>	<code>\prop_get:NnNTF</code>	$\langle property\ list \rangle$ $\{\langle key \rangle\}$ $\langle token\ list\ variable \rangle$
<code>\prop_get:(NVN NoN cnN cVN coN)TF</code>		$\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Updated: 2011-08-28

If the $\langle key \rangle$ is not present in the $\langle property\ list \rangle$, leaves the $\langle false\ code \rangle$ in the input stream and leaves the $\langle token\ list\ variable \rangle$ unchanged. If the $\langle key \rangle$ is present in the $\langle property\ list \rangle$, stores the corresponding $\langle value \rangle$ in the $\langle token\ list\ variable \rangle$ without removing it from the $\langle property\ list \rangle$. The $\langle token\ list\ variable \rangle$ is assigned locally.

<code>\prop_pop:NnNTF</code>	<code>\prop_pop:NnNTF</code>	$\langle property\ list \rangle$ $\{\langle key \rangle\}$ $\langle token\ list\ variable \rangle$
<code>\prop_pop:cnNTF</code>		$\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

New: 2011-08-18

If the $\langle key \rangle$ is not present in the $\langle property\ list \rangle$, leaves the $\langle false\ code \rangle$ in the input stream and leaves the $\langle token\ list\ variable \rangle$ unchanged. If the $\langle key \rangle$ is present in the $\langle property\ list \rangle$, pops the corresponding $\langle value \rangle$ in the $\langle token\ list\ variable \rangle$, *i.e.* removes the item from the $\langle property\ list \rangle$. Both the $\langle property\ list \rangle$ and the $\langle token\ list\ variable \rangle$ are assigned locally.

119 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.

120 Viewing property lists

`\prop_show:N`
`\prop_show:c`

`\prop_show:N` $\langle property\ list \rangle$
 Displays the entries in the $\langle property\ list \rangle$ in the terminal.

121 Experimental property list functions

This section contains functions which may or may not be retained, depending on how useful they are found to be.

`\prop_gpop:NnNTF`
`\prop_gpop:cnNTF`

New: 2011-08-18

`\prop_gpop:NnNTF` $\langle property\ list \rangle$ $\{\langle key \rangle\}$ $\langle token\ list\ variable \rangle$
 $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

If the $\langle key \rangle$ is not present in the $\langle property\ list \rangle$, leaves the $\langle false\ code \rangle$ in the input stream and leaves the $\langle token\ list\ variable \rangle$ unchanged. If the $\langle key \rangle$ is present in the $\langle property\ list \rangle$, pops the corresponding $\langle value \rangle$ in the $\langle token\ list\ variable \rangle$, *i.e.* removes the item from the $\langle property\ list \rangle$. The $\langle property\ list \rangle$ is modified globally, while the $\langle token\ list\ variable \rangle$ is assigned locally.

`\prop_map_tokens:Nn` ☆
`\prop_map_tokens:cn` ☆

New: 2011-08-18

`\prop_map_tokens:Nn` $\langle property\ list \rangle$ $\{\langle code \rangle\}$

Analogue of `\prop_map_function:Nn` which maps several tokens instead of a single function. Useful in particular when mapping through a property list while keeping track of a given key.

`\prop_get:Nn` ★
`\prop_get:cn` ★

Updated: 2011-09-15

`\prop_get:Nn` $\langle property\ list \rangle$ $\{\langle key \rangle\}$

Expands to the $\langle value \rangle$ corresponding to the $\langle key \rangle$ in the $\langle property\ list \rangle$. If the $\langle key \rangle$ is missing, this has an empty expansion.

T_EXhackers note: This function is slower than the non-expandable analogue `\prop_get:NnN`.

122 Internal property list functions

`\q_prop`

The internal token used to separate out property list entries, separating both the $\langle key \rangle$ from the $\langle value \rangle$ and also one entry from another.

`\c_empty_prop`

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_`.

123 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.

124 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> .

125 Measuring and setting box dimensions

`\box_dp:N` ★ `\box_dp:N` $\langle box \rangle$

`\box_dp:c` ★
Calculates the depth (below the baseline) of the $\langle box \rangle$ and leaves this in the input stream. The output of this function is suitable for use in a $\langle dimension expression \rangle$ for calculations.

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

`\box_ht:N` ★ `\box_ht:N` $\langle box \rangle$

`\box_ht:c` ★
Calculates the height (above the baseline) of the $\langle box \rangle$ and leaves this in the input stream. The output of this function is suitable for use in a $\langle dimension expression \rangle$ for calculations.

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

`\box_wd:N` ★ `\box_wd:N` $\langle box \rangle$

`\box_wd:c` ★
Calculates the width of the $\langle box \rangle$ and leaves this in the input stream. The output of this function is suitable for use in a $\langle dimension expression \rangle$ for calculations.

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

`\box_set_dp:Nn` `\box_set_dp:Nn` $\langle box \rangle$ $\{ \langle dimension expression \rangle \}$

`\box_set_dp:cn`
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.

`\box_set_ht:Nn` `\box_set_ht:Nn` $\langle box \rangle$ $\{ \langle dimension expression \rangle \}$

`\box_set_ht:cn`
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.

`\box_set_wd:Nn` `\box_set_wd:Nn` $\langle box \rangle$ $\{ \langle dimension expression \rangle \}$

`\box_set_wd:cn`
Set the width of the $\langle box \rangle$ to the value of the $\{ \langle dimension expression \rangle \}$. This is a global assignment.

126 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 <box> {<x-size>} {<y-size>}`

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

`\box_resize_to_ht_plus_dp:Nnn <box> {<y-size>}`

Resize the $\langle box \rangle$ to $\langle y-size \rangle$ vertically, scaling the horizontal size by the same amount ($\langle y-size \rangle$ is a dimension expression). The $\langle y-size \rangle$ is the vertical size (height plus depth) of the box. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the resizing is applied. A negative size will cause the material in the $\langle box \rangle$ to be reversed in direction, but the reference point of the $\langle box \rangle$ will be unchanged. The resizing applies within the current \TeX group level.

This function is experimental

`\box_resize_to_wd:Nnn``\box_resize_to_wd:cn`

New: 2011-09-02

`\box_resize_to_wd:Nnn <box> {<x-size>}`

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

`\box_rotate:Nn <box> {<angle>}`

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

`\box_scale:Nnn <box> {<x-scale>} {<y-scale>}`

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

127 Box conditionals

<hr/>	
<code>\box_if_empty_p:N</code> *	<code>\box_if_empty_p:N</code> $\langle box \rangle$
<code>\box_if_empty_p:c</code> *	<code>\box_if_empty:NTF</code> $\langle box \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>\box_if_empty:NTF</code> *	Tests if $\langle box \rangle$ is a empty (equal to <code>\c_empty_box</code>).
<code>\box_if_empty:cTF</code> *	
<hr/>	
<code>\box_if_horizontal_p:N</code> *	<code>\box_if_horizontal_p:N</code> $\langle box \rangle$
<code>\box_if_horizontal_p:c</code> *	<code>\box_if_horizontal:NTF</code> $\langle box \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>\box_if_horizontal:NTF</code> *	Tests if $\langle box \rangle$ is a horizontal box.
<code>\box_if_horizontal:cTF</code> *	
<hr/>	
<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> *	
<hr/>	

128 The last box inserted

<hr/>	
<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>	
<hr/>	

<hr/>	
<code>\l_last_box</code>	This is a box containing the last item added to the current partial list, except in the case of the main vertical list (main galley), in which case this box is always void. Notice that although this is not a constant, it is <i>not</i> settable by the programmer but is instead varied by \TeX .
<hr/>	

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

129 Constant boxes

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

130 Scratch boxes

<hr/>	
<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 \LaTeX 3-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>	
<hr/>	

131 Viewing box contents

<code>\box_show:N</code>	<code>\box_show:N <box></code>
<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`.

132 Horizontal mode boxes

<code>\hbox:n</code>	<code>\hbox:n {\langle contents \rangle}</code>
	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`.

<code>\hbox_to_wd:nn</code>	<code>\hbox_to_wd:nn {\langle dimexpr \rangle} {\langle contents \rangle}</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.

<code>\hbox_to_zero:n</code>	<code>\hbox_to_zero:n {\langle contents \rangle}</code>
	Typesets the $\langle contents \rangle$ into a horizontal box of zero width and then includes this box in the current list for typesetting.

<code>\hbox_set:Nn</code>	<code>\hbox_set:Nn <box> {\langle contents \rangle}</code>
<code>\hbox_set:cn</code>	Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is local.

<code>\hbox_gset:Nn</code>	<code>\hbox_gset:Nn <box> {\langle contents \rangle}</code>
<code>\hbox_gset:cn</code>	Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. The assignment is global.

<code>\hbox_set_to_wd:Nnn</code>	<code>\hbox_set_to_wd:Nnn <box> {\langle dimexpr \rangle} {\langle contents \rangle}</code>
<code>\hbox_set_to_wd:cnn</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.

<code>\hbox_gset_to_wd:Nnn</code>	<code>\hbox_gset_to_wd:Nnn <box> {\langle dimexpr \rangle} {\langle contents \rangle}</code>
<code>\hbox_gset_to_wd:cnn</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/> <code>\hbox_overlap_right:n</code> <hr/>	<code>\hbox_overlap_right:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the right of the insertion point.
<hr/> <code>\hbox_overlap_left:n</code> <hr/>	<code>\hbox_overlap_left:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the left of the insertion point.
<hr/> <code>\hbox_set:Nw</code> <code>\hbox_set:cw</code> <code>\hbox_set_end</code> <hr/>	<code>\hbox_set:Nw <box> <contents> \hbox_set_end:</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. 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.
<hr/> <code>\hbox_gset:Nw</code> <code>\hbox_gset:cw</code> <code>\hbox_gset_end</code> <hr/>	<code>\hbox_gset:Nw <box> <contents> \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.
<hr/> <code>\hbox_unpack:N</code> <code>\hbox_unpack:c</code> <hr/>	<code>\hbox_unpack:N <box></code> Unpacks the content of the horizontal $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set.
<hr/> <code>\hbox_unpack_clear:N</code> <code>\hbox_unpack_clear:c</code> <hr/>	<code>\hbox_unpack_clear:N <box></code> Unpacks the content of the horizontal $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set. The $\langle box \rangle$ is then cleared globally.

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

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

133 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.

<hr/> <hr/>	<code>\vbox:n</code> <code>{\langle contents \rangle}</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 _E X primitive <code>\vbox</code> .
<hr/> <hr/>	<code>\vbox_top:n</code> <code>{\langle contents \rangle}</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 _E X primitive <code>\vtop</code> .
<hr/> <hr/>	<code>\vbox_to_ht:nn</code> <code>{\langle dimexpr \rangle} {\langle contents \rangle}</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>{\langle contents \rangle}</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:Nn</code> $\langle box \rangle$ <code>{\langle contents \rangle}</code>
<code>\vbox_set:cn</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:Nn</code> $\langle box \rangle$ <code>{\langle contents \rangle}</code>
<code>\vbox_gset:cn</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:Nn</code> $\langle box \rangle$ <code>{\langle contents \rangle}</code>
<code>\vbox_set_top:cn</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:Nn</code> $\langle box \rangle$ <code>{\langle contents \rangle}</code>
<code>\vbox_gset_top:cn</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/> <code>\vbox_set_to_ht:Nnn</code> <code>\vbox_set_to_ht:cnn</code> <hr/>	<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/> <code>\vbox_gset_to_ht:Nnn</code> <code>\vbox_gset_to_ht:cnn</code> <hr/>	<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/> <code>\vbox_set:Nw</code> <code>\vbox_set:cw</code> <code>\vbox_set_end</code> <hr/>	<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.
<hr/> <code>\vbox_gset:Nw</code> <code>\vbox_gset:cw</code> <code>\vbox_gset_end</code> <hr/>	<code>\vbox_gset:Nw <box> <contents> \vbox_gset_end:</code> 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 <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.
<hr/> <code>\vbox_set_split_to_ht:NNn</code> <hr/>	<code>\vbox_set_split_to_ht:NNn <box1> <box2> {<dimexpr>}</code> 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 _E X primitive <code>\vsplit</code> .	
<hr/> <code>\vbox_unpack:N</code> <code>\vbox_unpack:c</code> <hr/>	<code>\vbox_unpack:N <box></code> 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 _E X primitive <code>\unvcopy</code> .	
<hr/> <code>\vbox_unpack_clear:N</code> <code>\vbox_unpack_clear:c</code> <hr/>	<code>\vbox_unpack:N <box></code> 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 _E X primitive <code>\unvbox</code> .	

134 Primitive box conditionals

`\if_hbox:N` ★ `\if_hbox:N` $\langle box \rangle$
 $\langle true\ code \rangle$
`\else:`
 $\langle false\ code \rangle$
`\fi:`

Tests if $\langle box \rangle$ is a horizontal box.

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

`\if_vbox:N` ★ `\if_vbox:N` $\langle box \rangle$
 $\langle true\ code \rangle$
`\else:`
 $\langle false\ code \rangle$
`\fi:`

Tests if $\langle box \rangle$ is a vertical box.

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

`\if_box_empty:N` ★ `\if_box_empty:N` $\langle box \rangle$
 $\langle true\ code \rangle$
`\else:`
 $\langle false\ code \rangle$
`\fi:`

Tests if $\langle box \rangle$ is an empty (void) box.

T_EXhackers note: This is the T_EX 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).

135 Creating and initialising coffins

`\coffin_new:N``\coffin_new:N <coffin>``\coffin_new:c`

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.

New: 2011-08-17

`\coffin_clear:N``\coffin_clear:N <coffin>``\coffin_clear:c`

Clears the content of the $\langle coffin \rangle$ within the current T_EX group level.

New: 2011-08-17

`\coffin_set_eq:NN``\coffin_set_eq:NN <coffin1> <coffin2>``\coffin_set_eq:(Nc|cN|cc)`

Sets both the content and poles of $\langle coffin1 \rangle$ equal to those of $\langle coffin2 \rangle$ within the current T_EX group level.

New: 2011-08-17

136 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:Nn <coffin> {\material}``\hcoffin_set:cn`

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.

New: 2011-08-17

Updated: 2011-09-03

`\hcoffin_set:Nw``\hcoffin_set:Nw <coffin> {\material} \hcoffin_set_end:``\hcoffin_set:cw``\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.

New: 2011-09-10

`\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 $\langle coffin \rangle$.

137 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.

138 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.

139 Coffin diagnostics

```
\coffin_display_handles:cn
\coffin_display_handles:cn
```

Updated: 2011-09-02

```
\coffin_display_handles:Nn <coffin> {<colour>}
```

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.

```
\coffin_mark_handle:Nnnn
\coffin_mark_handle:cnnn
```

Updated: 2011-09-02

```
\coffin_mark_handle:Nnnn <coffin> {<pole1>} {<pole2>} {<colour>}
```

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.

```
\coffin_show_structure:N
\coffin_show_structure:c
```

```
\coffin_show_structure:N <coffin>
```

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.

140 Colour in boxes

Controlling the colour of text in boxes requires a small number of control functions, so that the boxed material uses the colour at the point where it is set, rather than where it is used.

<hr/> <code>\color_group_begin</code> <code>\color_group_end</code> <hr/> <div>New: 2011-09-03</div> <hr/>	<code>\color_group_begin:</code> ... <code>\color_group_end:</code> Creates a colour group: one used to “trap” colour settings.
<hr/> <code>\color_ensure_current</code> <hr/> <div>New: 2011-09-03</div> <hr/>	<code>\color_ensure_current:</code> Ensures that material inside a box will use the foreground colour at the point where the box is set, rather than that in force when the box is used. This function should usually be used within a <code>\color_group_begin: ... \color_group_end: group</code> .

Part XVIII

The l3io package

Input–output operations

Reading and writing from file streams is handled in L^AT_EX3 using functions with prefixes `\iow_...` (file reading) and `\ior_...` (file writing). Many of the basic functions are very similar, with reading and writing using the same syntax and function concepts. As a result, the reading and writing functions are documented together where this makes sense.

As T_EX is limited to 16 input streams and 16 output streams, direct use of the streams by the programmer is not supported in L^AT_EX3. Instead, an internal pool of streams is maintained, and these are allocated and deallocated as needed by other modules. As a result, the programmer should close streams when they are no longer needed, to release them for other processes.

Reading from or writing to a file requires a $\langle stream \rangle$ to be used. This is a csname which refers to the file being processed, and is independent of the name of the file (except of course that the file name is needed when the file is opened).

141 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>	Closes the <code><stream></code> . Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.
Updated: 2011-09-26	

<code>\iow_close:N</code>	<code>\iow_close:N <stream></code>
<code>\iow_close:c</code>	Closes the <code><stream></code> . Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.
Updated: 2011-09-26	

<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.	

142 Writing to files

<code>\iow_now:Nn</code>	<code>\iow_now:Nn <stream> {<tokens>}</code>
<code>\iow_now:Nx</code>	This functions writes <code><tokens></code> to the specified <code><stream></code> immediately (<i>i.e.</i> the write operation is called on expansion of <code>\iow_now:Nn</code>).

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

<code>\iow_log:n</code>	<code>\iow_log:n {<tokens>}</code>
<code>\iow_log:x</code>	This function writes the given <code><tokens></code> to the log (transcript) file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

<code>\iow_term:n</code>	<code>\iow_term:n {<tokens>}</code>
<code>\iow_term:x</code>	This function writes the given <code><tokens></code> to the terminal file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

<code>\iow_now_when_avail:Nn</code>	<code>\iow_now_when_avail:Nn <stream> {<tokens>}</code>
<code>\iow_now_when_avail:Nx</code>	If <code><stream></code> is open, writes the <code><tokens></code> to the <code><stream></code> in the same manner as <code>\iow_now:Nn</code> . If the <code><stream></code> is not open, the <code><tokens></code> are simply thrown away.

<code>\iow_shipout:Nn</code>	<code>\iow_shipout:Nn <stream> {<tokens>}</code>
<code>\iow_shipout:Nx</code>	This functions writes <code><tokens></code> to the specified <code><stream></code> when the current page is finalised (<i>i.e.</i> at shipout). The x-type variants expand the <code><tokens></code> at the point where the function is used but <i>not</i> when the resulting tokens are written to the <code><stream></code> (<i>cf.</i> <code>\iow_shipout_x:Nn</code>).

<code>\iow_shipout_x:Nn</code> <code>\iow_shipout_x:Nx</code>	<code>\iow_shipout_x:Nn <stream> {<tokens>}</code> <p>This function writes <i><tokens></i> to the specified <i><stream></i> when the current page is finalised (<i>i.e.</i> at shipout). The <i><tokens></i> are expanded at the time of writing in addition to any expansion when the function is used. This makes these functions suitable for including material finalised during the page building process (such as the page number integer).</p>
--	--

TeXhackers note: `\iow_shipout_x:Nn` is the TeX primitive `\write` renamed.

<code>\iow_char:N</code> ★	<code>\iow_char:N <token></code> <p>Inserts <i><token></i> into the output stream. Useful when trying to write difficult characters such as %, {, }, <i>etc.</i> in messages, for example:</p>
----------------------------	---

`\iow_now:Nx \g_my_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>
-----------------------------	---

143 Wrapping lines in output

<hr/> <code>\iow_wrap:xnnnN</code> <hr/>	<code>\iow_wrap:xnnnN</code> $\{\langle text \rangle\}$ $\{\langle run-on text \rangle\}$ $\{\langle run-on length \rangle\}$ $\{\langle set up \rangle\}$ $\langle function \rangle$
Updated: 2011-09-21 <hr/>	<p>This function will wrap the $\langle text \rangle$ to a fixed number of characters per line. At the start of each line which is wrapped, the $\langle run-on text \rangle$ will be inserted. The line length targeted will be the value of <code>\l_iow_line_length_int</code> minus the $\langle run-on length \rangle$. The later value should be the number of characters in the $\langle run-on text \rangle$. Additional functions may be added to the wrapping by using the $\langle set up \rangle$, which is executed before the wrapping takes place. The result of the wrapping operation is passed as a braced argument to the $\langle function \rangle$, which will typically be a wrapper around a writing operation. Within the $\langle text \rangle$,</p> <ul style="list-style-type: none"> • <code>\\</code> may be used to force a new line, • <code>\ </code> may be used to represent a forced space (for example after a control sequence), • <code>\#</code>, <code>\%</code>, <code>\{</code>, <code>\}</code>, <code>\~</code> may be used to represent the corresponding character, • <code>\iow_indent:n</code> may be used to indent a part of the message. <p>Both the wrapping process and the subsequent write operation will perform x-type expansion. For this reason, material which is to be written “as is” should be given as the argument to <code>\token_to_str:N</code> or <code>\tl_to_str:n</code> (as appropriate) within the $\langle text \rangle$. The output of <code>\iow_wrap:xnnnN</code> (<i>i.e.</i> the argument passed to the $\langle function \rangle$) will consist of characters of category code 12 (other) and 10 (space) only. This means that the output will <i>not</i> expand further when written to a file.</p>
<hr/> <code>\iow_indent:n</code> <hr/>	<code>\iow_indent:n</code> $\{\langle text \rangle\}$
New: 2011-09-21 <hr/>	<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 <hr/>	

144 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.

145 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.

146 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.

147 Contextual information for messages

<hr/> <code>\msg_line_context</code> ☆ <hr/>	<code>\msg_line_context:</code> Prints the current line number when a message is given, and thus suitable for giving context to messages. The number itself is preceded by the text <code>on line</code> .
<hr/> <code>\msg_line_number</code> ☆ <hr/>	<code>\msg_line_number:</code> Prints the current line number when a message is given.
<hr/> <code>\c_msg_return_text_tl</code> <hr/>	Standard text to indicate that the user should try pressing <code><return></code> to continue. The standard definition reads: <code>Try typing <return> to proceed.</code> <code>If that doesn't work, type X <return> to quit.</code>
<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: <code>More errors will almost certainly follow:</code> <code>the LaTeX run should be aborted.</code>
<hr/> <code>\msg_fatal_text:n</code> ☆ <hr/>	<code>\msg_fatal_text:n {<module>}</code> Produces the standard text: <code>Fatal <module> error</code> 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: <code>Critical <module> error</code> 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> error</code> 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.

148 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).

149 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 }
```

150 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.

151 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.

152 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</code> $\{\langle error\ message\rangle\}$
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 154, 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`.

153 Creating keys

```
\keys_define:nn {<module>} {<keyval list>}
```

Parses the *<keyval list>* and defines the keys listed there for *<module>*. The *<module>* name should be a text value, but there are no restrictions on the nature of the text. In practice the *<module>* should be chosen to be unique to the module in question (unless deliberately adding keys to an existing module).

The *<keyval list>* should consist of one or more key names along with an associated key *property*. The properties of a key determine how it acts. The individual properties are described in the following text; a typical use of `\keys_define:nn` might read

```

\keys_define:nn { mymodule }
{
  keyname .code:n = Some~code~using~#1,
  keyname .value_required:
}

```

where the properties of the key begin from the `.` after the key name.

The various properties available take either no arguments at all, or require exactly one argument. This is indicated in the name of the property using an argument specification. In the following discussion, each property is illustrated attached to an arbitrary *<key>*, which when used may be supplied with a *<value>*. All key *definitions* are local.

```
<key> .bool_set:N = <boolean>
```

Defines *<key>* to set *<boolean>* to *<value>* (which must be either `true` or `false`). If the variable does not exist, it will be created at the point that the key is set up. The *<boolean>* will be assigned locally.

<hr/> .bool_gset:N <hr/>	<p>$\langle key \rangle$.bool_gset:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to $\langle value \rangle$ (which must be either true or false). If the variable does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned globally.</p>
<hr/> .bool_set_inverse:N <hr/> <div>New: 2011-08-28</div> <hr/>	<p>$\langle key \rangle$.bool_set_inverse:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to the logical inverse of $\langle value \rangle$ (which must be either true or false). If the $\langle boolean \rangle$ does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned locally.</p> <p>This property is experimental.</p>
<hr/> .bool_gset_inverse:N <hr/>	<p>$\langle key \rangle$.bool_gset_inverse:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to the logical inverse of $\langle value \rangle$ (which must be either true or false). If the $\langle boolean \rangle$ does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned globally.</p> <p>This property is experimental.</p>
<hr/> .choice: <hr/>	<p>$\langle key \rangle$.choice:</p> <p>Sets $\langle key \rangle$ to act as a choice key. Each valid choice for $\langle key \rangle$ must then be created, as discussed in section 155.</p>
<hr/> .choices:nn <hr/> <div>New: 2011-08-21</div> <hr/>	<p>$\langle key \rangle$.choices:nn $\langle choices \rangle$ $\langle code \rangle$</p> <p>Sets $\langle key \rangle$ to act as a choice key, and defines a series $\langle choices \rangle$ which are implemented using the $\langle code \rangle$. Inside $\langle code \rangle$, $\backslash l_keys_choice_tl$ will be the name of the choice made, and $\backslash l_keys_choice_int$ will be the position of the choice in the list of $\langle choices \rangle$ (indexed from 0). Choices are discussed in detail in section 155.</p> <p>This property is experimental.</p>
<hr/> .choice_code:n .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 155.</p>
<hr/> .clist_set:N .clist_set:c <hr/> <div>New: 2011/09/11</div> <hr/>	<p>$\langle key \rangle$.clist_set:N = $\langle comma\ list\ variable \rangle$</p> <p>Defines $\langle key \rangle$ to locally set $\langle comma\ list\ variable \rangle$ to $\langle value \rangle$. Spaces around commas and empty items will be stripped. If the variable does not exist, it will be created at the point that the key is set up.</p>
<hr/> .clist_gset:N .clist_gset:c <hr/> <div>New: 2011/09/11</div> <hr/>	<p>$\langle key \rangle$.clist_gset:N = $\langle comma\ list\ variable \rangle$</p> <p>Defines $\langle key \rangle$ to globally set $\langle comma\ list\ variable \rangle$ to $\langle value \rangle$. Spaces around commas and empty items will be stripped. If the variable does not exist, it will be created at the point that the key is set up.</p>

<u>.code:n</u>	<u>$\langle key \rangle$.code:n = $\langle code \rangle$</u>
<u>.code:x</u>	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.
<u>.default:n</u>	<u>$\langle key \rangle$.default:n = $\langle default \rangle$</u>
<u>.default:v</u>	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:
	<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>$\langle key \rangle$.dim_set:N = $\langle dimension \rangle$</u>
<u>.dim_set:c</u>	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.
<u>.dim_gset:N</u>	<u>$\langle key \rangle$.dim_gset:N = $\langle dimension \rangle$</u>
<u>.dim_gset:c</u>	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.
<u>.fp_set:N</u>	<u>$\langle key \rangle$.fp_set:N = $\langle floating point \rangle$</u>
<u>.fp_set:c</u>	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.
<u>.fp_gset:N</u>	<u>$\langle key \rangle$.fp_gset:N = $\langle floating point \rangle$</u>
<u>.fp_gset:c</u>	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.

<hr/> <hr/> <code>.generate_choices:n</code>	<code><key> .generate_choices:n = {<list>}</code>	This property will mark <code><key></code> as a multiple choice key, and will use the <code><list></code> to define the choices. The <code><list></code> should consist of a comma-separated list of choice names. Each choice will be set up to execute <code><code></code> as set using <code>.choice_code:n</code> (or <code>.choice_code:x</code>). Choices are discussed in detail in section 155.
<hr/> <code>.int_set:N</code> <hr/> <code>.int_set:c</code>	<code><key> .int_set:N = <integer></code>	Defines <code><key></code> to set <code><integer></code> to <code><value></code> (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The <code><integer></code> will be assigned locally.
<hr/> <code>.int_gset:N</code> <hr/> <code>.int_gset:c</code>	<code><key> .int_gset:N = <integer></code>	Defines <code><key></code> to set <code><integer></code> to <code><value></code> (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The <code><integer></code> will be assigned globally.
<hr/> <code>.meta:n</code> <hr/> <code>.meta:x</code>	<code><key> .meta:n = {<keyval list>}</code>	Makes <code><key></code> a meta-key, which will set <code><keyval list></code> in one go. If <code><key></code> is given with a value at the time the key is used, then the value will be passed through to the subsidiary <code><keys></code> for processing (as #1).
<hr/> <code>.multichoice:</code> <hr/> <small>New: 2011-08-21</small>	<code><key> .multichoice:</code>	Sets <code><key></code> to act as a multiple choice key. Each valid choice for <code><key></code> must then be created, as discussed in section 155. This property is experimental.
<hr/> <code>.multichoice:nn</code> <hr/> <small>New: 2011-08-21</small>	<code><key> .multichoice:nn <choices> <code></code>	Sets <code><key></code> to act as a multiple choice key, and defines a series <code><choices></code> which are implemented using the <code><code></code> . Inside <code><code></code> , <code>\l_keys_choice_tl</code> will be the name of the choice made, and <code>\l_keys_choice_int</code> will be the position of the choice in the list of <code><choices></code> (indexed from 0). Choices are discussed in detail in section 155. This property is experimental.
<hr/> <code>.skip_set:N</code> <hr/> <code>.skip_set:c</code>	<code><key> .skip_set:N = <skip></code>	Defines <code><key></code> to set <code><skip></code> to <code><value></code> (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The <code><skip></code> will be assigned locally.
<hr/> <code>.skip_gset:N</code> <hr/> <code>.skip_gset:c</code>	<code><key> .skip_gset:N = <skip></code>	Defines <code><key></code> to set <code><skip></code> to <code><value></code> (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The <code><skip></code> will be assigned globally.

<hr/> <code>.tl_set:N</code> <hr/>	<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.

154 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.

155 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.

156 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.

157 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>*.

158 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>*.

159 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.

160 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.

161 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.

162 Floating-point variables

<code>\fp_new:N</code>	<code>\fp_new:N <floating point variable></code>
------------------------	--

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

Creates a new *<floating point variable>* or raises an error if the name is already taken. The declaration global. The *<floating point>* will initially be set to `+0.000000000e0` (the zero floating point).

<code>\fp_const:Nn</code>	<code>\fp_const:Nn <floating point variable> {<value>}</code>
---------------------------	---

<code>\fp_const:cn</code>	
---------------------------	--

Creates a new constant *<floating point variable>* or raises an error if the name is already taken. The value of the *<floating point variable>* will be set globally to the *<value>*.

<code>\fp_set_eq:NN</code>	<code>\fp_set_eq:NN <fp var1> <fp var2></code>
----------------------------	--

<code>\fp_set_eq:(cN Nc cc)</code>	
------------------------------------	--

Sets the value of *<floating point variable1>* equal to that of *<floating point variable2>*. This assignment is restricted to the current `TeX` 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 T _E X 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 T _E X 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.

163 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.

164 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.

165 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.

<code>\fp_compare:nTF</code>	<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.

166 Unary floating-point operations

The unary operations alter the value stored within an `fp` variable.

<code>\fp_abs:N</code>	<code>\fp_abs:N \langle floating point variable \rangle</code>
<code>\fp_abs:c</code>	Converts the $\langle floating point variable \rangle$ to its absolute value, assigning the result within the current TeX group.

<code>\fp_gabs:N</code>	<code>\fp_gabs:N \langle floating point variable \rangle</code>
<code>\fp_gabs:c</code>	Converts the $\langle floating point variable \rangle$ to its absolute value, assigning the result globally.

<code>\fp_neg:N</code>	<code>\fp_neg:N \langle floating point variable \rangle</code>
<code>\fp_neg:c</code>	Reverse the sign of the $\langle floating point variable \rangle$, assigning the result within the current TeX group.

<code>\fp_gneg:N</code>	<code>\fp_gneg:N \langle floating point variable \rangle</code>
<code>\fp_gneg:c</code>	Reverse the sign of the $\langle floating point variable \rangle$, assigning the result globally.

167 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/> <hr/>	<hr/>
<code>\fp_add:Nn</code> <code>\fp_add:cn</code>	<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/> <hr/>	<hr/>
<code>\fp_gadd:Nn</code> <code>\fp_gadd:cn</code>	<code>\fp_gadd:Nn <floating point> {<value>}</code> Adds the <i><value></i> to the <i><floating point></i> , making the assignment globally.
<hr/> <hr/>	<hr/>
<code>\fp_sub:Nn</code> <code>\fp_sub:cn</code>	<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/> <hr/>	<hr/>
<code>\fp_gsub:Nn</code> <code>\fp_gsub:cn</code>	<code>\fp_gsub:Nn <floating point> {<value>}</code> Subtracts the <i><value></i> from the <i><floating point></i> , making the assignment globally.
<hr/> <hr/>	<hr/>
<code>\fp_mul:Nn</code> <code>\fp_mul:cn</code>	<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/> <hr/>	<hr/>
<code>\fp_gmul:Nn</code> <code>\fp_gmul:cn</code>	<code>\fp_gmul:Nn <floating point> {<value>}</code> Multiplies the <i><floating point></i> by the <i><value></i> , making the assignment globally.
<hr/> <hr/>	<hr/>
<code>\fp_div:Nn</code> <code>\fp_div:cn</code>	<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/> <hr/>	<hr/>
<code>\fp_gdiv:Nn</code> <code>\fp_gdiv:cn</code>	<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.

168 Floating-point power operations

<hr/> <hr/>	<hr/>
<code>\fp_pow:Nn</code> <code>\fp_pow:cn</code>	<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.

169 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.

170 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.

171 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.

172 Notes on the floating point unit

As calculation of the elemental transcendental functions is computationally expensive compared to storage of results, after calculating a trigonometric function, exponent, *etc.* the module stored the result for reuse. Thus the performance of the module for repeated operations, most probably trigonometric functions, should be much higher than if the values were re-calculated every time they were needed.

Anyone with experience of programming floating point calculations will know that this is a complex area. The aim of the unit is to be accurate enough for the likely applications in a typesetting context. The arithmetic operations are therefore intended to provide ten digit accuracy with the last digit accurate to ± 1 . The elemental transcendental functions may not provide such high accuracy in every case, although the design aim has been to provide 10 digit accuracy for cases likely to be relevant in typesetting situations. A good overview of the challenges in this area can be found in J.-M. Muller,

Elementary functions: algorithms and implementation, 2nd edition, Birkhäuser Boston, New York, USA, 2006.

The internal representation of numbers is tuned to the needs of the underlying \TeX system. This means that the format is somewhat different from that used in, for example, computer floating point units. Programming in \TeX makes it most convenient to use a radix 10 system, using \TeX `count` registers for storage and taking advantage where possible of delimited arguments.

Part XXIII

The l3`luatex` package

LuaTeX-specific functions

173 Breaking out to Lua

The LuaTeX engine provides access to the Lua programming language, and with it access to the “internals” of TeX. In order to use this within the framework provided here, a family of functions is available. When used with pdfTeX or XeTeX these will raise an error: use `\luatex_if_engine:T` to avoid this. Details of coding the LuaTeX engine are detailed in the LuaTeX manual.

<code>\lua_now:n</code>	★	<code>\lua_now:n {⟨token list⟩}</code>
-------------------------	---	--

<code>\lua_now:x</code>	★	
-------------------------	---	--

The `⟨token list⟩` is first tokenized by TeX, which will include converting line ends to spaces in the usual TeX manner and which respects currently-applicable TeX category codes. The resulting `⟨Lua input⟩` is passed to the Lua interpreter for processing. Each `\lua_now:n` block is treated by Lua as a separate chunk. The Lua interpreter will execute the `⟨Lua input⟩` immediately, and in an expandable manner.

TeXhackers note: `\lua_now:x` is the LuaTeX primitive `\directlua` renamed.

<code>\lua_shipout:n</code>		<code>\lua_shipout:x {⟨token list⟩}</code>
-----------------------------	--	--

<code>\lua_shipout:x</code>		
-----------------------------	--	--

The `⟨token list⟩` is first tokenized by TeX, which will include converting line ends to spaces in the usual TeX manner and which respects currently-applicable TeX category codes. The resulting `⟨Lua input⟩` is passed to the Lua interpreter when the current page is finalised (*i.e.* at shipout). Each `\lua_shipout:n` block is treated by Lua as a separate chunk. The Lua interpreter will execute the `⟨Lua input⟩` during the page-building routine: no TeX expansion of the `⟨Lua input⟩` will occur at this stage.

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

<code>\lua_shipout_x:n</code> <code>\lua_shipout_x:x</code>	<code>\lua_shipout:n {⟨token list⟩}</code> <p>The <i>⟨token list⟩</i> is first tokenized by \TeX, which will include converting line ends to spaces in the usual \TeX manner and which respects currently-applicable \TeX category codes. The resulting <i>⟨Lua input⟩</i> is passed to the Lua interpreter when the current page is finalised (<i>i.e.</i> at shipout). Each <code>\lua_shipout:n</code> block is treated by Lua as a separate chunk. The Lua interpreter will execute the <i>⟨Lua input⟩</i> during the page-building routine: the <i>⟨Lua input⟩</i> is expanded during this process in addition to any expansion when the argument was read. This makes these functions suitable for including material finalised during the page building process (such as the page number).</p>
--	--

\TeX hackers note: `\lua_shipout_x:n` is the \LaTeX primitive `\latelua` named using the \LaTeX 3 scheme.

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

174 Category code tables

As well as providing methods to break out into Lua, there are places where additional \LaTeX 3 functions are provided by the \LaTeX engine. In particular, \LaTeX provides category code tables. These can be used to ensure that a set of category codes are in force in a more robust way than is possible with other engines. These are therefore used by `\ExplSyntaxOn` and `\ExplSyntaxOff` when using the \LaTeX engine.

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

<code>\cctab_gset:Nn</code>	<code>\cctab_gset:Nn ⟨category code table⟩ {⟨category code set up⟩}</code> <p>Sets the <i>⟨category code table⟩</i> to apply the category codes which apply when the prevailing regime is modified by the <i>⟨category code set up⟩</i>. Thus within a standard code block the starting point will be the code applied by <code>\c_code_cctab</code>. The assignment of the table is global: the underlying primitive does not respect grouping.</p>
-----------------------------	---

<code>\cctab_begin:N</code>	<code>\cctab_begin:N ⟨category code table⟩</code> <p>Switches the category codes in force to those stored in the <i>⟨category code table⟩</i>. The prevailing codes before the function is called are added to a stack, for use with <code>\cctab_end:</code>.</p>
-----------------------------	---

<code>\cctab_end</code>	<code>\cctab_end:</code> <p>Ends the scope of a <i>⟨category code table⟩</i> started using <code>\cctab_begin:N</code>, retuning the codes to those in force before the matching <code>\cctab_begin:N</code> was used.</p>
-------------------------	---

<code>\c_code_cctab</code>	<p>Category code table for the code environment. This does not include setting the behaviour of the line-end character, which is only altered by <code>\ExplSyntaxOn</code>.</p>
----------------------------	--

<hr/> <hr/> <code>\c_document_cctab</code> <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

175 Bootstrap code

```
1 <*initex | package>
```

175.1 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>

```

175.2 Package-specific code

The package starts by identifying itself: the information itself is taken from the SVN Id string at the start of the source file.

```

46 <*package>
47 \ProvidesPackage{l3bootstrap}
48 [%
49   \ExplFileDate\space v\ExplFileVersion\space
50   L3 Experimental bootstrap code%
51 ]
52 </package>

```

For LuaTeX the functionality of the `\pdfstrcmp` primitive needs to be provided: the `pdftexmcds` package is used to do this if necessary. At present, there is also a need to deal with some low-level allocation stuff that could usefully be added to `lualatex.ini`. As it is currently not, load Heiko Oberdiek's `luatex` package instead.

```

53 <*package>
54 \def\@tempa%
55 {%

```

```

56 \def\@tempa{}%
57 \RequirePackage{luatex}%
58 \RequirePackage{pdfsync}%
59 \let\pdfsync\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
`\ExplSyntaxOn` redefined in `expl3` for the package and in `l3final` for the format.

```

67 \begin{package}
68 \protected\def\ExplSyntaxOff
69 {
70 \catcode 9 = \the\catcode 9\relax
71 \catcode 32 = \the\catcode 32\relax
72 \catcode 34 = \the\catcode 34\relax
73 \catcode 38 = \the\catcode 38\relax
74 \catcode 58 = \the\catcode 58\relax
75 \catcode 94 = \the\catcode 94\relax
76 \catcode 95 = \the\catcode 95\relax
77 \catcode 124 = \the\catcode 124\relax
78 \catcode 126 = \the\catcode 126\relax
79 \endlinechar = \the\endlinechar\relax
80 \chardef\csname\detokenize{l_expl_status_bool}\endcsname = 0 \relax
81 }
82 \protected\def\ExplSyntaxOn
83 {
84 \catcode 9 = 9 \relax
85 \catcode 32 = 9 \relax
86 \catcode 34 = 12 \relax
87 \catcode 58 = 11 \relax
88 \catcode 94 = 7 \relax
89 \catcode 95 = 11 \relax
90 \catcode 124 = 12 \relax
91 \catcode 126 = 10 \relax
92 \endlinechar = 32 \relax
93 \chardef\csname\detokenize{l_expl_status_bool}\endcsname = 1 \relax
94 }
95 \end{package}

```

(End definition for `\ExplSyntaxOff` and `\ExplSyntaxOn`. These functions are documented on page

6.)

`\l_expl_status_bool` The status for experimental code syntax: this is off at present. This code is used by both
the package and the format.

```

96 \expandafter\chardef\csname\detokenize{l_expl_status_bool}\endcsname = 0 \relax
(End definition for \l_expl_status_bool. This function is documented on page ??.)

```

175.3 Dealing with package-mode meta-data

`\GetIdInfo` Functions for collecting up meta-data from the SVN information used by the L^AT_EX3 Project.

`\GetIdInfoFull`

`\GetIdInfoAuxI` 97 `<*package>`

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

`\GetIdInfoAuxIII` 99 `{`

`\GetIdInfoAuxCVS` 100 `\begingroup`

`\GetIdInfoAuxSVN` 101 `\catcode 32 = 10 \relax`

102 `\GetIdInfoAuxI`

103 `}`

104 `\protected\def\GetIdInfoAuxI$#1$#2%`

105 `{`

106 `\def\tempa{#1}%`

107 `\def\tempb{Id}%`

108 `\ifx\tempa\tempb`

109 `\def\tempa`

110 `{%`

111 `\endgroup`

112 `\def\ExplFileName{9999/99/99}%`

113 `\def\ExplFileDescription{#2}%`

114 `\def\ExplFileName{[unknown name]}%`

115 `\def\ExplFileVersion{999}%`

116 `%`

117 `\else`

118 `\def\tempa`

119 `{%`

120 `\endgroup`

121 `\GetIdInfoAuxII$#1$#2}%`

122 `%`

123 `\fi`

124 `\tempa`

125 `}`

126 `\protected\def\GetIdInfoAuxII$#1 #2.#3 #4 #5 #6 #7 #8$#9%`

127 `{%`

128 `\def\ExplFileName{#2}%`

129 `\def\ExplFileVersion{#4}%`

130 `\def\ExplFileDescription{#9}%`

131 `\GetIdInfoAuxIII#5\relax#3\relax#5\relax#6\relax`

132 `}`

133 `\protected\def\GetIdInfoAuxIII#1#2#3#4#5#6\relax`

134 `{%`

135 `\ifx#5/%`

136 `\expandafter\GetIdInfoAuxCVS`

137 `\else`

138 `\expandafter\GetIdInfoAuxSVN`

139 `\fi`

140 `}`

141 `\protected\def\GetIdInfoAuxCVS#1,v\relax#2\relax#3\relax`

142 `{\def\ExplFileName{#2}}`


```

143 \protected\def\GetIdInfoAuxSVN#1\relax#2-#3-#4\relax#5Z\relax
144   {\def\ExplFileDate{#2/#3/#4}}
145 \</package>
      (End definition for \GetIdInfo. This function is documented on page ??.)

```

\ProvidesExplPackage For other packages and classes building on this one it is convenient not to need
 \ProvidesExplClass \ExplSyntaxOn each time.
 \ProvidesExplFile

```

146 <*package>
147 \protected\def\ProvidesExplPackage#1#2#3#4%
148   {%
149     \ProvidesPackage{#1}[#2 v#3 #4]%
150     \ExplSyntaxOn
151   }
152 \protected\def\ProvidesExplClass#1#2#3#4%
153   {%
154     \ProvidesClass{#1}[#2 v#3 #4]%
155     \ExplSyntaxOn
156   }
157 \protected\def\ProvidesExplFile#1#2#3#4%
158   {%
159     \ProvidesFile{#1}[#2 v#3 #4]%
160     \ExplSyntaxOn
161   }
162 \</package>

```

(End definition for \ProvidesExplPackage, \ProvidesExplClass, and \ProvidesExplFile. These functions are documented on page 6.)

\@pushfilename The idea here is to use L^AT_EX 2_ε's \@pushfilename and \@popfilename to track the
 \@popfilename current syntax status. This can be achieved by saving the current status flag at each
 push to a stack, then recovering it at the pop stage and checking if the code environment
 should still be active.

```

163 <*package>
164 \edef\@pushfilename
165   {%
166     \edef\expandafter\noexpand
167     \csname\detokenize{l_expl_status_stack_tl}\endcsname
168     {%
169       \noexpand\ifodd\expandafter\noexpand
170       \csname\detokenize{l_expl_status_bool}\endcsname
171       1%
172       \noexpand\else
173       0%
174       \noexpand\fi
175       \expandafter\noexpand
176       \csname\detokenize{l_expl_status_stack_tl}\endcsname
177     }%
178     \ExplSyntaxOff
179     \unexpanded\expandafter{\@pushfilename}%
180   }

```

```

181 \edef\@popfilename
182 {%
183   \unexpanded\expandafter{\@popfilename}%
184   \noexpand\if a\expandafter\noexpand\csname
185     \detokenize{l_expl_status_stack_tl}\endcsname a%
186     \ExplSyntaxOff
187   \noexpand\else
188     \noexpand\expandafter
189     \expandafter\noexpand\csname
190       \detokenize{expl_status_pop:w}\endcsname
191       \expandafter\noexpand\csname
192         \detokenize{l_expl_status_stack_tl}\endcsname
193         \noexpand\@nil
194   \noexpand\fi
195 }
196 \</package>

```

(End definition for \@pushfilename and \@popfilename. These functions are documented on page ??.)

`\l_expl_status_stack_tl` As expl3 itself cannot be loaded with the code environment already active, at the end of the package `\ExplSyntaxOff` can safely be called.

```

197 \< *package>
198 \@namedef{\detokenize{l_expl_status_stack_tl}}{0}
199 \< /package>

```

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

`\expl_status_pop:w` The pop auxiliary function removes the first item from the stack, saves the rest of the stack and then does the test. As `\ExplSyntaxOff` is already defined as a protected macro, there is no need for `\noexpand` here.

```

200 \< *package>
201 \expandafter\edef\csname\detokenize{expl_status_pop:w}\endcsname#1#2\@nil
202 {%
203   \def\expandafter\noexpand
204     \csname\detokenize{l_expl_status_stack_tl}\endcsname{#2}%
205   \noexpand\ifodd#1\space
206     \noexpand\expandafter\noexpand\ExplSyntaxOn
207   \noexpand\else
208     \noexpand\expandafter\ExplSyntaxOff
209   \noexpand\fi
210 }
211 \< /package>

```

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

We want the expl3 bundle to be loaded “as one”; this command is used to ensure that one of the 13 packages isn’t loaded on its own.

```

212 \< *package>
213 \expandafter\protected\expandafter\def
214   \csname\detokenize{package_check_loaded_expl:}\endcsname
215   {%

```

```

216 \@ifpackageloaded{expl3}
217 {}
218 {%
219 \PackageError{expl3}
220 {Cannot load the expl3 modules separately}
221 {%
222 The expl3 modules cannot be loaded separately;\MessageBreak
223 please \string\usepackage\string{expl3\string} instead.
224 }%
225 }%
226 }
227 \</package>

```

175.4 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

```

175.5 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

```

175.6 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>

```

176 l3names implementation

```

331 <*initex | package>
332 <*package>
333 \ProvidesExplPackage
334   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
335 </package>

```

The code here simply renames all of the primitives to new, internal, names. In format mode, it also deletes all of the existing names (although some do come back later).

`\tex_undefined:D` This function does not exist at all, but is the name used by the plain T_EX format for an undefined function. So it should be marked here as “taken”.

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

The `\let` primitive is renamed by hand first as it is essential for the entire process to follow. This also uses `\global`, as that way we avoid leaving an unneeded csname in the hash table.

```

336 \let \tex_global:D \global
337 \let \tex_let:D \let

```

Everything is inside a (rather long) group, which keeps `\name_primitive:NN` trapped.

```

338 \begingroup

```

`\name_primitive:NN` A temporary function to actually do the renaming. This also allows the original names to be removed in format mode.

```

339 \long \def \name_primitive:NN #1#2
340 {
341   \tex_global:D \tex_let:D #2 #1
342 <*initex>
343   \tex_global:D \tex_let:D #1 \tex_undefined:D
344 </initex>
345 }

```

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

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

```

346 \name_primitive:NN \tex_space:D
347 \name_primitive:NN \tex_italiccor:D
348 \name_primitive:NN \tex_hyphen:D

```

Now all the other primitives.

```

349 \name_primitive:NN \tex_let:D
350 \name_primitive:NN \tex_def:D
351 \name_primitive:NN \tex_edef:D
352 \name_primitive:NN \tex_gdef:D
353 \name_primitive:NN \tex_xdef:D
354 \name_primitive:NN \tex_chardef:D
355 \name_primitive:NN \tex_countdef:D
356 \name_primitive:NN \tex_dimendef:D
357 \name_primitive:NN \tex_skipdef:D
358 \name_primitive:NN \tex_muskipdef:D

```

359	\name_primitive:NN \mathchardef	\tex_mathchardef:D
360	\name_primitive:NN \toksdef	\tex_toksdef:D
361	\name_primitive:NN \futurelet	\tex_futurelet:D
362	\name_primitive:NN \advance	\tex_advance:D
363	\name_primitive:NN \divide	\tex_divide:D
364	\name_primitive:NN \multiply	\tex_multiply:D
365	\name_primitive:NN \font	\tex_font:D
366	\name_primitive:NN \fam	\tex_fam:D
367	\name_primitive:NN \global	\tex_global:D
368	\name_primitive:NN \long	\tex_long:D
369	\name_primitive:NN \outer	\tex_outer:D
370	\name_primitive:NN \setlanguage	\tex_setlanguage:D
371	\name_primitive:NN \globaldefs	\tex_globaldefs:D
372	\name_primitive:NN \afterassignment	\tex_afterassignment:D
373	\name_primitive:NN \aftergroup	\tex_aftergroup:D
374	\name_primitive:NN \expandafter	\tex_expandafter:D
375	\name_primitive:NN \noexpand	\tex_noexpand:D
376	\name_primitive:NN \begingroup	\tex_begingroup:D
377	\name_primitive:NN \endgroup	\tex_endgroup:D
378	\name_primitive:NN \halign	\tex_halign:D
379	\name_primitive:NN \valign	\tex_valign:D
380	\name_primitive:NN \cr	\tex_cr:D
381	\name_primitive:NN \crrcr	\tex_crrcr:D
382	\name_primitive:NN \noalign	\tex_noalign:D
383	\name_primitive:NN \omit	\tex_omit:D
384	\name_primitive:NN \span	\tex_span:D
385	\name_primitive:NN \tabskip	\tex_tabskip:D
386	\name_primitive:NN \everycr	\tex_everycr:D
387	\name_primitive:NN \if	\tex_if:D
388	\name_primitive:NN \ifcase	\tex_ifcase:D
389	\name_primitive:NN \ifcat	\tex_ifcat:D
390	\name_primitive:NN \ifnum	\tex_ifnum:D
391	\name_primitive:NN \ifodd	\tex_ifodd:D
392	\name_primitive:NN \ifdim	\tex_ifdim:D
393	\name_primitive:NN \ifeof	\tex_ifeof:D
394	\name_primitive:NN \ifhbox	\tex_ifhbox:D
395	\name_primitive:NN \ifvbox	\tex_ifvbox:D
396	\name_primitive:NN \ifvoid	\tex_ifvoid:D
397	\name_primitive:NN \ifx	\tex_ifx:D
398	\name_primitive:NN \iffalse	\tex_iffalse:D
399	\name_primitive:NN \iftrue	\tex_iftrue:D
400	\name_primitive:NN \ifhmode	\tex_ifhmode:D
401	\name_primitive:NN \ifmmode	\tex_ifmmode:D
402	\name_primitive:NN \ifvmode	\tex_ifvmode:D
403	\name_primitive:NN \ifinner	\tex_ifinner:D
404	\name_primitive:NN \else	\tex_else:D
405	\name_primitive:NN \fi	\tex_fi:D
406	\name_primitive:NN \or	\tex_or:D
407	\name_primitive:NN \immediate	\tex_immediate:D
408	\name_primitive:NN \closeout	\tex_closeout:D

409	\name_primitive:NN \openin	\tex_openin:D
410	\name_primitive:NN \openout	\tex_openout:D
411	\name_primitive:NN \read	\tex_read:D
412	\name_primitive:NN \write	\tex_write:D
413	\name_primitive:NN \closein	\tex_closein:D
414	\name_primitive:NN \newlinechar	\tex_newlinechar:D
415	\name_primitive:NN \input	\tex_input:D
416	\name_primitive:NN \endinput	\tex_endinput:D
417	\name_primitive:NN \inputlineno	\tex_inputlineno:D
418	\name_primitive:NN \errmessage	\tex_errmessage:D
419	\name_primitive:NN \message	\tex_message:D
420	\name_primitive:NN \show	\tex_show:D
421	\name_primitive:NN \showthe	\tex_showthe:D
422	\name_primitive:NN \showbox	\tex_showbox:D
423	\name_primitive:NN \showlists	\tex_showlists:D
424	\name_primitive:NN \errhelp	\tex_errhelp:D
425	\name_primitive:NN \errorcontextlines	\tex_errorcontextlines:D
426	\name_primitive:NN \tracingcommands	\tex_tracingcommands:D
427	\name_primitive:NN \tracinglostchars	\tex_tracinglostchars:D
428	\name_primitive:NN \tracingmacros	\tex_tracingmacros:D
429	\name_primitive:NN \tracingonline	\tex_tracingonline:D
430	\name_primitive:NN \tracingoutput	\tex_tracingoutput:D
431	\name_primitive:NN \tracingpages	\tex_tracingpages:D
432	\name_primitive:NN \tracingparagraphs	\tex_tracingparagraphs:D
433	\name_primitive:NN \tracingrestores	\tex_tracingrestores:D
434	\name_primitive:NN \tracingstats	\tex_tracingstats:D
435	\name_primitive:NN \pausing	\tex_pausing:D
436	\name_primitive:NN \showboxbreadth	\tex_showboxbreadth:D
437	\name_primitive:NN \showboxdepth	\tex_showboxdepth:D
438	\name_primitive:NN \batchmode	\tex_batchmode:D
439	\name_primitive:NN \errorstopmode	\tex_errorstopmode:D
440	\name_primitive:NN \nonstopmode	\tex_nonstopmode:D
441	\name_primitive:NN \scrollmode	\tex_scrollmode:D
442	\name_primitive:NN \end	\tex_end:D
443	\name_primitive:NN \csname	\tex_csname:D
444	\name_primitive:NN \endcsname	\tex_endcsname:D
445	\name_primitive:NN \ignorespaces	\tex_ignorespaces:D
446	\name_primitive:NN \relax	\tex_relax:D
447	\name_primitive:NN \the	\tex_the:D
448	\name_primitive:NN \mag	\tex_mag:D
449	\name_primitive:NN \language	\tex_language:D
450	\name_primitive:NN \mark	\tex_mark:D
451	\name_primitive:NN \topmark	\tex_topmark:D
452	\name_primitive:NN \firstmark	\tex_firstmark:D
453	\name_primitive:NN \botmark	\tex_botmark:D
454	\name_primitive:NN \splitfirstmark	\tex_splitfirstmark:D
455	\name_primitive:NN \splitbotmark	\tex_splitbotmark:D
456	\name_primitive:NN \fontname	\tex_fontname:D
457	\name_primitive:NN \escapechar	\tex_escapechar:D
458	\name_primitive:NN \endlinechar	\tex_endlinechar:D

459	\name_primitive:NN \mathchoice	\tex_mathchoice:D
460	\name_primitive:NN \delimiter	\tex_delimiter:D
461	\name_primitive:NN \mathaccent	\tex_mathaccent:D
462	\name_primitive:NN \mathchar	\tex_mathchar:D
463	\name_primitive:NN \mskip	\tex_mskip:D
464	\name_primitive:NN \radical	\tex_radical:D
465	\name_primitive:NN \vcenter	\tex_vcenter:D
466	\name_primitive:NN \mkern	\tex_mkern:D
467	\name_primitive:NN \above	\tex_above:D
468	\name_primitive:NN \abovewithdelims	\tex_abovewithdelims:D
469	\name_primitive:NN \atop	\tex_atop:D
470	\name_primitive:NN \atopwithdelims	\tex_atopwithdelims:D
471	\name_primitive:NN \over	\tex_over:D
472	\name_primitive:NN \overwithdelims	\tex_overwithdelims:D
473	\name_primitive:NN \displaystyle	\tex_displaystyle:D
474	\name_primitive:NN \textstyle	\tex_textstyle:D
475	\name_primitive:NN \scriptstyle	\tex_scriptstyle:D
476	\name_primitive:NN \scriptscriptstyle	\tex_scriptscriptstyle:D
477	\name_primitive:NN \nonscript	\tex_nonscript:D
478	\name_primitive:NN \eqno	\tex_eqno:D
479	\name_primitive:NN \leqno	\tex_leqno:D
480	\name_primitive:NN \abovedisplayshortskip	\tex_abovedisplayshortskip:D
481	\name_primitive:NN \abovedisplayskip	\tex_abovedisplayskip:D
482	\name_primitive:NN \belowdisplayshortskip	\tex_belowdisplayshortskip:D
483	\name_primitive:NN \belowdisplayskip	\tex_belowdisplayskip:D
484	\name_primitive:NN \displaywidowpenalty	\tex_displaywidowpenalty:D
485	\name_primitive:NN \displayindent	\tex_displayindent:D
486	\name_primitive:NN \displaywidth	\tex_displaywidth:D
487	\name_primitive:NN \everydisplay	\tex_everydisplay:D
488	\name_primitive:NN \predisplaysize	\tex_predisplaysize:D
489	\name_primitive:NN \predisplaypenalty	\tex_predisplaypenalty:D
490	\name_primitive:NN \postdisplaypenalty	\tex_postdisplaypenalty:D
491	\name_primitive:NN \mathbin	\tex_mathbin:D
492	\name_primitive:NN \mathclose	\tex_mathclose:D
493	\name_primitive:NN \mathinner	\tex_mathinner:D
494	\name_primitive:NN \mathop	\tex_mathop:D
495	\name_primitive:NN \displaylimits	\tex_displaylimits:D
496	\name_primitive:NN \limits	\tex_limits:D
497	\name_primitive:NN \nolimits	\tex_nolimits:D
498	\name_primitive:NN \mathopen	\tex_mathopen:D
499	\name_primitive:NN \mathord	\tex_mathord:D
500	\name_primitive:NN \mathpunct	\tex_mathpunct:D
501	\name_primitive:NN \mathrel	\tex_mathrel:D
502	\name_primitive:NN \overline	\tex_overline:D
503	\name_primitive:NN \underline	\tex_underline:D
504	\name_primitive:NN \left	\tex_left:D
505	\name_primitive:NN \right	\tex_right:D
506	\name_primitive:NN \binoppenalty	\tex_binoppenalty:D
507	\name_primitive:NN \relpenalty	\tex_relpenalty:D
508	\name_primitive:NN \delimitershortfall	\tex_delimitershortfall:D

509	\name_primitive:NN \delimiterfactor	\tex_delimiterfactor:D
510	\name_primitive:NN \nulldelimiterspace	\tex_nulldelimiterspace:D
511	\name_primitive:NN \everymath	\tex_everymath:D
512	\name_primitive:NN \mathsurround	\tex_mathsurround:D
513	\name_primitive:NN \medmuskip	\tex_medmuskip:D
514	\name_primitive:NN \thinmuskip	\tex_thinmuskip:D
515	\name_primitive:NN \thickmuskip	\tex_thickmuskip:D
516	\name_primitive:NN \scriptspace	\tex_scriptspace:D
517	\name_primitive:NN \noboundary	\tex_noboundary:D
518	\name_primitive:NN \accent	\tex_accent:D
519	\name_primitive:NN \char	\tex_char:D
520	\name_primitive:NN \discretionary	\tex_discretionary:D
521	\name_primitive:NN \hfil	\tex_hfil:D
522	\name_primitive:NN \hfilneg	\tex_hfilneg:D
523	\name_primitive:NN \hfill	\tex_hfill:D
524	\name_primitive:NN \hskip	\tex_hskip:D
525	\name_primitive:NN \hss	\tex_hss:D
526	\name_primitive:NN \vfil	\tex_vfil:D
527	\name_primitive:NN \vfilneg	\tex_vfilneg:D
528	\name_primitive:NN \vfill	\tex_vfill:D
529	\name_primitive:NN \vskip	\tex_vskip:D
530	\name_primitive:NN \vss	\tex_vss:D
531	\name_primitive:NN \unskip	\tex_unskip:D
532	\name_primitive:NN \kern	\tex_kern:D
533	\name_primitive:NN \unkern	\tex_unkern:D
534	\name_primitive:NN \hrule	\tex_hrule:D
535	\name_primitive:NN \vrule	\tex_vrule:D
536	\name_primitive:NN \leaders	\tex_leaders:D
537	\name_primitive:NN \cleaders	\tex_cleaders:D
538	\name_primitive:NN \xleaders	\tex_xleaders:D
539	\name_primitive:NN \lastkern	\tex_lastkern:D
540	\name_primitive:NN \lastskip	\tex_lastskip:D
541	\name_primitive:NN \indent	\tex_indent:D
542	\name_primitive:NN \par	\tex_par:D
543	\name_primitive:NN \noindent	\tex_noindent:D
544	\name_primitive:NN \vadjust	\tex_vadjust:D
545	\name_primitive:NN \baselineskip	\tex_baselineskip:D
546	\name_primitive:NN \lineskip	\tex_lineskip:D
547	\name_primitive:NN \lineskiplimit	\tex_lineskiplimit:D
548	\name_primitive:NN \clubpenalty	\tex_clubpenalty:D
549	\name_primitive:NN \widowpenalty	\tex_widowpenalty:D
550	\name_primitive:NN \exhyphenpenalty	\tex_exhyphenpenalty:D
551	\name_primitive:NN \hyphenpenalty	\tex_hyphenpenalty:D
552	\name_primitive:NN \linepenalty	\tex_linepenalty:D
553	\name_primitive:NN \doublehyphendemerits	\tex_doublehyphendemerits:D
554	\name_primitive:NN \finalhyphendemerits	\tex_finalhyphendemerits:D
555	\name_primitive:NN \adjdemerits	\tex_adjdemerits:D
556	\name_primitive:NN \hangafter	\tex_hangafter:D
557	\name_primitive:NN \hangindent	\tex_hangindent:D
558	\name_primitive:NN \parshape	\tex_parshape:D

559	\name_primitive:NN \hsize	\tex_hsize:D
560	\name_primitive:NN \lefthyphenmin	\tex_lefthyphenmin:D
561	\name_primitive:NN \righthyphenmin	\tex_righthyphenmin:D
562	\name_primitive:NN \leftskip	\tex_leftskip:D
563	\name_primitive:NN \rightskip	\tex_rightskip:D
564	\name_primitive:NN \looseness	\tex_looseness:D
565	\name_primitive:NN \parskip	\tex_parskip:D
566	\name_primitive:NN \parindent	\tex_parindent:D
567	\name_primitive:NN \uchyph	\tex_uchyph:D
568	\name_primitive:NN \emergencystretch	\tex_emergencystretch:D
569	\name_primitive:NN \pretolerance	\tex_pretolerance:D
570	\name_primitive:NN \tolerance	\tex_tolerance:D
571	\name_primitive:NN \spaceskip	\tex_spaceskip:D
572	\name_primitive:NN \xspaceskip	\tex_xspaceskip:D
573	\name_primitive:NN \parfillskip	\tex_parfillskip:D
574	\name_primitive:NN \everypar	\tex_everypar:D
575	\name_primitive:NN \prevgraf	\tex_prevgraf:D
576	\name_primitive:NN \spacefactor	\tex_spacefactor:D
577	\name_primitive:NN \shipout	\tex_shipout:D
578	\name_primitive:NN \vsize	\tex_vsize:D
579	\name_primitive:NN \interlinepenalty	\tex_interlinepenalty:D
580	\name_primitive:NN \brokenpenalty	\tex_brokenpenalty:D
581	\name_primitive:NN \topskip	\tex_topskip:D
582	\name_primitive:NN \maxdeadcycles	\tex_maxdeadcycles:D
583	\name_primitive:NN \maxdepth	\tex_maxdepth:D
584	\name_primitive:NN \output	\tex_output:D
585	\name_primitive:NN \deadcycles	\tex_deadcycles:D
586	\name_primitive:NN \pagedepth	\tex_pagedepth:D
587	\name_primitive:NN \pagestretch	\tex_pagestretch:D
588	\name_primitive:NN \pagefilstretch	\tex_pagefilstretch:D
589	\name_primitive:NN \pagefillstretch	\tex_pagefillstretch:D
590	\name_primitive:NN \pagefillllstretch	\tex_pagefillllstretch:D
591	\name_primitive:NN \pageshrink	\tex_pageshrink:D
592	\name_primitive:NN \pagegoal	\tex_pagegoal:D
593	\name_primitive:NN \pagetotal	\tex_pagetotal:D
594	\name_primitive:NN \outputpenalty	\tex_outputpenalty:D
595	\name_primitive:NN \hoffset	\tex_hoffset:D
596	\name_primitive:NN \voffset	\tex_voffset:D
597	\name_primitive:NN \insert	\tex_insert:D
598	\name_primitive:NN \holdinginserts	\tex_holdinginserts:D
599	\name_primitive:NN \floatingpenalty	\tex_floatingpenalty:D
600	\name_primitive:NN \insertpenalties	\tex_insertpenalties:D
601	\name_primitive:NN \lower	\tex_lower:D
602	\name_primitive:NN \moveleft	\tex_moveleft:D
603	\name_primitive:NN \moveright	\tex_moveright:D
604	\name_primitive:NN \raise	\tex_raise:D
605	\name_primitive:NN \copy	\tex_copy:D
606	\name_primitive:NN \lastbox	\tex_lastbox:D
607	\name_primitive:NN \vsplit	\tex_vsplit:D
608	\name_primitive:NN \unhbox	\tex_unhbox:D

609	\name_primitive:NN \unhcopy	\tex_unhcopy:D
610	\name_primitive:NN \unvbox	\tex_unvbox:D
611	\name_primitive:NN \unvcopy	\tex_unvcopy:D
612	\name_primitive:NN \setbox	\tex_setbox:D
613	\name_primitive:NN \hbox	\tex_hbox:D
614	\name_primitive:NN \vbox	\tex_vbox:D
615	\name_primitive:NN \vtop	\tex_vtop:D
616	\name_primitive:NN \prevdepth	\tex_prevdepth:D
617	\name_primitive:NN \badness	\tex_badness:D
618	\name_primitive:NN \hbadness	\tex_hbadness:D
619	\name_primitive:NN \vbadness	\tex_vbadness:D
620	\name_primitive:NN \hfuzz	\tex_hfuzz:D
621	\name_primitive:NN \vfuzz	\tex_vfuzz:D
622	\name_primitive:NN \overfullrule	\tex_overfullrule:D
623	\name_primitive:NN \boxmaxdepth	\tex_boxmaxdepth:D
624	\name_primitive:NN \splitmaxdepth	\tex_splitmaxdepth:D
625	\name_primitive:NN \splittopskip	\tex_splittopskip:D
626	\name_primitive:NN \everyhbox	\tex_everyhbox:D
627	\name_primitive:NN \everyvbox	\tex_everyvbox:D
628	\name_primitive:NN \nullfont	\tex_nullfont:D
629	\name_primitive:NN \textfont	\tex_textfont:D
630	\name_primitive:NN \scriptfont	\tex_scriptfont:D
631	\name_primitive:NN \scriptscriptfont	\tex_scriptscriptfont:D
632	\name_primitive:NN \fontdimen	\tex_fontdimen:D
633	\name_primitive:NN \hyphenchar	\tex_hyphenchar:D
634	\name_primitive:NN \skewchar	\tex_skewchar:D
635	\name_primitive:NN \defaultthyphenchar	\tex_defaultthyphenchar:D
636	\name_primitive:NN \defaultskewchar	\tex_defaultskewchar:D
637	\name_primitive:NN \number	\tex_number:D
638	\name_primitive:NN \romannumeral	\tex_romannumeral:D
639	\name_primitive:NN \string	\tex_string:D
640	\name_primitive:NN \lowercase	\tex_lowercase:D
641	\name_primitive:NN \uppercase	\tex_uppercase:D
642	\name_primitive:NN \meaning	\tex_meaning:D
643	\name_primitive:NN \penalty	\tex_penalty:D
644	\name_primitive:NN \unpenalty	\tex_unpenalty:D
645	\name_primitive:NN \lastpenalty	\tex_lastpenalty:D
646	\name_primitive:NN \special	\tex_special:D
647	\name_primitive:NN \dump	\tex_dump:D
648	\name_primitive:NN \patterns	\tex_patterns:D
649	\name_primitive:NN \hyphenation	\tex_hyphenation:D
650	\name_primitive:NN \time	\tex_time:D
651	\name_primitive:NN \day	\tex_day:D
652	\name_primitive:NN \month	\tex_month:D
653	\name_primitive:NN \year	\tex_year:D
654	\name_primitive:NN \jobname	\tex_jobname:D
655	\name_primitive:NN \everyjob	\tex_everyjob:D
656	\name_primitive:NN \count	\tex_count:D
657	\name_primitive:NN \dimen	\tex_dimen:D
658	\name_primitive:NN \skip	\tex_skip:D

659	\name_primitive:NN	\toks	\tex_toks:D
660	\name_primitive:NN	\muskip	\tex_muskip:D
661	\name_primitive:NN	\box	\tex_box:D
662	\name_primitive:NN	\wd	\tex_wd:D
663	\name_primitive:NN	\ht	\tex_ht:D
664	\name_primitive:NN	\dp	\tex_dp:D
665	\name_primitive:NN	\catcode	\tex_catcode:D
666	\name_primitive:NN	\delcode	\tex_delcode:D
667	\name_primitive:NN	\sfcode	\tex_sfcode:D
668	\name_primitive:NN	\lccode	\tex_lccode:D
669	\name_primitive:NN	\uccode	\tex_uccode:D
670	\name_primitive:NN	\mathcode	\tex_mathcode:D

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

671	\name_primitive:NN	\ifdefined	\etex_ifdefined:D
672	\name_primitive:NN	\ifcsname	\etex_ifcsname:D
673	\name_primitive:NN	\unless	\etex_unless:D
674	\name_primitive:NN	\eTeXversion	\etex_eTeXversion:D
675	\name_primitive:NN	\eTeXrevision	\etex_eTeXrevision:D
676	\name_primitive:NN	\marks	\etex_marks:D
677	\name_primitive:NN	\topmarks	\etex_topmarks:D
678	\name_primitive:NN	\firstmarks	\etex_firstmarks:D
679	\name_primitive:NN	\botmarks	\etex_botmarks:D
680	\name_primitive:NN	\splitfirstmarks	\etex_splitfirstmarks:D
681	\name_primitive:NN	\splitbotmarks	\etex_splitbotmarks:D
682	\name_primitive:NN	\unexpanded	\etex_unexpanded:D
683	\name_primitive:NN	\detokenize	\etex_detokenize:D
684	\name_primitive:NN	\scantokens	\etex_scantokens:D
685	\name_primitive:NN	\showtokens	\etex_showtokens:D
686	\name_primitive:NN	\readline	\etex_readline:D
687	\name_primitive:NN	\tracingassigns	\etex_tracingassigns:D
688	\name_primitive:NN	\tracingscantokens	\etex_tracingscantokens:D
689	\name_primitive:NN	\tracingnesting	\etex_tracingnesting:D
690	\name_primitive:NN	\tracingifs	\etex_tracingifs:D
691	\name_primitive:NN	\currentiflevel	\etex_currentiflevel:D
692	\name_primitive:NN	\currentifbranch	\etex_currentifbranch:D
693	\name_primitive:NN	\currentifttype	\etex_currentifttype:D
694	\name_primitive:NN	\tracinggroups	\etex_tracinggroups:D
695	\name_primitive:NN	\currentgrouplevel	\etex_currentgrouplevel:D
696	\name_primitive:NN	\currentgrouptype	\etex_currentgrouptype:D
697	\name_primitive:NN	\showgroups	\etex_showgroups:D
698	\name_primitive:NN	\showifs	\etex_showifs:D
699	\name_primitive:NN	\interactionmode	\etex_interactionmode:D
700	\name_primitive:NN	\lastnodetype	\etex_lastnodetype:D
701	\name_primitive:NN	\iffontchar	\etex_iffontchar:D
702	\name_primitive:NN	\fontcharht	\etex_fontcharht:D
703	\name_primitive:NN	\fontchardp	\etex_fontchardp:D
704	\name_primitive:NN	\fontcharwd	\etex_fontcharwd:D
705	\name_primitive:NN	\fontcharic	\etex_fontcharic:D

706	\name_primitive:NN \parshapeindent	\etex_parshapeindent:D
707	\name_primitive:NN \parshapelength	\etex_parshapelength:D
708	\name_primitive:NN \parshapedimen	\etex_parshapedimen:D
709	\name_primitive:NN \numexpr	\etex_numexpr:D
710	\name_primitive:NN \dimexpr	\etex_dimexpr:D
711	\name_primitive:NN \glueexpr	\etex_glueexpr:D
712	\name_primitive:NN \muexpr	\etex_muexpr:D
713	\name_primitive:NN \gluestretch	\etex_gluestretch:D
714	\name_primitive:NN \glueshrink	\etex_glueshrink:D
715	\name_primitive:NN \gluestretchorder	\etex_gluestretchorder:D
716	\name_primitive:NN \glueshrinkorder	\etex_glueshrinkorder:D
717	\name_primitive:NN \gluetomu	\etex_gluetomu:D
718	\name_primitive:NN \mutoglua	\etex_mutoglua:D
719	\name_primitive:NN \lastlinefit	\etex_lastlinefit:D
720	\name_primitive:NN \interlinepenalties	\etex_interlinepenalties:D
721	\name_primitive:NN \clubpenalties	\etex_clubpenalties:D
722	\name_primitive:NN \widowpenalties	\etex_widowpenalties:D
723	\name_primitive:NN \displaywidowpenalties	\etex_displaywidowpenalties:D
724	\name_primitive:NN \middle	\etex_middle:D
725	\name_primitive:NN \savinghyphcodes	\etex_savinghyphcodes:D
726	\name_primitive:NN \savingvdiscards	\etex_savingvdiscards:D
727	\name_primitive:NN \pagediscards	\etex_pagediscards:D
728	\name_primitive:NN \splitdiscards	\etex_splitdiscards:D
729	\name_primitive:NN \TeXstate	\etex_TeXstate:D
730	\name_primitive:NN \beginL	\etex_beginL:D
731	\name_primitive:NN \endL	\etex_endL:D
732	\name_primitive:NN \beginR	\etex_beginR:D
733	\name_primitive:NN \endR	\etex_endR:D
734	\name_primitive:NN \predisplaydirection	\etex_predisplaydirection:D
735	\name_primitive:NN \everyeof	\etex_everyeof:D
736	\name_primitive:NN \protected	\etex_protected:D

The newer primitives are more complex: there are an awful lot of them, and we don't use them all at the moment. So the following is selective. In the case of the pdfTeX primitives, we retain pdf at the start of the names *only* for directly PDF-related primitives, as there are a lot of pdfTeX primitives that start \pdf... but are not related to PDF output. These ones related to PDF output.

737	\name_primitive:NN \pdfcreationdate	\pdfTEX_pdfcreationdate:D
738	\name_primitive:NN \pdfcolorstack	\pdfTEX_pdfcolorstack:D
739	\name_primitive:NN \pdfcompresslevel	\pdfTEX_pdfcompresslevel:D
740	\name_primitive:NN \pdfdecimaldigits	\pdfTEX_pdfdecimaldigits:D
741	\name_primitive:NN \pdfhorigin	\pdfTEX_pdfhorigin:D
742	\name_primitive:NN \pdfinfo	\pdfTEX_pdfinfo:D
743	\name_primitive:NN \pdfliteral	\pdfTEX_pdfliteral:D
744	\name_primitive:NN \pdfminorversion	\pdfTEX_pdfminorversion:D
745	\name_primitive:NN \pdfobjcompresslevel	\pdfTEX_pdfobjcompresslevel:D
746	\name_primitive:NN \pdfoutput	\pdfTEX_pdfoutput:D
747	\name_primitive:NN \pdfrestore	\pdfTEX_pdfrestore:D
748	\name_primitive:NN \pdfsave	\pdfTEX_pdfsave:D
749	\name_primitive:NN \pdfsetmatrix	\pdfTEX_pdfsetmatrix:D

```

750 \name_primitive:NN \pdfpkresolution \pdfTeX_pdfpkresolution:D
751 \name_primitive:NN \pdfTeXrevision \pdfTeX_pdfTeXrevision:D
752 \name_primitive:NN \pdfvorigin \pdfTeX_pdfvorigin:D

```

While these are not.

```

753 \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.

```

754 \name_primitive:NN \XeTeXversion \xetex_XeTeXversion:D

```

Primitives from LuaTeX.

```

755 \name_primitive:NN \catcodetable \luaTeX_catcodetable:D
756 \name_primitive:NN \directlua \luaTeX_directlua:D
757 \name_primitive:NN \initcatcodetable \luaTeX_initcatcodetable:D
758 \name_primitive:NN \latelua \luaTeX_latelua:D
759 \name_primitive:NN \luaTeXversion \luaTeX_luaTeXversion:D
760 \name_primitive:NN \savecatcodetable \luaTeX_savecatcodetable:D

```

The job is done: close the group (using the primitive renamed!).

```

761 \tex_endgroup:D

```

L^AT_EX 2_ε will have moved a few primitives, so these are sorted out.

```

762 <*package>
763 \tex_let:D \tex_end:D \@@end
764 \tex_let:D \tex_everydisplay:D \frozen@everydisplay
765 \tex_let:D \tex_everymath:D \frozen@everymath
766 \tex_let:D \tex_hyphen:D \@@hyph
767 \tex_let:D \tex_input:D \@@input
768 \tex_let:D \tex_italic_correction:D \@@italiccorr
769 \tex_let:D \tex_underline:D \@@underline

```

That is also true for the luaTeX package for L^AT_EX 2_ε.

```

770 \tex_let:D \luaTeX_catcodetable:D \luaTeXcatcodetable
771 \tex_let:D \luaTeX_initcatcodetable:D \luaTeXinitcatcodetable
772 \tex_let:D \luaTeX_latelua:D \luaTeXlatelua
773 \tex_let:D \luaTeX_savecatcodetable:D \luaTeXsavecatcodetable
774 </package>
775 </initex | package>

```

177 l3basics implementation

```

776 <*initex | package>
777 <*package>
778 \ProvidesExplPackage
779   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
780 \package_check_loaded_expl:
781 </package>

```

177.1 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: 782 \tex_let:D \if_true:          \tex_iftrue:D
\or:       783 \tex_let:D \if_false:        \tex_iffalse:D
\else:     784 \tex_let:D \or:              \tex_or:D
\fi:       785 \tex_let:D \else:            \tex_else:D
\reverse_if:N 786 \tex_let:D \fi:          \tex_fi:D
\if:w      787 \tex_let:D \reverse_if:N     \etex_unless:D
\if_bool:N 788 \tex_let:D \if:w            \tex_if:D
\if_predicate:w 789 \tex_let:D \if_bool:N    \tex_ifodd:D
\if_charcode:w 790 \tex_let:D \if_predicate:w  \tex_ifodd:D
\if_catcode:w 791 \tex_let:D \if_charcode:w   \tex_if:D
\if_meaning:w 792 \tex_let:D \if_catcode:w   \tex_ifcat:D
              793 \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: 794 \tex_let:D \if_mode_math:      \tex_ifmmode:D
\if_mode_vertical:   795 \tex_let:D \if_mode_horizontal: \tex_ifhmode:D
\if_mode_inner:      796 \tex_let:D \if_mode_vertical:   \tex_ifvmode:D
                    797 \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 798 \tex_let:D \if_cs_exist:N      \etex_ifdefined:D
              799 \tex_let:D \if_cs_exist:w      \etex_ifcsname: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    800 \tex_let:D \exp_after:wN      \tex_expandafter:D
\exp_not:n    801 \tex_let:D \exp_not:N        \tex_noexpand:D
              802 \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 803 \tex_let:D \token_to_meaning:N \tex_meaning:D
\cs:w           804 \tex_let:D \token_to_str:N   \tex_string:D
\cs_end:        805 \tex_let:D \cs:w           \tex_csname:D
\cs_meaning:N   806 \tex_let:D \cs_end:         \tex_endcsname:D
\cs_show:N      807 \tex_let:D \cs_meaning:N    \tex_meaning:D
              808 \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:
809 \tex_let:D \scan_stop:      \tex_relax:D
810 \tex_let:D \group_begin:    \tex_begingroup:D
811 \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
```

```
812 \tex_let:D \if_int_compare:w \tex_ifnum:D
813 \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
```

```
814 \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
```

```
815 \tex_let:D \tex_global:D      \tex_global:D
816 \tex_let:D \tex_long:D       \tex_long:D
817 \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.

```
818 \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
819 \tex_def:D \cs_meaning:c { \exp_args:Nc \cs_meaning:N }
820 \tex_def:D \token_to_str:c { \exp_args:Nc \token_to_str:N }
821 \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 ??.)

177.2 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
822 \tex_let:D \cs_set_nopar:Npn \tex_def:D
823 \tex_let:D \cs_set_nopar:Npx \tex_edef:D
```

```
\cs_set_protected_nopar:Npn
824 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npn
```

```
\cs_set_protected_nopar:Npx
825 { \tex_long:D \cs_set_nopar:Npn }
```

```
\cs_set_protected:Npn
826 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npx
```

```
\cs_set_protected:Npx
827 { \tex_long:D \cs_set_nopar:Npx }
```

```
828 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npn
```

```
829 { \tex_protected:D \cs_set_nopar:Npn }
```

```

830 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npx
831 { \tex_protected:D \cs_set_nopar:Npx }
832 \cs_set_protected_nopar:Npn \cs_set_protected:Npn
833 { \tex_protected:D \tex_long:D \cs_set_nopar:Npn }
834 \cs_set_protected_nopar:Npn \cs_set_protected:Npx
835 { \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 836 \tex_let:D \cs_gset_nopar:Npn \tex_gdef:D
\cs_gset:Npn 837 \tex_let:D \cs_gset_nopar:Npx \tex_xdef:D
\cs_gset:Npx 838 \cs_set_protected_nopar:Npn \cs_gset:Npn
\cs_gset_protected_nopar:Npn 839 { \tex_long:D \cs_gset_nopar:Npn }
\cs_gset_protected_nopar:Npx 840 \cs_set_protected_nopar:Npn \cs_gset:Npx
\cs_gset_protected:Npn 841 { \tex_long:D \cs_gset_nopar:Npx }
\cs_gset_protected:Npx 842 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npn
843 { \tex_protected:D \cs_gset_nopar:Npn }
844 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npx
845 { \tex_protected:D \cs_gset_nopar:Npx }
846 \cs_set_protected_nopar:Npn \cs_gset_protected:Npn
847 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npn }
848 \cs_set_protected_nopar:Npn \cs_gset_protected:Npx
849 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npx }
(End definition for \cs_gset_nopar:Npn. This function is documented on page ??.)

```

177.3 Selecting tokens

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

```

850 \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.

```

\cs_tmp:w 851 \cs_set_protected:Npn \use:x #1
852 {
853   \group_begin:
854   \cs_set:Npx \cs_tmp:w {#1}
855   \exp_after:wN
856   \group_end:
857   \cs_tmp:w
858 }
859 \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:nn 860 \cs_set:Npn \use:n #1 {#1}
\use:nnn 861 \cs_set:Npn \use:nn #1#2 {#1#2}
\use:nnnn 862 \cs_set:Npn \use:nnn #1#2#3 {#1#2#3}
863 \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      864 \cs_set:Npn \use_i:nn  #1#2 {#1}
                  865 \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      866 \cs_set:Npn \use_i:nnnn  #1#2#3 {#1}
\use_iii:nnnn      867 \cs_set:Npn \use_ii:nnnn #1#2#3 {#2}
\use_i_iii:nnnn    868 \cs_set:Npn \use_iii:nnnn #1#2#3 {#3}
\use_i:nnnnn      869 \cs_set:Npn \use_i_iii:nnnn #1#2#3 {#1#2}
\use_ii:nnnnn      870 \cs_set:Npn \use_i:nnnnn  #1#2#3#4 {#1}
\use_iii:nnnnn     871 \cs_set:Npn \use_ii:nnnnn  #1#2#3#4 {#2}
\use_iv:nnnnn      872 \cs_set:Npn \use_iii:nnnnn  #1#2#3#4 {#3}
                  873 \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      874 \cs_set:Npn \use_none_delimit_by_q_nil:w #1 \q_nil { }
\use_none_delimit_by_q_recursion_stop:w 875 \cs_set:Npn \use_none_delimit_by_q_stop:w #1 \q_stop { }
                  876 \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      877 \cs_set:Npn \use_i_delimit_by_q_nil:nw #1#2 \q_nil {#1}
\use_i_delimit_by_q_recursion_stop:nw 878 \cs_set:Npn \use_i_delimit_by_q_stop:nw #1#2 \q_stop {#1}
                  879 \cs_set:Npn \use_i_delimit_by_q_recursion_stop:nw #1#2 \q_recursion_stop {#1}
```

177.4 Gobbling tokens from input

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:n      880 \cs_set:Npn \use_none:n      #1      { }
\use_none:nn     881 \cs_set:Npn \use_none:nn     #1#2    { }
\use_none:nnn    882 \cs_set:Npn \use_none:nnn    #1#2#3   { }
\use_none:nnnn   883 \cs_set:Npn \use_none:nnnn   #1#2#3#4  { }
\use_none:nnnnn  884 \cs_set:Npn \use_none:nnnnn  #1#2#3#4#5  { }
\use_none:nnnnnn 885 \cs_set:Npn \use_none:nnnnnn  #1#2#3#4#5#6  { }
\use_none:nnnnnnn 886 \cs_set:Npn \use_none:nnnnnnn #1#2#3#4#5#6#7  { }
\use_none:nnnnnnnn 887 \cs_set:Npn \use_none:nnnnnnnn #1#2#3#4#5#6#7#8  { }
\use_none:nnnnnnnnn 888 \cs_set:Npn \use_none:nnnnnnnnn #1#2#3#4#5#6#7#8#9 { }
```

177.5 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 \TeX in. Therefore, a simple user interface could be something like

```
\if_meaning:w #1#2 \prg_return_true: \else:
  \if_meaning:w #1#3 \prg_return_true: \else:
    \prg_return_false:
\fi: \fi:
```

Usually, a \TeX programmer would have to insert a number of `\exp_after:wN`s to ensure the state value is returned at exactly the point where the last conditional is finished. However, that obscures the code and forces the \TeX programmer to prove that he/she knows the $2^n - 1$ table. We therefore provide the simpler interface.

`\prg_return_true:` The idea here is that `\int_to_roman:w` will expand fully any `\else:` and the `\fi:` that 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.

`\prg_return_false:`

```
889 \cs_set_nopar:Npn \prg_return_true:
890 { \exp_after:wN \use_i:nn \int_to_roman:w }
891 \cs_set_nopar:Npn \prg_return_false:
892 { \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`

`\prg_new_conditional:Npnn`

`\prg_set_protected_conditional:Npnn`

`\prg_new_protected_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.

```
893 \cs_set_protected:Npn \prg_set_conditional:Npnn #1
894 {
895   \prg_get_parm_aux:nw
896   {
897     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
898     \cs_set:Npn { parm }
899   }
900 }
901 \cs_set_protected:Npn \prg_new_conditional:Npnn #1
902 {
903   \prg_get_parm_aux:nw
904   {
905     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
906     \cs_new:Npn { parm }
907   }
908 }
```

```

909 \cs_set_protected:Npn \prg_set_protected_conditional:Npnn #1
910 {
911   \prg_get_parm_aux:nw{
912     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
913     \cs_set_protected:Npn { parm }
914   }
915 }
916 \cs_set_protected:Npn \prg_new_protected_conditional:Npnn #1
917 {
918   \prg_get_parm_aux:nw
919   {
920     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
921     \cs_new_protected:Npn { parm }
922   }
923 }

```

\prg_set_conditional:Nnn
\prg_new_conditional:Nnn
\prg_set_protected_conditional:Nnn
\prg_new_protected_conditional:Nnn

The user functions for the types automatically inserting the correct parameter text based on the signature. Call aux function after calculating number of arguments, split the base function into name and signature and then use, *e.g.*, \cs_set:Npn to define it with.

```

924 \cs_set_protected:Npn \prg_set_conditional:Nnn #1
925 {
926   \exp_args:Nnf \prg_get_count_aux:nn
927   {
928     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
929     \cs_set:Npn { count }
930   }
931   { \cs_get_arg_count_from_signature:N #1 }
932 }
933 \cs_set_protected:Npn \prg_new_conditional:Nnn #1
934 {
935   \exp_args:Nnf \prg_get_count_aux:nn
936   {
937     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
938     \cs_new:Npn { count }
939   }
940   { \cs_get_arg_count_from_signature:N #1 }
941 }
942
943 \cs_set_protected:Npn \prg_set_protected_conditional:Nnn #1{
944   \exp_args:Nnf \prg_get_count_aux:nn{
945     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
946     \cs_set_protected:Npn {count}
947   }{\cs_get_arg_count_from_signature:N #1}
948 }
949
950 \cs_set_protected:Npn \prg_new_protected_conditional:Nnn #1
951 {
952   \exp_args:Nnf \prg_get_count_aux:nn
953   {

```

```

954         \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
955         \cs_new_protected:Npn {count}
956     }
957     { \cs_get_arg_count_from_signature:N #1 }
958 }

```

\prg_set_eq_conditional:NNn The obvious setting-equal functions.

\prg_new_eq_conditional:NNn

```

959 \cs_set_protected:Npn \prg_set_eq_conditional:NNn #1#2#3
960 { \prg_set_eq_conditional_aux:NNNn \cs_set_eq:cc #1#2 {#3} }
961 \cs_set_protected:Npn \prg_new_eq_conditional:NNn #1#2#3
962 { \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.

```

963 \cs_set:Npn \prg_get_count_aux:nn #1#2 { #1 {#2} }
964 \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.

```

965 \cs_set_protected:Npn \prg_generate_conditional_aux:nnNNnnnn #1#2#3#4#5#6#7#8
966 {
967     \prg_generate_conditional_aux:nnw {#5}
968     {
969         #4 {#1} {#2} {#6} {#8}
970     }
971     #7 , ? , \q_recursion_stop
972 }

```

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.

```

973 \cs_set_protected:Npn \prg_generate_conditional_aux:nnw #1#2#3 ,
974 {
975     \if:w ?#3
976     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
977     \fi:
978     \use:c { prg_generate_#3_form_#1:Nnnnn } #2
979     \prg_generate_conditional_aux:nnw {#1} {#2}
980 }

```

`\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.

```

981 \cs_set_protected:Npn \prg_generate_p_form_parm:Nnnnn #1#2#3#4#5
982 {
983   \exp_args:Nc #1 { #2 _p: #3 } #4
984   {
985     #5 \c_zero
986     \c_true_bool \c_false_bool
987   }
988 }
989 \cs_set_protected:Npn \prg_generate_T_form_parm:Nnnnn #1#2#3#4#5
990 {
991   \exp_args:Nc #1 { #2 : #3 T } #4
992   {
993     #5 \c_zero
994     \use:n \use_none:n
995   }
996 }
997 \cs_set_protected:Npn \prg_generate_F_form_parm:Nnnnn #1#2#3#4#5
998 {
999   \exp_args:Nc #1 { #2 : #3 F } #4
1000   {
1001     #5 \c_zero
1002     { }
1003   }
1004 }
1005 \cs_set_protected:Npn \prg_generate_TF_form_parm:Nnnnn #1#2#3#4#5
1006 {
1007   \exp_args:Nc #1 { #2 : #3 TF } #4
1008   { #5 \c_zero }
1009 }

```

`\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.

```

1010 \cs_set_protected:Npn \prg_generate_p_form_count:Nnnnn #1#2#3#4#5
1011 {
1012   \cs_generate_from_arg_count:cNnn { #2 _p: #3 } #1 {#4}
1013   {
1014     #5 \c_zero
1015     \c_true_bool \c_false_bool
1016   }
1017 }
1018 \cs_set_protected:Npn \prg_generate_T_form_count:Nnnnn #1#2#3#4#5
1019 {
1020   \cs_generate_from_arg_count:cNnn { #2 : #3 T } #1 {#4}
1021   {

```

```

1022         #5 \c_zero
1023         \use:n \use_none:n
1024     }
1025 }
1026 \cs_set_protected:Npn \prg_generate_F_form_count:Nnnnn #1#2#3#4#5
1027 {
1028     \cs_generate_from_arg_count:cNnn { #2 : #3 F } #1 {#4}
1029     {
1030         #5 \c_zero
1031         { }
1032     }
1033 }
1034 \cs_set_protected:Npn \prg_generate_TF_form_count:Nnnnn #1#2#3#4#5
1035 {
1036     \cs_generate_from_arg_count:cNnn { #2 : #3 TF } #1 {#4}
1037     { #5 \c_zero }
1038 }

```

\prg_set_eq_conditional_aux:NNNn

\prg_set_eq_conditional_aux:NNNw

```

1039 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNn #1#2#3#4
1040 { \prg_set_eq_conditional_aux:NNNw #1#2#3#4 , ? , \q_recursion_stop }

```

Manual clist loop over argument #4.

```

1041 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNw #1#2#3#4 ,
1042 {
1043     \if:w ? #4 \scan_stop:
1044     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1045     \fi:
1046     #1
1047     { \exp_args:NNc \cs_split_function:NN #2 { prg_conditional_form_#4:nnn } }
1048     { \exp_args:NNc \cs_split_function:NN #3 { prg_conditional_form_#4:nnn } }
1049     \prg_set_eq_conditional_aux:NNNw #1 {#2} {#3}
1050 }
1051 \cs_set:Npn \prg_conditional_form_p:nnn #1#2#3 { #1 _p : #2 }
1052 \cs_set:Npn \prg_conditional_form_TF:nnn #1#2#3 { #1 : #2 TF }
1053 \cs_set:Npn \prg_conditional_form_T:nnn #1#2#3 { #1 : #2 T }
1054 \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

```

1055 \tex_chardef:D \c_true_bool = 1~
1056 \tex_chardef:D \c_false_bool = 0~

```


177.6 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.

```

1057 \cs_set_nopar:Npn \cs_to_str:N
1058 {
1059   \if:w \int_to_roman:w - '0 \token_to_str:N \ %
1060     \cs_to_str_aux:w
1061     \fi:
1062     \exp_after:wN \use_none:n \token_to_str:N
1063 }
1064 \cs_set_nopar:Npn \cs_to_str_aux:w #1 \use_none:n
1065 { ~ \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`.

```

1066 \group_begin:
1067   \tex_lccode:D '\@ = '\: \scan_stop:
1068   \tex_catcode:D '\@ = 12~
1069   \tex_lowercase:D
1070   {
1071     \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.

```

1072   \cs_set:Npn \cs_split_function:NN #1#2
1073   {
1074     \exp_after:wN \cs_split_function_aux:w
1075     \int_to_roman:w - '\q \cs_to_str:N #1 @ a \q_stop #2
1076   }

```

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.

```

1077   \cs_set:Npn \cs_split_function_aux:w #1 @ #2#3 \q_stop #4
1078   {
1079     \if_meaning:w a #2
1080       \exp_after:wN \use_i:nn
1081     \else:
1082       \exp_after:wN \use_ii:nn
1083     \fi:
1084     { #4 {#1} { } \c_false_bool }
1085     { \cs_split_function_auxii:w #2#3 \q_stop #4 {#1} }
1086   }
1087   \cs_set:Npn \cs_split_function_auxii:w #1 @a \q_stop #2#3
1088   { #2{#3}{#1}\c_true_bool }

```

End of lowercase

```

1089   }

```

`\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.

```

1090   \cs_set:Npn \cs_get_function_name:N #1
1091   { \cs_split_function:NN #1 \use_i:nnn }
1092   \cs_set:Npn \cs_get_function_signature:N #1
1093   { \cs_split_function:NN #1 \use_ii:nnn }

```

177.7 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:`.

```

1094 \prg_set_conditional:Npnn \cs_if_exist:N #1 { p , T , F , TF }
1095 {
1096   \if_meaning:w #1 \scan_stop:
1097   \prg_return_false:
1098   \else:
1099     \if_cs_exist:N #1
1100     \prg_return_true:
1101     \else:
1102       \prg_return_false:
1103     \fi:
1104   \fi:
1105 }
```

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.

```

1106 \prg_set_conditional:Npnn \cs_if_exist:c #1 { p , T , F , TF }
1107 {
1108   \if_cs_exist:w #1 \cs_end:
1109   \exp_after:wN \use_i:nn
1110   \else:
1111     \exp_after:wN \use_ii:nn
1112   \fi:
1113   {
1114     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1115     \prg_return_false:
1116     \else:
1117       \prg_return_true:
1118     \fi:
1119   }
1120   \prg_return_false:
1121 }
```

(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 1122 \prg_set_conditional:Npnn \cs_if_free:N #1 { p , T , F , TF }
1123 {
1124   \if_meaning:w #1 \scan_stop:
```

```

1125     \prg_return_true:
1126   \else:
1127     \if_cs_exist:N #1
1128       \prg_return_false:
1129     \else:
1130       \prg_return_true:
1131     \fi:
1132   \fi:
1133 }
1134 \prg_set_conditional:Npnn \cs_if_free:c #1 { p , T , F , TF }
1135 {
1136   \if_cs_exist:w #1 \cs_end:
1137     \exp_after:wN \use_i:nn
1138   \else:
1139     \exp_after:wN \use_ii:nn
1140   \fi:
1141   {
1142     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1143     \prg_return_true:
1144   \else:
1145     \prg_return_false:
1146   \fi:
1147 }
1148 { \prg_return_true: }
1149 }

```

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

177.8 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_sixteen` The rest are defined in the `l3int` module – at least for the ones that can be defined with `\tex_chardef:D` or `\tex_mathchardef:D`. For other constants the `l3int` module is required but it can't be used until the allocation has been set up properly! The actual allocation mechanism is in `l3alloc` and as \TeX wants to reserve count registers 0–9, the first available one is 10 so we use that for `\c_minus_one`.

`\c_six`

`\c_seven`

`\c_twelve`

```

1150 <*package>
1151 \tex_let:D \c_minus_one \m@ne
1152 </package>
1153 <*initex>
1154 \tex_countdef:D \c_minus_one = 10 ~
1155 \c_minus_one = -1 ~
1156 </initex>
1157 \tex_chardef:D \c_sixteen = 16~
1158 \tex_chardef:D \c_zero = 0~
1159 \tex_chardef:D \c_six = 6~
1160 \tex_chardef:D \c_seven = 7~
1161 \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`

```
1162 \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`.

```
1163 \cs_set_protected_nopar:Npn \iow_log:x
1164 { \tex_immediate:D \tex_write:D \c_minus_one }
1165 \cs_set_protected_nopar:Npn \iow_term:x
1166 { \tex_immediate:D \tex_write:D \c_sixteen }
```

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

`\msg_kernel_error:nxxx` If an internal error occurs before L^AT_EX3 has loaded `l3msg` then the code should issue a
`\msg_kernel_error:nxx` usable if terse error message and halt. This can only happen if a coding error is made by
`\msg_kernel_error:nn` the team, so this is a reasonable response.

```
1167 \cs_set_protected_nopar:Npn \msg_kernel_error:nxxx #1#2#3#4
1168 {
1169   \tex_errmessage:D
1170   {
1171     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!~! ^^J
1172     Argh,~internal~LaTeX3~error! ^^J ^^J
1173     Module ~ #1 , ~ message~name~"#2": ^^J
1174     Arguments~'#3'~and~'#4' ^^J ^^J
1175     This~is~one~for~The~LaTeX3~Project:~bailing~out
1176   }
1177   \tex_end:D
1178 }
1179 \cs_set_protected_nopar:Npn \msg_kernel_error:nxx #1#2#3
1180 { \msg_kernel_error:nxxx {#1} {#2} {#3} { } }
1181 \cs_set_protected_nopar:Npn \msg_kernel_error:nn #1#2
1182 { \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.

```
1183 \cs_set_nopar:Npn \msg_line_context:
1184 { on~line~\tex_the:D \tex_inputlineno:D }
```

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

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

```

1185 \cs_set_protected_nopar:Npn \chk_if_free_cs:N #1
1186 {
1187   \cs_if_free:NF #1
1188   {
1189     \msg_kernel_error:nnxx { kernel } { command-already-defined }
1190     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1191   }
1192 }
1193 \*package
1194 \tex_ifodd:D \l@expl@log@functions@bool
1195 \cs_set_protected_nopar:Npn \chk_if_free_cs:N #1
1196 {
1197   \cs_if_free:NF #1
1198   {
1199     \msg_kernel_error:nnxx { kernel } { command-already-defined }
1200     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1201   }
1202   \iow_log:x { Defining~\token_to_str:N #1~ \msg_line_context: }
1203 }
1204 \fi:
1205 \*package
1206 \cs_set_protected_nopar:Npn \chk_if_free_cs:c
1207 { \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 not exist.

```

1208 \cs_set_protected_nopar:Npn \chk_if_exist_cs:N #1
1209 {
1210   \cs_if_exist:NF #1
1211   {
1212     \msg_kernel_error:nnxx { kernel } { command-not-defined }
1213     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1214   }
1215 }
1216 \cs_set_protected_nopar:Npn \chk_if_exist_cs:c
1217 { \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 ??.)

177.9 More new definitions

`\cs_new_nopar:Npn` Global versions of the above functions.

`\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`

```

1218 \cs_set:Npn \cs_tmp:w #1#2
1219 {
1220   \cs_set_protected_nopar:Npn #1 ##1
1221   {
1222     \chk_if_free_cs:N ##1
1223     #2 ##1
1224   }
1225 }
1226 \cs_tmp:w \cs_new_nopar:Npn      \cs_gset_nopar:Npn
1227 \cs_tmp:w \cs_new_nopar:Npx      \cs_gset_nopar:Npx
1228 \cs_tmp:w \cs_new:Npn            \cs_gset:Npn
1229 \cs_tmp:w \cs_new:Npx            \cs_gset:Npx
1230 \cs_tmp:w \cs_new_protected_nopar:Npn \cs_gset_protected_nopar:Npn
1231 \cs_tmp:w \cs_new_protected_nopar:Npx \cs_gset_protected_nopar:Npx
1232 \cs_tmp:w \cs_new_protected:Npn    \cs_gset_protected:Npn
1233 \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.

```

1234 \cs_set:Npn \cs_tmp:w #1#2
1235 { \cs_new_protected_nopar:Npn #1 { \exp_args:Nc #2 } }
1236 \cs_tmp:w \cs_set_nopar:cpn \cs_set_nopar:Npn
1237 \cs_tmp:w \cs_set_nopar:cpx \cs_set_nopar:Npx
1238 \cs_tmp:w \cs_gset_nopar:cpn \cs_gset_nopar:Npn
1239 \cs_tmp:w \cs_gset_nopar:cpx \cs_gset_nopar:Npx
1240 \cs_tmp:w \cs_new_nopar:cpn \cs_new_nopar:Npn
1241 \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.

```

1242 \cs_tmp:w \cs_set:cpn \cs_set:Npn
1243 \cs_tmp:w \cs_set:cpx \cs_set:Npx
1244 \cs_tmp:w \cs_gset:cpn \cs_gset:Npn
1245 \cs_tmp:w \cs_gset:cpx \cs_gset:Npx
1246 \cs_tmp:w \cs_new:cpn \cs_new:Npn
1247 \cs_tmp:w \cs_new:cpx \cs_new:Npx

```

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

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

```

1248 \cs_tmp:w \cs_set_protected_nopar:cpn \cs_set_protected_nopar:Npn

```

\cs_gset_protected_nopar:cpn
\cs_gset_protected_nopar:cpx
\cs_new_protected_nopar:cpn
\cs_new_protected_nopar:cpx

```

1249 \cs_tmp:w \cs_set_protected_nopar:cpx \cs_set_protected_nopar:Npx
1250 \cs_tmp:w \cs_gset_protected_nopar:cpn \cs_gset_protected_nopar:Npn
1251 \cs_tmp:w \cs_gset_protected_nopar:cpx \cs_gset_protected_nopar:Npx
1252 \cs_tmp:w \cs_new_protected_nopar:cpn \cs_new_protected_nopar:Npn
1253 \cs_tmp:w \cs_new_protected_nopar:cpx \cs_new_protected_nopar:Npx
(End definition for \cs_set_protected_nopar:cpn. This function is documented on page ??.)

```

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

```

\cs_set_protected:cpx
\cs_gset_protected:cpn 1254 \cs_tmp:w \cs_set_protected:cpn \cs_set_protected:Npn
\cs_gset_protected:cpx 1255 \cs_tmp:w \cs_set_protected:cpx \cs_set_protected:Npx
\cs_new_protected:cpn 1256 \cs_tmp:w \cs_gset_protected:cpn \cs_gset_protected:Npn
\cs_new_protected:cpx 1257 \cs_tmp:w \cs_gset_protected:cpx \cs_gset_protected:Npx
1258 \cs_tmp:w \cs_new_protected:cpn \cs_new_protected:Npn
1259 \cs_tmp:w \cs_new_protected:cpx \cs_new_protected:Npx
(End definition for \cs_set_protected:cpn. This function is documented on page ??.)

```

177.10 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. For the definition of \c_space_char{~} to work we need the ~ after the =.

\cs_set_eq:Nc \cs_set_eq:cc \cs_set_eq:NN is long to avoid problems with a literal argument of \par. While \cs_new_eq:NN will probably never be correct with a first argument of \par, define it long in order to throw an “already defined” error rather than “runaway argument”.

```

1260 \cs_new_protected:Npn \cs_set_eq:NN #1 { \tex_let:D #1 =~ }
1261 \cs_new_protected_nopar:Npn \cs_set_eq:cN { \exp_args:Nc \cs_set_eq:NN }
1262 \cs_new_protected_nopar:Npn \cs_set_eq:Nc { \exp_args:NNc \cs_set_eq:NN }
1263 \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 1264 \cs_new_protected:Npn \cs_new_eq:NN #1
\cs_new_eq:Nc 1265 {
\cs_new_eq:cc 1266 \chk_if_free_cs:N #1
1267 \tex_global:D \cs_set_eq:NN #1
1268 }
1269 \cs_new_protected_nopar:Npn \cs_new_eq:cN { \exp_args:Nc \cs_new_eq:NN }
1270 \cs_new_protected_nopar:Npn \cs_new_eq:Nc { \exp_args:NNc \cs_new_eq:NN }
1271 \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 1272 \cs_new_protected_nopar:Npn \cs_gset_eq:NN { \tex_global:D \cs_set_eq:NN }
\cs_gset_eq:Nc 1273 \cs_new_protected_nopar:Npn \cs_gset_eq:Nc { \exp_args:NNc \cs_gset_eq:NN }
\cs_gset_eq:cc 1274 \cs_new_protected_nopar:Npn \cs_gset_eq:cN { \exp_args:Nc \cs_gset_eq:NN }
1275 \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 ??.)

```


177.11 Undefined functions

`\cs_undefine:N` The following function is used to free the main memory from the definition of some
`\cs_undefine:c` function that isn't in use any longer. The `c` variant is careful not to add the control sequence to the hash table if it isn't there yet, and it also avoids nesting \TeX conditionals in case `#1` is unbalanced in this matter.

```

1276 \cs_new_protected_nopar:Npn \cs_undefine:N #1
1277 { \cs_gset_eq:NN #1 \c_undefined:D }
1278 \cs_new_protected_nopar:Npn \cs_undefine:c #1
1279 {
1280   \if_cs_exist:w #1 \cs_end:
1281     \exp_after:wN \use:n
1282   \else:
1283     \exp_after:wN \use_none:n
1284   \fi:
1285   { \cs_gset_eq:cN {#1} \c_undefined:D }
1286 }

```

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

177.12 Defining functions from a given number of arguments

`\cs_get_arg_count_from_signature:N` Counting the number of tokens in the signature, i.e., the number of arguments the func-
`\cs_get_arg_count_from_signature_aux:nnN` tion should take. If there is no signature, we return that there is -1 arguments to signal
`\cs_get_arg_count_from_signature_auxii:w` an error. Otherwise we insert the string 9876543210 after the signature. If the signature is empty, the number we want is 0 so we remove the first nine tokens and return the tenth. Similarly, if the signature is `nnn` we want to remove the nine tokens `nnn987654` and return 3. Therefore, we simply remove the first nine tokens and then return the tenth.

```

1287 \cs_new:Npn \cs_get_arg_count_from_signature:N #1
1288 { \cs_split_function:NN #1 \cs_get_arg_count_from_signature_aux:nnN }
1289 \cs_new:Npn \cs_get_arg_count_from_signature_aux:nnN #1#2#3
1290 {
1291   \if_predicate:w #3
1292     \exp_after:wN \use_i:nn
1293   \else:
1294     \exp_after:wN \use_ii:nn
1295   \fi:
1296   {
1297     \exp_after:wN \cs_get_arg_count_from_signature_auxii:w
1298     \use_none:nnnnnnnn #2 9876543210 \q_stop
1299   }
1300   { -1 }
1301 }
1302 \cs_new:Npn \cs_get_arg_count_from_signature_auxii:w #1#2 \q_stop {#1}

```

A variant form we need right away.

```

1303 \cs_new_nopar:Npn \cs_get_arg_count_from_signature:c
1304 { \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

```
1305 \cs_new_protected:Npn \cs_generate_from_arg_count:NNnn #1#2#3#4
1306 {
1307   \if_case:w \int_eval:w #3 \int_eval_end:
1308     \cs_generate_from_arg_count_aux:nwn {}
1309   \or: \cs_generate_from_arg_count_aux:nwn {##1}
1310   \or: \cs_generate_from_arg_count_aux:nwn {##1##2}
1311   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3}
1312   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4}
1313   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5}
1314   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6}
1315   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7}
1316   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8}
1317   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8##9}
1318   \else:
1319     \cs_generate_from_arg_count_error_msg:Nn #1 {#3}
1320     \use_i:nnn
1321   \fi:
1322   {#2#1}
1323   {#4}
1324 }
1325 \cs_new_protected_nopar:Npn
1326   \cs_generate_from_arg_count_aux:nwn #1 #2 \fi: #3
1327   { \fi: #3 #1 }
```

A variant form we need right away.

```
1328 \cs_new_nopar:Npn \cs_generate_from_arg_count:cNnn
1329   { \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.

```
1330 \cs_new:Npn \cs_generate_from_arg_count_error_msg:Nn #1#2
1331 {
1332   \msg_kernel_error:nnxx { kernel } { bad-number-of-arguments }
1333   { \token_to_str:N #1 } { \int_eval:n {#2} }
1334 }
```

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

177.13 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.

We want to define `\cs_set:Nn` as

```

\cs_set:Nn
\cs_set:Nx
\cs_set_nopar:Nn
\cs_set_nopar:Nx
\cs_set_protected:Nn
\cs_set_protected:Nx
\cs_set_protected_nopar:Nn
\cs_set_protected_nopar:Nx
\cs_gset:Nn
\cs_gset:Nx
\cs_gset_nopar:Nn
\cs_gset_nopar:Nx
\cs_gset_protected:Nn
\cs_gset_protected:Nx
\cs_gset_protected_nopar:Nn
\cs_gset_protected_nopar:Nx

```

In short, to define `\cs_set:Nn` we need just use `\cs_set:Npn`, everything else is the same for each variant. Therefore, we can make it simpler by temporarily defining a function to do this for us.

```

1335 \cs_set:Npn \cs_tmp:w #1#2#3
1336 {
1337   \cs_set_protected:cpx { cs_ #1 : #2 } ##1##2
1338   {
1339     \exp_not:N \cs_generate_from_arg_count:NNnn ##1
1340     \exp_after:wN \exp_not:N \cs:w cs_#1 : #3 \cs_end:
1341     { \exp_not:N \cs_get_arg_count_from_signature:N ##1 }{##2}
1342   }
1343 }

```

Then we define the 32 variants beginning with N.

```

1344 \cs_tmp:w { set } { Nn } { Npn }
1345 \cs_tmp:w { set } { Nx } { Npx }
1346 \cs_tmp:w { set_nopar } { Nn } { Npn }
1347 \cs_tmp:w { set_nopar } { Nx } { Npx }
1348 \cs_tmp:w { set_protected } { Nn } { Npn }
1349 \cs_tmp:w { set_protected } { Nx } { Npx }
1350 \cs_tmp:w { set_protected_nopar } { Nn } { Npn }
1351 \cs_tmp:w { set_protected_nopar } { Nx } { Npx }
1352 \cs_tmp:w { gset } { Nn } { Npn }
1353 \cs_tmp:w { gset } { Nx } { Npx }
1354 \cs_tmp:w { gset_nopar } { Nn } { Npn }
1355 \cs_tmp:w { gset_nopar } { Nx } { Npx }
1356 \cs_tmp:w { gset_protected } { Nn } { Npn }
1357 \cs_tmp:w { gset_protected } { Nx } { Npx }
1358 \cs_tmp:w { gset_protected_nopar } { Nn } { Npn }
1359 \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

```

```

1360 \cs_tmp:w { new } { Nn } { Npn }
1361 \cs_tmp:w { new } { Nx } { Npx }
1362 \cs_tmp:w { new_nopar } { Nn } { Npn }

```

```

1363 \cs_tmp:w { new_nopar } { Nx } { Npx }
1364 \cs_tmp:w { new_protected } { Nn } { Npn }
1365 \cs_tmp:w { new_protected } { Nx } { Npx }
1366 \cs_tmp:w { new_protected_nopar } { Nn } { Npn }
1367 \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}
}

```

```

1368 \cs_set:Npn \cs_tmp:w #1#2#3
1369 {
1370   \cs_set_protected:cpx {cs_#1:#2}##1##2{
1371     \exp_not:N\cs_generate_from_arg_count:cNnn {##1}
1372     \exp_after:wN \exp_not:N \cs:w cs_#1:#3 \cs_end:
1373     { \exp_not:N \cs_get_arg_count_from_signature:c {##1} } {##2}
1374   }
1375 }

```

\cs_set:cn The 32 c variants.

```

\cs_set:cx 1376 \cs_tmp:w { set } { cn } { Npn }
\cs_set_nopar:cn 1377 \cs_tmp:w { set } { cx } { Npx }
\cs_set_nopar:cx 1378 \cs_tmp:w { set_nopar } { cn } { Npn }
\cs_set_protected:cn 1379 \cs_tmp:w { set_nopar } { cx } { Npx }
\cs_set_protected:cx 1380 \cs_tmp:w { set_protected } { cn } { Npn }
\cs_set_protected_nopar:cn 1381 \cs_tmp:w { set_protected } { cx } { Npx }
\cs_set_protected_nopar:cx 1382 \cs_tmp:w { set_protected_nopar } { cn } { Npn }
\cs_gset:cn 1383 \cs_tmp:w { set_protected_nopar } { cx } { Npx }
\cs_gset:cx 1384 \cs_tmp:w { gset } { cn } { Npn }
\cs_gset_nopar:cn 1385 \cs_tmp:w { gset } { cx } { Npx }
\cs_gset_nopar:cx 1386 \cs_tmp:w { gset_nopar } { cn } { Npn }
\cs_gset_protected:cn 1387 \cs_tmp:w { gset_nopar } { cx } { Npx }
\cs_gset_protected:cx 1388 \cs_tmp:w { gset_protected } { cn } { Npn }
\cs_gset_protected_nopar:cn 1389 \cs_tmp:w { gset_protected } { cx } { Npx }
\cs_gset_protected_nopar:cx 1390 \cs_tmp:w { gset_protected_nopar } { cn } { Npn }
1391 \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 1392 \cs_tmp:w { new } { cn } { Npn }
\cs_new_nopar:cn 1393 \cs_tmp:w { new } { cx } { Npx }
\cs_new_nopar:cx 1394 \cs_tmp:w { new_nopar } { cn } { Npn }
\cs_new_protected:cn 1395 \cs_tmp:w { new_nopar } { cx } { Npx }
\cs_new_protected:cx 1396 \cs_tmp:w { new_protected } { cn } { Npn }
\cs_new_protected_nopar:cn 1397 \cs_tmp:w { new_protected } { cx } { Npx }
\cs_new_protected_nopar:cx

```

```

1398 \cs_tmp:w { new_protected_nopar } { cn } { Npn }
1399 \cs_tmp:w { new_protected_nopar } { cx } { Npx }
      (End definition for \cs_new:cn. This function is documented on page ??.)

```

177.14 Checking control sequence equality

```

\cs_if_eq:NN Check if two control sequences are identical.
\cs_if_eq:cN 1400 \prg_new_conditional:Npnn \cs_if_eq:NN #1#2 { p , T , F , TF }
\cs_if_eq:Nc 1401 {
\cs_if_eq:cc 1402   \if_meaning:w #1#2
              1403   \prg_return_true: \else: \prg_return_false: \fi:
              1404 }
1405 \cs_new_nopar:Npn \cs_if_eq_p:cN { \exp_args:Nc \cs_if_eq_p:NN }
1406 \cs_new_nopar:Npn \cs_if_eq:cNTF { \exp_args:Nc \cs_if_eq:NNTF }
1407 \cs_new_nopar:Npn \cs_if_eq:cNT { \exp_args:Nc \cs_if_eq:NNT }
1408 \cs_new_nopar:Npn \cs_if_eq:cNF { \exp_args:Nc \cs_if_eq:NNF }
1409 \cs_new_nopar:Npn \cs_if_eq_p:Nc { \exp_args:NNc \cs_if_eq_p:NN }
1410 \cs_new_nopar:Npn \cs_if_eq:NcTF { \exp_args:NNc \cs_if_eq:NNTF }
1411 \cs_new_nopar:Npn \cs_if_eq:NcT { \exp_args:NNc \cs_if_eq:NNT }
1412 \cs_new_nopar:Npn \cs_if_eq:NcF { \exp_args:NNc \cs_if_eq:NNF }
1413 \cs_new_nopar:Npn \cs_if_eq_p:cc { \exp_args:Ncc \cs_if_eq_p:NN }
1414 \cs_new_nopar:Npn \cs_if_eq:ccTF { \exp_args:Ncc \cs_if_eq:NNTF }
1415 \cs_new_nopar:Npn \cs_if_eq:ccT { \exp_args:Ncc \cs_if_eq:NNT }
1416 \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 ??.)

```

177.15 Diagnostic wrapper functions

```

\kernel_register_show:N
\kernel_register_show:c 1417 \cs_new_nopar:Npn \kernel_register_show:N #1
                        1418 {
                        1419   \cs_if_exist:NTF #1
                        1420   { \tex_showthe:D #1 }
                        1421   {
                        1422     \msg_kernel_error:nxx { kernel } { variable-not-defined }
                        1423     { \token_to_str:N #1 }
                        1424   }
                        1425 }
1426 \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 ??.)

```

177.16 Engine specific definitions

```

\xetex_if_engine: In some cases it will be useful to know which engine we're running. This can all be
\luatex_if_engine: hard-coded for speed.
\pdftex_if_engine: 1427 \cs_new_eq:NN \luatex_if_engine:T \use_none:n
                  1428 \cs_new_eq:NN \luatex_if_engine:F \use:n

```

```

1429 \cs_new_eq:NN \luatex_if_engine:TF \use_ii:nn
1430 \cs_new_eq:NN \pdftex_if_engine:T \use:n
1431 \cs_new_eq:NN \pdftex_if_engine:F \use_none:n
1432 \cs_new_eq:NN \pdftex_if_engine:TF \use_i:nn
1433 \cs_new_eq:NN \xetex_if_engine:T \use_none:n
1434 \cs_new_eq:NN \xetex_if_engine:F \use:n
1435 \cs_new_eq:NN \xetex_if_engine:TF \use_ii:nn
1436 \cs_new_eq:NN \luatex_if_engine_p: \c_false_bool
1437 \cs_new_eq:NN \pdftex_if_engine_p: \c_true_bool
1438 \cs_new_eq:NN \xetex_if_engine_p: \c_false_bool
1439 \cs_if_exist:NT \xetex_XeTeXversion:D
1440 {
1441     \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1442     \cs_set_eq:NN \pdftex_if_engine:F \use:n
1443     \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1444     \cs_set_eq:NN \xetex_if_engine:T \use:n
1445     \cs_set_eq:NN \xetex_if_engine:F \use_none:n
1446     \cs_set_eq:NN \xetex_if_engine:TF \use_i:nn
1447     \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1448     \cs_set_eq:NN \xetex_if_engine_p: \c_true_bool
1449 }
1450 \cs_if_exist:NT \luatex_directlua:D
1451 {
1452     \cs_set_eq:NN \luatex_if_engine:T \use:n
1453     \cs_set_eq:NN \luatex_if_engine:F \use_none:n
1454     \cs_set_eq:NN \luatex_if_engine:TF \use_i:nn
1455     \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1456     \cs_set_eq:NN \pdftex_if_engine:F \use:n
1457     \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1458     \cs_set_eq:NN \luatex_if_engine_p: \c_true_bool
1459     \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1460 }

```

(End definition for `\xetex_if_engine:`, `\luatex_if_engine:`, and `\pdftex_if_engine:`. These functions are documented on page ??.)

177.17 Doing nothing functions

`\prg_do_nothing:` This does not fit anywhere else!

```

1461 \cs_new_nopar:Npn \prg_do_nothing: { }

```

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

177.18 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.

```

1462 \prg_new_conditional:Npnn \str_if_eq:nn #1#2 { p , T , F , TF }
1463 {

```

```

1464 \if_int_compare:w \pdfTeX_strcmp:D { \exp_not:n {#1} } { \exp_not:n {#2} }
1465 = \c_zero
1466 \prg_return_true: \else: \prg_return_false: \fi:
1467 }
1468 \prg_new_conditional:Npnn \str_if_eq:xx #1#2 { p , T , F , TF }
1469 {
1470 \if_int_compare:w \pdfTeX_strcmp:D {#1} {#2} = \c_zero
1471 \prg_return_true: \else: \prg_return_false: \fi:
1472 }

```

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

177.19 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

```

1473 \*deprecated
1474 \cs_new_eq:NN \cs_gnew_nopar:Npn \cs_new_nopar:Npn
1475 \cs_new_eq:NN \cs_gnew:Npn \cs_new:Npn
1476 \cs_new_eq:NN \cs_gnew_protected_nopar:Npn \cs_new_protected_nopar:Npn
1477 \cs_new_eq:NN \cs_gnew_protected:Npn \cs_new_protected:Npn
1478 \cs_new_eq:NN \cs_gnew_nopar:Npx \cs_new_nopar:Npx
1479 \cs_new_eq:NN \cs_gnew:Npx \cs_new:Npx
1480 \cs_new_eq:NN \cs_gnew_protected_nopar:Npx \cs_new_protected_nopar:Npx
1481 \cs_new_eq:NN \cs_gnew_protected:Npx \cs_new_protected:Npx
1482 \cs_new_eq:NN \cs_gnew_nopar:cpn \cs_new_nopar:cpn
1483 \cs_new_eq:NN \cs_gnew:cpn \cs_new:cpn
1484 \cs_new_eq:NN \cs_gnew_protected_nopar:cpn \cs_new_protected_nopar:cpn
1485 \cs_new_eq:NN \cs_gnew_protected:cpn \cs_new_protected:cpn
1486 \cs_new_eq:NN \cs_gnew_nopar:cpx \cs_new_nopar:cpx
1487 \cs_new_eq:NN \cs_gnew:cpx \cs_new:cpx
1488 \cs_new_eq:NN \cs_gnew_protected_nopar:cpx \cs_new_protected_nopar:cpx
1489 \cs_new_eq:NN \cs_gnew_protected:cpx \cs_new_protected:cpx
1490 \*deprecated
1491 \*deprecated
1492 \cs_new_eq:NN \cs_gnew_eq:NN \cs_new_eq:NN
1493 \cs_new_eq:NN \cs_gnew_eq:cN \cs_new_eq:cN
1494 \cs_new_eq:NN \cs_gnew_eq:Nc \cs_new_eq:Nc
1495 \cs_new_eq:NN \cs_gnew_eq:cc \cs_new_eq:cc
1496 \*deprecated
1497 \*deprecated
1498 \cs_new_eq:NN \cs_gundefine:N \cs_undefine:N
1499 \cs_new_eq:NN \cs_gundefine:c \cs_undefine:c
1500 \*deprecated
1501 \*deprecated
1502 \cs_new_eq:NN \group_execute_after:N \group_insert_after:N
1503 \*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

```

1504 \cs_new_eq:NN \c_luatex_is_engine_bool \luatex_if_engine_p:
1505 \cs_new_eq:NN \c_pdftex_is_engine_bool \pdftex_if_engine_p:
1506 \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 TeX conditionals.

```

1507 \cs_set:Npn \use_i_after_fi:nw #1 \fi: { \fi: #1 }
1508 \cs_set:Npn \use_i_after_else:nw #1 \else: #2 \fi: { \fi: #1 }
1509 \cs_set:Npn \use_i_after_or:nw #1 \or: #2 \fi: { \fi: #1 }
1510 \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

```

```

1511 \tex_let:D \cs_set_eq:NwN \tex_let:D

```

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

```

1512 </initex | package>

```

178 l3expan implementation

```

1513 <*initex | package>

```

We start by ensuring that the required packages are loaded.

```

1514 <*package>
1515 \ProvidesExplPackage
1516   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1517 \package_check_loaded_expl:
1518 </package>

```

```

\exp_after:wN
\exp_not:N
\exp_not:n

```

These are defined in `l3basics`.

(End definition for `\exp_after:wN`. This function is documented on page 29.)

178.1 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 178.3. In section 178.2 some common cases are coded by a more direct method for efficiency, typically using calls to `\exp_after:wN`.

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.

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

```
1519 \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.

```
1520 \cs_new:Npn \exp_arg_next:nnn #1#2#3 { #2 \::: { #3 {#1} } }
1521 \cs_new:Npn \exp_arg_next_nobrace:nnn #1#2#3 { #2 \::: { #3 #1 } }
      (End definition for \exp_arg_next:nnn. This function is documented on page ??.)
```

`\:::` The end marker is just another name for the identity function.

```
1522 \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.

```
1523 \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.

```
1524 \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 a control sequence without expansion.

```
1525 \cs_new:Npn \::c #1 \::: #2#3
1526 { \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.

```
1527 \cs_new:Npn \::o #1 \::: #2#3
1528 { \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.

```

1529 \cs_new:Npn \::f #1 \:: #2#3
1530 {
1531   \exp_after:wN \exp_arg_next:nnn
1532   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1533   {#1} {#2}
1534 }
1535 \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.

```

1536 \cs_new_protected:Npn \::x #1 \:: #2#3
1537 {
1538   \cs_set_nopar:Npx \l_exp_tl { {#3} }
1539   \exp_after:wN \exp_arg_next:nnn \l_exp_tl {#1} {#2}
1540 }

```

(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 `csname` from its argument given in braces and then evaluates it as if it was a `V`. The primitive `\romannumeral` sets off an expansion similar to an `f` type expansion, which we will terminate using `\c_zero`. The argument is returned in braces.

```

1541 \cs_new:Npn \::V #1 \:: #2#3
1542 {
1543   \exp_after:wN \exp_arg_next:nnn
1544   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1545   {#1} {#2}
1546 }
1547 \cs_new:Npn \::v # 1\:: #2#3
1548 {
1549   \exp_after:wN \exp_arg_next:nnn
1550   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1551   {#1} {#2}
1552 }

```

(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:`.

```

1553 \cs_new_nopar:Npn \exp_eval_register:N #1
1554 {
1555     \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.

```

1556     \if_meaning:w \scan_stop: #1
1557     \exp_eval_error_msg:w
1558 \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`.

```

1559 \else:
1560     \exp_after:wN \use_i_ii:nnn
1561 \fi:
1562 \exp_after:wN \c_zero \tex_the:D #1
1563 }
1564 \cs_new_nopar:Npn \exp_eval_register:c #1
1565 { \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}

```

```

1566 \cs_new:Npn \exp_eval_error_msg:w #1 \tex_the:D #2
1567 {
1568     \fi:
1569 \fi:

```

```

1570 \msg_expandable_error:n { Erroneous ~ variable ~ #2 used! }
1571 \c_zero
1572 }

```

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

178.2 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 1573 \cs_new:Npn \exp_args:No #1#2 { \exp_after:wN #1 \exp_after:wN {#2} }
\exp_args:NNNo 1574 \cs_new:Npn \exp_args:NNNo #1#2#3
1575 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN {#3} }
1576 \cs_new:Npn \exp_args:NNNo #1#2#3#4
1577 { \exp_after:wN #1 \exp_after:wN#2 \exp_after:wN #3 \exp_after:wN {#4} }

```

(End definition for `\exp_args:No`. This function is documented on page 28.)

`\exp_args:Nc` In l3basics

(End definition for `\exp_args:Nc`. This function is documented on page 26.)

`\exp_args:cc` Here are the functions that turn their argument into csnames but are expandable.

```

\exp_args:NNc 1578 \cs_new:Npn \exp_args:cc #1#2
\exp_args:Ncc 1579 { \cs:w #1 \exp_after:wN \cs_end: \cs:w #2 \cs_end: }
\exp_args:Nccc 1580 \cs_new:Npn \exp_args:NNc #1#2#3
1581 { \exp_after:wN #1 \exp_after:wN #2 \cs:w # 3\cs_end: }
1582 \cs_new:Npn \exp_args:Ncc #1#2#3
1583 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: \cs:w #3 \cs_end: }
1584 \cs_new:Npn \exp_args:Nccc #1#2#3#4
1585 {
1586 \exp_after:wN #1
1587 \cs:w #2 \exp_after:wN \cs_end:
1588 \cs:w #3 \exp_after:wN \cs_end:
1589 \cs:w #4 \cs_end:
1590 }

```

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

`\exp_args:Nf`

```

\exp_args:Nv 1591 \cs_new:Npn \exp_args:Nf #1#2
\exp_args:Nx 1592 { \exp_after:wN #1 \exp_after:wN { \tex_romannumeral:D -'0 #2 } }
1593 \cs_new:Npn \exp_args:Nv #1#2
1594 {
1595 \exp_after:wN #1 \exp_after:wN
1596 { \tex_romannumeral:D \exp_eval_register:c {#2} }
1597 }
1598 \cs_new:Npn \exp_args:Nv #1#2
1599 {
1600 \exp_after:wN #1 \exp_after:wN

```

```

1601     { \tex_romannumeral:D \exp_eval_register:N #2 }
1602   }

```

(End definition for \exp_args:Nf and others. These functions are documented on page 27.)

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

```

\exp_args:NNV 1603 \cs_new:Npn \exp_args:NNf #1#2#3
\exp_args:NNv 1604 {
\exp_args:NNf 1605   \exp_after:wN #1
\exp_args:Ncf 1606   \exp_after:wN #2
\exp_args:Nco 1607   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1608 }
\cs_new:Npn \exp_args:NNv #1#2#3
1609 {
1610   \exp_after:wN #1
1611   \exp_after:wN #2
1612   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1613 }
\cs_new:Npn \exp_args:NNV #1#2#3
1614 {
1615   \exp_after:wN #1
1616   \exp_after:wN #2
1617   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1618 }
\cs_new:Npn \exp_args:Nco #1#2#3
1619 {
1620   \exp_after:wN #1
1621   \cs:w #2 \exp_after:wN \cs_end:
1622   \exp_after:wN {#3}
1623 }
\cs_new:Npn \exp_args:Ncf #1#2#3
1624 {
1625   \exp_after:wN #1
1626   \cs:w #2 \exp_after:wN \cs_end:
1627   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1628 }
\cs_new_nopar:Npn \exp_args:NVV #1#2#3
1629 {
1630   \exp_after:wN #1
1631   \exp_after:wN { \tex_romannumeral:D \exp_after:wN
1632     \exp_eval_register:N \exp_after:wN #2 \exp_after:wN }
1633   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1634 }

```

(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 1640 \cs_new:Npn \exp_args:NNNV #1#2#3#4
\exp_args:NcNo 1641 {
\exp_args:NNNV

```

```

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

```

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

178.3 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`

```
1668 \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 1669 \cs_new_nopar:Npn \exp_args:Nnc { \::n \::c \::: }
\exp_args:Ncx 1670 \cs_new_nopar:Npn \exp_args:Nfo { \::f \::o \::: }
\exp_args:Nfo 1671 \cs_new_nopar:Npn \exp_args:Nff { \::f \::f \::: }
\exp_args:Nff 1672 \cs_new_nopar:Npn \exp_args:Nnf { \::n \::f \::: }
\exp_args:Nnf 1673 \cs_new_nopar:Npn \exp_args:Nno { \::n \::o \::: }
\exp_args:Nno 1674 \cs_new_nopar:Npn \exp_args:NnV { \::n \::V \::: }
\exp_args:NnV 1675 \cs_new_nopar:Npn \exp_args:Noc { \::o \::c \::: }
\exp_args:Nnx 1676 \cs_new_nopar:Npn \exp_args:Noo { \::o \::o \::: }
\exp_args:Noo 1677 \cs_new_protected_nopar:Npn \exp_args:NNx { \::N \::x \::: }
\exp_args:Noc 1678 \cs_new_protected_nopar:Npn \exp_args:Ncx { \::c \::x \::: }
\exp_args:Nox
\exp_args:Nxo
\exp_args:Nxx

```

```

1679 \cs_new_protected_nopar:Npn \exp_args:Nnx { \::n \::x \:: }
1680 \cs_new_protected_nopar:Npn \exp_args:Nox { \::o \::x \:: }
1681 \cs_new_protected_nopar:Npn \exp_args:Nxo { \::x \::o \:: }
1682 \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
\exp_args:NNno
\exp_args:Nnno
\exp_args:Nnnx
\exp_args:Nnox
\exp_args:Nooo
\exp_args:Noox
\exp_args:Nnnc
\exp_args:NNnx
\exp_args:NNoo
\exp_args:NNox

```

```

1683 \cs_new_nopar:Npn \exp_args:NNno { \::N \::n \::o \:: }
1684 \cs_new_nopar:Npn \exp_args:NNoo { \::N \::o \::o \:: }
1685 \cs_new_nopar:Npn \exp_args:Nnnc { \::n \::n \::c \:: }
1686 \cs_new_nopar:Npn \exp_args:Nnno { \::n \::n \::o \:: }
1687 \cs_new_nopar:Npn \exp_args:Nnox { \::o \::o \::o \:: }
1688 \cs_new_protected_nopar:Npn \exp_args:NNnx { \::N \::n \::x \:: }
1689 \cs_new_protected_nopar:Npn \exp_args:NNox { \::N \::o \::x \:: }
1690 \cs_new_protected_nopar:Npn \exp_args:Nnnx { \::n \::n \::x \:: }
1691 \cs_new_protected_nopar:Npn \exp_args:Nnox { \::n \::o \::x \:: }
1692 \cs_new_protected_nopar:Npn \exp_args:Nccx { \::c \::c \::x \:: }
1693 \cs_new_protected_nopar:Npn \exp_args:Ncnx { \::c \::n \::x \:: }
1694 \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 ??.)

178.4 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

```

```

1695 \cs_new:Npn \exp_arg_last_unbraced:nn #1#2 { #2#1 }
1696 \cs_new:Npn \::f_unbraced \::: #1#2
1697 {
1698   \exp_after:wN \exp_arg_last_unbraced:nn
1699   \exp_after:wN { \tex_romannumeral:D -'0 #2 } {#1}
1700 }
1701 \cs_new:Npn \::o_unbraced \::: #1#2
1702 { \exp_after:wN \exp_arg_last_unbraced:nn \exp_after:wN {#2} {#1} }
1703 \cs_new:Npn \::V_unbraced \::: #1#2
1704 {
1705   \exp_after:wN \exp_arg_last_unbraced:nn
1706   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #2 } {#1}
1707 }
1708 \cs_new:Npn \::v_unbraced \::: #1#2
1709 {
1710   \exp_after:wN \exp_arg_last_unbraced:nn
1711   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#2} } {#1}
1712 }

```

(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:Nf
\exp_last_unbraced:No
\exp_last_unbraced:NcV
\exp_last_unbraced:NNV
\exp_last_unbraced:NNo
\exp_last_unbraced:Noo
\exp_last_unbraced:Nfo
\exp_last_unbraced:NNNV
\exp_last_unbraced:NNNo

```

```

1713 \cs_new:Npn \exp_last_unbraced:Nv #1#2

```

```

1714 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:N #2 }
1715 \cs_new:Npn \exp_last_unbraced:Nv #1#2
1716 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:c {#2} }
1717 \cs_new:Npn \exp_last_unbraced:No #1#2 { \exp_after:wN #1 #2 }
1718 \cs_new:Npn \exp_last_unbraced:Nf #1#2
1719 { \exp_after:wN #1 \tex_romannumeral:D -'0 #2 }
1720 \cs_new:Npn \exp_last_unbraced:NcV #1#2#3
1721 {
1722   \exp_after:wN #1
1723   \cs:w #2 \exp_after:wN \cs_end:
1724   \tex_romannumeral:D \exp_eval_register:N #3
1725 }
1726 \cs_new:Npn \exp_last_unbraced:NNV #1#2#3
1727 {
1728   \exp_after:wN #1
1729   \exp_after:wN #2
1730   \tex_romannumeral:D \exp_eval_register:N #3
1731 }
1732 \cs_new:Npn \exp_last_unbraced:NNo #1#2#3
1733 { \exp_after:wN #1 \exp_after:wN #2 #3 }
1734 \cs_new_nopar:Npn \exp_last_unbraced:Nno { \::n \::o_unbraced \:: }
1735 \cs_new_nopar:Npn \exp_last_unbraced:Noo { \::o \::o_unbraced \:: }
1736 \cs_new_nopar:Npn \exp_last_unbraced:Nfo { \::f \::o_unbraced \:: }
1737 \cs_new:Npn \exp_last_unbraced:NNNV #1#2#3#4
1738 {
1739   \exp_after:wN #1
1740   \exp_after:wN #2
1741   \exp_after:wN #3
1742   \tex_romannumeral:D \exp_eval_register:N #4
1743 }
1744 \cs_new:Npn \exp_last_unbraced:NNNo #1#2#3#4
1745 { \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.

```

1746 \cs_new:Npn \exp_last_two_unbraced:Noo #1#2#3
1747 { \exp_after:wN \exp_last_two_unbraced_aux:noN \exp_after:wN {#3} {#2} #1 }
1748 \cs_new:Npn \exp_last_two_unbraced_aux:noN #1#2#3
1749 { \exp_after:wN #3 #2 #1 }

```

(End definition for \exp_last_two_unbraced:Noo. This function is documented on page 29.)

178.5 Preventing expansion

```

\exp_not:o
\exp_not:f 1750 \cs_new:Npn \exp_not:o #1 { \etex_unexpanded:D \exp_after:wN {#1} }
\exp_not:V 1751 \cs_new:Npn \exp_not:f #1
\exp_not:v 1752 { \etex_unexpanded:D \exp_after:wN { \tex_romannumeral:D -'0 #1 } }
1753 \cs_new:Npn \exp_not:V #1
1754 {
1755   \etex_unexpanded:D \exp_after:wN
1756   { \tex_romannumeral:D \exp_eval_register:N #1 }
1757 }
1758 \cs_new:Npn \exp_not:v #1
1759 {
1760   \etex_unexpanded:D \exp_after:wN
1761   { \tex_romannumeral:D \exp_eval_register:c {#1} }
1762 }

```

(End definition for `\exp_not:o`. This function is documented on page 30.)

`\exp_not:c` A helper function.

```

1763 \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.)

178.6 Defining function variants

<pre> \cs_generate_variant:Nn \cs_generate_variant_aux:nnNNn \cs_generate_variant_aux:Nnnw \cs_generate_variant_aux:NNn </pre>	<p>#1 : Base form of a function; e.g., <code>\tl_set:Nn</code></p> <p>#2 : One or more variant argument specifiers; e.g., <code>{Nx,c,cx}</code></p> <p>Test whether the base function is protected or not and define <code>\cs_tmp:w</code> as either <code>\cs_new_nopar:Npx</code> or <code>\cs_new_protected_nopar:Npx</code>, 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 <code>\tl_set:Nn</code> which needs a <code>c</code> variant form, we want the new signature to be <code>cn</code>.</p>
--	--

```

1764 \cs_new_protected:Npn \cs_generate_variant:Nn #1
1765 {
1766   \chk_if_exist_cs:N #1
1767   \cs_generate_variant_aux:N #1
1768   \cs_split_function:NN #1 \cs_generate_variant_aux:nnNNn
1769   #1
1770 }

```

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.

```

1771 \cs_new:Npn \cs_generate_variant_aux:nnNNn #1#2#3#4#5
1772 { \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.

```

1773 \cs_new:Npn \cs_generate_variant_aux:Nnnw #1#2#3#4 ,
1774 {
1775   \if:w ? #4
1776     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1777   \fi:
1778   \exp_args:NNc \cs_generate_variant_aux:NNn
1779   #1
1780   {
1781     #2 : #4
1782     \exp_after:wN \use_i_delimit_by_q_stop:nw
1783     \use_none:nnnnnnnn #4
1784     \use_none:nnnnnnnn
1785     \use_none:nnnnnnnn
1786     \use_none:nnnnnnnn
1787     \use_none:nnnnnnnn
1788     \use_none:nnnnnn
1789     \use_none:nnnnn
1790     \use_none:nnnn
1791     \use_none:nnn
1792     \use_none:nn
1793     \use_none:n
1794     { }
1795     \q_stop
1796     #3
1797   }
1798   {#4}
1799   \cs_generate_variant_aux:Nnnw #1 {#2} {#3}

```

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.

```

1800 \cs_new:Npn \cs_generate_variant_aux:NNn #1 #2 #3
1801 {
1802   \cs_if_free:NTF #2
1803   {
1804     \cs_tmp:w #2 { \exp_not:c { exp_args:N #3 } \exp_not:N #1 }
1805     \cs_generate_internal_variant:n {#3}
1806   }
1807   {
1808     \iow_log:x

```

```

1809         {
1810             Variant~\token_to_str:N #2~%
1811             already~defined;~ not~ changing~ it~on~line~%
1812             \tex_the:D \tex_inputlineno:D
1813         }
1814     }
1815 }

```

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

\cs_generate_variant_aux:N The idea here is to pick up protected parent functions, using the nature of the meaning
\cs_generate_variant_aux:w string that they generate. The test here is almost the same as \tl_if_empty:nTF, but
has to be hard-coded as that function is not yet available and because it has to match
both long and short macros.

```

1816 \group_begin:
1817 \tex_lccode:D '\Z = '\d \scan_stop:
1818 \tex_lccode:D '\? ='\ \ \scan_stop:
1819 \tex_catcode:D '\P = 12 \scan_stop:
1820 \tex_catcode:D '\R = 12 \scan_stop:
1821 \tex_catcode:D '\O = 12 \scan_stop:
1822 \tex_catcode:D '\T = 12 \scan_stop:
1823 \tex_catcode:D '\E = 12 \scan_stop:
1824 \tex_catcode:D '\C = 12 \scan_stop:
1825 \tex_catcode:D '\Z = 12 \scan_stop:
1826 \tex_lowercase:D
1827 {
1828     \group_end:
1829     \cs_new_nopar:Npn \cs_generate_variant_aux:N #1
1830     {
1831         \exp_after:wN \cs_generate_variant_aux:w
1832         \token_to_meaning:N #1
1833         \q_mark \cs_new_protected_nopar:Npx
1834         ? PROTECTEZ
1835         \q_mark \cs_new_nopar:Npx
1836         \q_stop
1837     }
1838     \cs_new:Npn \cs_generate_variant_aux:w
1839     #1 ? PROTECTEZ #2 \q_mark #3 #4 \q_stop
1840     {
1841         \cs_set_eq:NN \cs_tmp:w #3
1842     }
1843 }

```

(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
\cs_generate_internal_variant_aux:N the chars in #1

```

1844 \cs_new_protected:Npn \cs_generate_internal_variant:n #1
1845 {
1846     \cs_if_free:cT { exp_args:N #1 }
1847     {

```

```

1848         \cs_new:cpx { exp_args:N #1 }
1849         { \cs_generate_internal_variant_aux:N #1 : }
1850     }
1851 }

```

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.

```

1852 \cs_new:Npn \cs_generate_internal_variant_aux:N #1
1853 {
1854     \exp_not:c { :: #1 }
1855     \if_meaning:w : #1
1856         \exp_after:wN \use_none:n
1857     \fi:
1858     \cs_generate_internal_variant_aux:N
1859 }

```

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

178.7 Variants which cannot be created earlier

```

\str_if_eq:Vn
\str_if_eq:on
\str_if_eq:nV
\str_if_eq:no
\str_if_eq:VV

```

These cannot come earlier as they need \cs_generate_variant:Nn.

```

1860 \cs_generate_variant:Nn \str_if_eq_p:nn { V , o }
1861 \cs_generate_variant:Nn \str_if_eq_p:nn { nV , no , VV }
1862 \cs_generate_variant:Nn \str_if_eq:nnT { V , o }
1863 \cs_generate_variant:Nn \str_if_eq:nnT { nV , no , VV }
1864 \cs_generate_variant:Nn \str_if_eq:nnF { V , o }
1865 \cs_generate_variant:Nn \str_if_eq:nnF { nV , no , VV }
1866 \cs_generate_variant:Nn \str_if_eq:nnTF { V , o }
1867 \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 ??.)

```

1868 \</initex | package>

```

179 l3prg implementation

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

```

1869 <*initex | package>
1870 <*package>
1871 \ProvidesExplPackage
1872     {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1873 \package_check_loaded_expl:
1874 </package>

```

179.1 Defining a set of conditional functions

These are all defined in l3basics, as they are needed “early”. This is just a reminder that that is the case!

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

```

\prg_set_conditional:Npnn
\prg_new_conditional:Npnn
    \prg_set_protected_conditional:Npnn
    \prg_new_protected_conditional:Npnn
\prg_set_conditional:Nnn
\prg_new_conditional:Nnn
    \prg_set_protected_conditional:Nnn
    \prg_new_protected_conditional:Nnn
\prg_set_eq_conditional:Nnn
\prg_new_eq_conditional:Nnn
    \prg_return_true:
    \prg_return_false:

```

179.2 The boolean data type

`\bool_new:N` Boolean variables have to be initiated when they are created. Other than that there is not much to say here.

```
1875 \cs_new_protected_nopar:Npn \bool_new:N #1 { \cs_new_eq:NN #1 \c_false_bool }
1876 \cs_generate_variant:Nn \bool_new:N { c }
(End definition for \bool_new:N and \bool_new:c. These functions are documented on page ??.)
```

`\bool_set_true:N` Setting is already pretty easy.

```
\bool_set_true:c 1877 \cs_new_protected_nopar:Npn \bool_set_true:N #1
\bool_gset_true:N 1878 { \cs_set_eq:NN #1 \c_true_bool }
\bool_gset_true:c 1879 \cs_new_protected_nopar:Npn \bool_set_false:N #1
\bool_set_false:N 1880 { \cs_set_eq:NN #1 \c_false_bool }
\bool_set_false:c 1881 \cs_new_protected_nopar:Npn \bool_gset_true:N #1
\bool_gset_false:N 1882 { \cs_gset_eq:NN #1 \c_true_bool }
\bool_gset_false:c 1883 \cs_new_protected_nopar:Npn \bool_gset_false:N #1
1884 { \cs_gset_eq:NN #1 \c_false_bool }
1885 \cs_generate_variant:Nn \bool_set_true:N { c }
1886 \cs_generate_variant:Nn \bool_set_false:N { c }
1887 \cs_generate_variant:Nn \bool_gset_true:N { c }
1888 \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 1889 \cs_new_eq:NN \bool_set_eq:NN \cs_set_eq:NN
\bool_set_eq:Nc 1890 \cs_new_eq:NN \bool_set_eq:Nc \cs_set_eq:Nc
\bool_set_eq:cc 1891 \cs_new_eq:NN \bool_set_eq:cN \cs_set_eq:cN
\bool_gset_eq:NN 1892 \cs_new_eq:NN \bool_set_eq:cc \cs_set_eq:cc
\bool_gset_eq:cN 1893 \cs_new_eq:NN \bool_gset_eq:NN \cs_gset_eq:NN
\bool_gset_eq:Nc 1894 \cs_new_eq:NN \bool_gset_eq:Nc \cs_gset_eq:Nc
\bool_gset_eq:cN 1895 \cs_new_eq:NN \bool_gset_eq:cN \cs_gset_eq:cN
1896 \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 1897 \cs_new:Npn \bool_set:Nn #1#2
\bool_gset:cn 1898 { \tex_chardef:D #1 = \bool_if_p:n {#2} }
1899 \cs_new:Npn \bool_gset:Nn #1#2
1900 { \tex_global:D \tex_chardef:D #1 = \bool_if_p:n {#2} }
1901 \cs_generate_variant:Nn \bool_set:Nn { c }
1902 \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.

```
1903 \prg_new_conditional:Npnn \bool_if:N #1 { p , T , F , TF }
1904 {
1905   \if_bool:N #1
1906     \prg_return_true:
```

```

1907     \else:
1908         \prg_return_false:
1909     \fi:
1910 }
1911 \cs_generate_variant:Nn \bool_if_p:N { c }
1912 \cs_generate_variant:Nn \bool_if:NT { c }
1913 \cs_generate_variant:Nn \bool_if:NF { c }
1914 \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
1915 \bool_new:N \l_tmpa_bool
1916 \bool_new:N \g_tmpa_bool

```

(End definition for \l_tmpa_bool and \g_tmpa_bool. These functions are documented on page 36.)

179.3 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 \bool_get_next:N for logical “Not”, && for logical “And” and || for logical “Or”. We shall use the terms \bool_cleanup:N Not, And, Or, Open and Close for these operations.

\bool_choose:NN Any expression is terminated by a Close operation. Evaluation happens from left to right in the following manner using a GetNext function:

- If an Open is seen, start evaluating a new expression using the Eval function and call GetNext again.
- If a Not is seen, insert a negating function (if-even in this case) and call GetNext.
- 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.

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_eval_skip_to_end:Nw **<true>And** Current truth value is true, logical And seen, continue with GetNext to examine truth value of next boolean (sub-)expression.

\bool_eval_skip_to_end_aux:Nw **<false>And** Current truth value is false, logical And seen, stop evaluating the predicates within this sub-expression and break to the nearest Close. Then return **<false>**.

\bool_eval_skip_to_end_aux_ii:Nw **<true>Or** Current truth value is true, logical Or seen, stop evaluating the predicates within this sub-expression and break to the nearest Close. Then return **<true>**.

\bool_eval_skip_to_end_aux_iii:Nw **<false>Or** Current truth value is false, logical Or seen, continue with GetNext to examine truth value of next boolean (sub-)expression.

$\langle true \rangle$ Close Current truth value is true, Close seen, return $\langle true \rangle$.

$\langle false \rangle$ Close Current truth value is false, Close seen, return $\langle false \rangle$.

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 T_EX. We also need to finish this special group before finally returning a `\c_true_bool` or `\c_false_bool` as there might otherwise be something left in front in the input stream. For this we call the Stop operation, denoted simply by a S following the last Close operation.

```

1917 \prg_new_conditional:Npnn \bool_if:n #1 { T , F , TF }
1918 {
1919   \if_predicate:w \bool_if_p:n {#1}
1920   \prg_return_true:
1921   \else:
1922     \prg_return_false:
1923   \fi:
1924 }
1925 \cs_new:Npn \bool_if_p:n #1
1926 {
1927   \group_align_safe_begin:
1928   \bool_get_next:N ( #1 ) S
1929 }
```

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

```

1930 \cs_new:Npn \bool_get_next:N #1
1931 {
1932   \use:c
1933   {
1934     bool_
1935     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1936     :w
1937   }
1938   #1
1939 }
```

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.

```

1940 \cs_new:Npn \bool_get_not_next:N #1
1941 {
1942   \use:c
1943   {
1944     bool_not_
```

```

1945     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1946     :w
1947     }
1948     #1
1949 }

```

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

```

1950 \cs_new:Npn \bool_get_next:NN #1#2 { \bool_get_next:N #2 }
1951 \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.

```

1952 \cs_new:cpn { bool_!:w } #1#2
1953 {
1954     \if_meaning:w ( #2
1955     \exp_after:wN \bool_Not:w
1956     \else:
1957     \if_meaning:w ! #2
1958     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_next:NN
1959     \else:
1960     \exp_after:wN \exp_after:wN \exp_after:wN \bool_Not:N
1961     \fi:
1962     \fi:
1963     #2
1964 }

```

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

```

1965 \cs_new:cpn { bool_not_!:w } #1#2
1966 {
1967     \if_meaning:w ( #2
1968     \exp_after:wN \bool_not_Not:w
1969     \else:
1970     \if_meaning:w ! #2
1971     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_not_next:NN
1972     \else:
1973     \exp_after:wN \exp_after:wN \exp_after:wN \bool_not_Not:N
1974     \fi:
1975     \fi:
1976     #2
1977 }

```

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.

```

1978 \cs_new:Npn \bool_Not:w { \exp_after:wN \int_value:w \bool_get_not_next:N }
1979 \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.

```

1980 \cs_new:Npn \bool_Not:N #1

```



```

1981 {
1982   \exp_after:wN \bool_p:w
1983   \if_meaning:w #1 \c_true_bool
1984   \c_false_bool
1985   \else:
1986   \c_true_bool
1987   \fi:
1988 }
1989 \cs_new:Npn \bool_not_Not:N #1
1990 {
1991   \exp_after:wN \bool_p:w
1992   \if_meaning:w #1 \c_true_bool
1993   \c_true_bool
1994   \else:
1995   \c_false_bool
1996   \fi:
1997 }

```

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 `!(...)`.

```

1998 \cs_new:cpn { bool_(w } #1
1999 { \exp_after:wN \bool_cleanup:N \int_value:w \bool_get_next:N }
2000 \cs_new:cpn { bool_not_(w } #1
2001 { \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.

```

2002 \cs_new:cpn { bool_p:w } { \exp_after:wN \bool_cleanup:N \int_value:w }
2003 \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.

```

2004 \cs_new:Npn \bool_cleanup:N #1
2005 {
2006   \exp_after:wN \bool_choose:NN \exp_after:wN #1
2007   \int_to_roman:w - '\q
2008 }
2009 \cs_new:Npn \bool_not_cleanup:N #1
2010 {
2011   \exp_after:wN \bool_not_choose:NN \exp_after:wN #1
2012   \int_to_roman:w - '\q
2013 }

```

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

```

2014 \cs_new_nopar:Npn \bool_choose:NN #1#2 { \use:c { bool_ #2 _ #1 :w } }
2015 \cs_new_nopar:Npn \bool_not_choose:NN #1#2 { \use:c { bool_not_ #2 _ #1 :w } }

```

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

```

2016 \cs_new_nopar:cpn { bool_&_1:w } & { \bool_get_next:N }
2017 \cs_new_nopar:cpn { bool_|_0:w } | { \bool_get_next:N }
2018 \cs_new_nopar:cpn { bool_not_&_0:w } & { \bool_get_next:N }
2019 \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.

```

2020 \cs_new_nopar:cpn { bool_)_0:w } { \c_false_bool }
2021 \cs_new_nopar:cpn { bool_)_1:w } { \c_true_bool }
2022 \cs_new_nopar:cpn { bool_not_)_0:w } { \c_true_bool }
2023 \cs_new_nopar:cpn { bool_not_)_1:w } { \c_false_bool }
2024 \cs_new_nopar:cpn { bool_S_0:w } { \group_align_safe_end: \c_false_bool }
2025 \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.

```

2026 \cs_new_nopar:cpn { bool_&_0:w } & { \bool_eval_skip_to_end:Nw \c_false_bool }
2027 \cs_new_nopar:cpn { bool_|_1:w } | { \bool_eval_skip_to_end:Nw \c_true_bool }
2028 \cs_new_nopar:cpn { bool_not_&_1:w } & {
2029   { \bool_eval_skip_to_end:Nw \c_false_bool }
2030 \cs_new_nopar:cpn { bool_not_|_0:w } |
2031   { \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.

```
2032 %% (
2033 \cs_new:Npn \bool_eval_skip_to_end:Nw #1#2 )
2034 {
2035   \bool_eval_skip_to_end_aux:Nw #1#2 ( % )
2036   \q_no_value \q_stop
2037   {#2}
2038 }
```

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

```
2039 \cs_new:Npn \bool_eval_skip_to_end_aux:Nw #1#2 ( #3#4 \q_stop #5 % )
2040 {
2041   \quark_if_no_value:NTF #3
2042   {#1}
2043   { \bool_eval_skip_to_end_aux_ii:Nw #1 #5 }
2044 }
```

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!

```
2045 \cs_new:Npn \bool_eval_skip_to_end_aux_ii:Nw #1#2 ( #3 )
2046 { % (
2047   \bool_eval_skip_to_end:Nw #1#3 )
2048 }
```

`\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.

```
2049 \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.

```
2050 \cs_new:Npn \bool_xor_p:nn #1#2
2051 {
2052   \int_compare:nNnTF { \bool_if_p:n {#1} } = { \bool_if_p:n {#2} }
2053   \c_false_bool
2054   \c_true_bool
2055 }
```

179.4 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_while_do:Nn
\bool_until_do:Nn
\bool_until_do:cn
2056 \cs_new:Npn \bool_while_do:Nn #1#2
2057 { \bool_if:NT #1 { #2 \bool_while_do:Nn #1 {#2} } }
2058 \cs_new:Npn \bool_until_do:Nn #1#2
2059 { \bool_if:NF #1 { #2 \bool_until_do:Nn #1 {#2} } }
2060 \cs_generate_variant:Nn \bool_while_do:Nn { c }
2061 \cs_generate_variant:Nn \bool_until_do:Nn { c }
```

`\bool_do_while:Nn` A do-while loop where the body is performed at least once and the boolean is tested after executing the body. Otherwise identical to the above functions.

```
\bool_do_while:cn
\bool_do_while:Nn
\bool_do_until:Nn
\bool_do_until:cn
2062 \cs_new:Npn \bool_do_while:Nn #1#2
2063 { #2 \bool_if:NT #1 { \bool_do_while:Nn #1 {#2} } }
2064 \cs_new:Npn \bool_do_until:Nn #1#2
2065 { #2 \bool_if:NF #1 { \bool_do_until:Nn #1 {#2} } }
2066 \cs_generate_variant:Nn \bool_do_while:Nn { c }
2067 \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
2068 \cs_new:Npn \bool_while_do:nn #1#2
2069 {
2070   \bool_if:nT {#1}
2071   {
2072     #2
2073     \bool_while_do:nn {#1} {#2}
2074   }
2075 }
2076 \cs_new:Npn \bool_do_while:nn #1#2
2077 {
2078   #2
2079   \bool_if:nT {#1} { \bool_do_while:nn {#1} {#2} }
2080 }
2081 \cs_new:Npn \bool_until_do:nn #1#2
2082 {
2083   \bool_if:nF {#1}
2084   {
2085     #2
2086     \bool_until_do:nn {#1} {#2}
2087   }
2088 }
2089 \cs_new:Npn \bool_do_until:nn #1#2
2090 {
2091   #2
2092   \bool_if:nF {#1} { \bool_do_until:nn {#1} {#2} }
2093 }
```

179.5 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.

```
2094 \cs_new:Npn \prg_case_end:nw #1 #2 \q_recursion_stop { \c_zero #1 }
```

`\prg_case_int:nnn` For integer cases, the first task is to fully expand the check condition. After that, a loop is started to compare each possible value and stop if the test is true. The tested value is put at the end to ensure that there is necessarily a match, which will fire the “else” pathway. The leading `\romannumeral` triggers an expansion which is then stopped in `\prg_case_end:nw`.

```
2095 \cs_new:Npn \prg_case_int:nnn #1
2096 {
2097   \tex_romannumeral:D
2098   \exp_args:Nf \prg_case_int_aux:nnn { \int_eval:n {#1} }
2099 }
2100 \cs_new:Npn \prg_case_int_aux:nnn #1 #2 #3
2101 { \prg_case_int_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2102 \cs_new:Npn \prg_case_int_aux:nw #1#2#3
2103 {
2104   \int_compare:nNnTF {#1} = {#2}
2105   { \prg_case_end:nw {#3} }
2106   { \prg_case_int_aux:nw {#1} }
2107 }
```

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

```
\prg_case_dim_aux:nnn
\prg_case_dim_aux:nw
2108 \cs_new:Npn \prg_case_dim:nnn #1
2109 {
2110   \tex_romannumeral:D
2111   \exp_args:Nf \prg_case_dim_aux:nnn { \dim_eval:n {#1} }
2112 }
2113 \cs_new:Npn \prg_case_dim_aux:nnn #1 #2 #3
2114 { \prg_case_dim_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2115 \cs_new:Npn \prg_case_dim_aux:nw #1#2#3
2116 {
2117   \dim_compare:nNnTF {#1} = {#2}
2118   { \prg_case_end:nw {#3} }
2119   { \prg_case_dim_aux:nw {#1} }
2120 }
```

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

```
\prg_case_str:onn
\prg_case_str:xxn
\prg_case_str_aux:nw
\prg_case_str_x_aux:nw
2121 \cs_new:Npn \prg_case_str:nnn #1#2#3
2122 {
2123   \tex_romannumeral:D
2124   \prg_case_str_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
```

```

2125 }
2126 \cs_new:Npn \prg_case_str_aux:nw #1#2#3
2127 {
2128   \str_if_eq:nnTF {#1} {#2}
2129     { \prg_case_end:nw {#3} }
2130     { \prg_case_str_aux:nw {#1} }
2131 }
2132 \cs_generate_variant:Nn \prg_case_str:nnn { o }
2133 \cs_new:Npn \prg_case_str:xxn #1#2#3
2134 {
2135   \tex_romannumeral:D
2136   \prg_case_str_x_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2137 }
2138 \cs_new:Npn \prg_case_str_x_aux:nw #1#2#3
2139 {
2140   \str_if_eq:xxTF {#1} {#2}
2141     { \prg_case_end:nw {#3} }
2142     { \prg_case_str_x_aux:nw {#1} }
2143 }

```

```

\prg_case_tl:Nnn
\prg_case_tl:cnn
\prg_case_tl_aux:Nw

```

Similar again, but this time with some variants.

```

2144 \cs_new:Npn \prg_case_tl:Nnn #1#2#3
2145 {
2146   \tex_romannumeral:D
2147   \prg_case_tl_aux:Nw #1 #2 #1 {#3} \q_recursion_stop
2148 }
2149 \cs_new:Npn \prg_case_tl_aux:Nw #1#2#3
2150 {
2151   \tl_if_eq:NNTF #1 #2
2152     { \prg_case_end:nw {#3} }
2153     { \prg_case_tl_aux:Nw #1 }
2154 }
2155 \cs_generate_variant:Nn \prg_case_tl:Nnn { c }

```

179.6 Producing n copies

This function uses a cascading csname technique by David Kastrup (who else :-)

The idea is to make the input 25 result in first adding five, and then 20 copies of the code to be replicated. The technique uses cascading csnames which means that we start building several csnames so we end up with a list of functions to be called in reverse order. This is important here (and other places) because it means that we can for instance make the function that inserts five copies of something to also hand down ten to the next function in line. This is exactly what happens here: in the example with 25 then the next function is the one that inserts two copies but it sees the ten copies handed down by the previous function. In order to avoid the last function to insert say, 100 copies of the original argument just to gobble them again we define separate functions to be inserted first. These functions also close the expansion of `\int_to_roman:w`, which ensures that `\prg_replicate:nn` only requires two steps of expansion.

```

\prg_replicate:nn
\prg_replicate_aux:N
\prg_replicate_first_aux:N
\prg_replicate_
\prg_replicate_0:n
\prg_replicate_1:n
\prg_replicate_2:n
\prg_replicate_3:n
\prg_replicate_4:n
\prg_replicate_5:n
\prg_replicate_6:n
\prg_replicate_7:n
\prg_replicate_8:n
\prg_replicate_9:n
\prg_replicate_first_~:n
\prg_replicate_first_0:n
\prg_replicate_first_1:n
\prg_replicate_first_2:n
\prg_replicate_first_3:n
\prg_replicate_first_4:n
\prg_replicate_first_5:n
\prg_replicate_first_6:n

```

This function has one flaw though: Since it constantly passes down ten copies of its previous argument it will severely affect the main memory once you start demanding hundreds of thousands of copies. Now I don't think this is a real limitation for any ordinary use, and if necessary, it is possible to write `\prg_replicate:nn{1000}{\prg_replicate:nn{1000}{\code}`. An alternative approach is to create a string of m's with `\int_to_roman:w` which can be done with just four macros but that method has its own problems since it can exhaust the string pool. Also, it is considerably slower than what we use here so the few extra csnames are well spent I would say.

```

2156 \cs_new_nopar:Npn \prg_replicate:nn #1
2157 {
2158   \int_to_roman:w
2159   \exp_after:wN \prg_replicate_first_aux:N
2160   \int_value:w \int_eval:w #1 \int_eval_end:
2161   \cs_end:
2162 }
2163 \cs_new_nopar:Npn \prg_replicate_aux:N #1
2164 { \cs:w prg_replicate_#1 :n \prg_replicate_aux:N }
2165 \cs_new_nopar:Npn \prg_replicate_first_aux:N #1
2166 { \cs:w prg_replicate_first_#1 :n \prg_replicate_aux:N }

```

Then comes all the functions that do the hard work of inserting all the copies.

```

2167 \cs_new_nopar:Npn \prg_replicate_ :n #1 { \cs_end: }
2168 \cs_new:cpn { prg_replicate_0:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } }
2169 \cs_new:cpn { prg_replicate_1:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1 }
2170 \cs_new:cpn { prg_replicate_2:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1 }
2171 \cs_new:cpn { prg_replicate_3:n } #1
2172 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1 }
2173 \cs_new:cpn { prg_replicate_4:n } #1
2174 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1 }
2175 \cs_new:cpn { prg_replicate_5:n } #1
2176 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1 }
2177 \cs_new:cpn { prg_replicate_6:n } #1
2178 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1 }
2179 \cs_new:cpn { prg_replicate_7:n } #1
2180 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1#1 }
2181 \cs_new:cpn { prg_replicate_8:n } #1
2182 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1#1#1 }
2183 \cs_new:cpn { prg_replicate_9:n } #1
2184 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1#1#1#1 }

```

Users shouldn't ask for something to be replicated once or even not at all but...

```

2185 \cs_new:cpn { prg_replicate_first_ -:n } #1 { \c_zero \negative_replication }
2186 \cs_new:cpn { prg_replicate_first_0:n } #1 { \c_zero }
2187 \cs_new:cpn { prg_replicate_first_1:n } #1 { \c_zero #1 }
2188 \cs_new:cpn { prg_replicate_first_2:n } #1 { \c_zero #1#1 }
2189 \cs_new:cpn { prg_replicate_first_3:n } #1 { \c_zero #1#1#1 }
2190 \cs_new:cpn { prg_replicate_first_4:n } #1 { \c_zero #1#1#1#1 }
2191 \cs_new:cpn { prg_replicate_first_5:n } #1 { \c_zero #1#1#1#1#1 }
2192 \cs_new:cpn { prg_replicate_first_6:n } #1 { \c_zero #1#1#1#1#1#1 }
2193 \cs_new:cpn { prg_replicate_first_7:n } #1 { \c_zero #1#1#1#1#1#1#1 }

```

```

2194 \cs_new:cpn { prg_replicate_first_8:n } #1 { \c_zero #1#1#1#1#1#1#1#1 }
2195 \cs_new:cpn { prg_replicate_first_9:n } #1 { \c_zero #1#1#1#1#1#1#1#1#1 }
      (End definition for \bool_if:n. This function is documented on page ??.)

```

`\prg_stepwise_function:nnnN`
`\prg_stepwise_function_incr:nnnN`
`\prg_stepwise_function_decr:nnnN`

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.

```

2196 \cs_new:Npn \prg_stepwise_function:nnnN #1#2#3#4
2197 {
2198   \int_compare:nNnTF {#2} = \c_zero
2199   { \msg_expandable_error:n { Zero~step~size~for~stepwise~function. } }
2200   {
2201     \int_compare:nNnTF {#2} > \c_zero
2202     { \exp_args:Nf \prg_stepwise_function_incr:nnnN }
2203     { \exp_args:Nf \prg_stepwise_function_decr:nnnN }
2204     { \int_eval:n {#1} } {#2} {#3} #4
2205   }
2206 }
2207 \cs_new:Npn \prg_stepwise_function_incr:nnnN #1#2#3#4
2208 {
2209   \int_compare:nNnF {#1} > {#3}
2210   {
2211     #4 {#1}
2212     \exp_args:Nf \prg_stepwise_function_incr:nnnN
2213     { \int_eval:n { #1 + #2 } } {#2} {#3} #4
2214   }
2215 }
2216 \cs_new:Npn \prg_stepwise_function_decr:nnnN #1#2#3#4
2217 {
2218   \int_compare:nNnF {#1} < {#3}
2219   {
2220     #4 {#1}
2221     \exp_args:Nf \prg_stepwise_function_decr:nnnN
2222     { \int_eval:n { #1 + #2 } } {#2} {#3} #4
2223   }
2224 }

```

(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.

```

2225 \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`
`\prg_stepwise_variable:nnnNn`
`\prg_stepwise_aux:NNnnnn`

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_function:nnnN`.

```

2226 \cs_new_protected:Npn \prg_stepwise_inline:nnnn
2227 {
2228   \exp_args:NNc \prg_stepwise_aux:NNnnnn
2229   \cs_gset_nopar:Npn
2230   { g_prg_stepwise_ \int_use:N \g_prg_stepwise_level_int :n }

```



```

2231 }
2232 \cs_new_protected:Npn \prg_stepwise_variable:nnnNn #1#2#3#4#5
2233 {
2234   \exp_args:Nnc \prg_stepwise_aux:NNnnnn
2235   \cs_gset_nopar:Npx
2236   { g_prg_stepwise_ \int_use:N \g_prg_stepwise_level_int :n }
2237   {#1}{#2}{#3}
2238   {
2239     \tl_set:Nn \exp_not:N #4 {##1}
2240     \exp_not:n {#5}
2241   }
2242 }
2243 \cs_new_protected:Npn \prg_stepwise_aux:NNnnnn #1#2#3#4#5#6
2244 {
2245   #1 #2 ##1 {#6}
2246   \int_gincr:N \g_prg_stepwise_level_int
2247   \prg_stepwise_function:nnnN {#3}{#4}{#5} #2
2248   \int_gdecr:N \g_prg_stepwise_level_int
2249 }

```

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

179.7 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.

```

2250 \prg_new_conditional:Npnn \mode_if_vertical: { p , T , F , TF }
2251 { \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.

```

2252 \prg_new_conditional:Npnn \mode_if_horizontal: { p , T , F , TF }
2253 { \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.

```

2254 \prg_new_conditional:Npnn \mode_if_inner: { p , T , F , TF }
2255 { \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.

```

2256 \prg_new_conditional:Npnn \mode_if_math: { p , T , F , TF }
2257 { \if_mode_math: \prg_return_true: \else: \prg_return_false: \fi: }

```

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

179.8 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.

```

2258 \cs_new_nopar:Npn \group_align_safe_begin:
2259 { \if_int_compare:w \if_false: { \fi: ' } = \c_zero \fi: }
2260 \cs_new_nopar:Npn \group_align_safe_end:
2261 { \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:nNtT \etex_currentgrouptype:D = \c_six
  {
    \int_compare:nNf \etex_lastnodetype:D = \c_zero
    {
      \int_compare:nNf \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.

```
2262 \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
2263 \group_begin:
2264   \tex_lccode:D '\& = '\g \scan_stop:
2265   \tex_catcode:D '\& = \c_twelve
2266   \tl_to_lowercase:n
2267   {
2268     \group_end:
2269     \cs_new_nopar:Npn \prg_variable_get_scope:N #1
2270     {
2271       \exp_last_unbraced:Nf \prg_variable_get_scope_aux:w
2272       { \cs_to_str:N #1 \exp_stop_f: \q_stop }
2273     }
2274     \cs_new_nopar:Npn \prg_variable_get_scope_aux:w #1#2 \q_stop
2275     { \token_if_eq_meaning:NNT & #1 { g } }
2276   }
2277 \group_begin:
2278   \tex_lccode:D '\& = '\_ \scan_stop:
2279   \tex_catcode:D '\& = \c_twelve
2280   \tl_to_lowercase:n
2281   {
2282     \group_end:
2283     \cs_new_nopar:Npn \prg_variable_get_type:N #1
2284     {
2285       \exp_after:wN \prg_variable_get_type_aux:w
2286       \token_to_str:N #1 & a \q_stop
2287     }
2288     \cs_new_nopar:Npn \prg_variable_get_type_aux:w #1 & #2#3 \q_stop
2289     {
2290       \token_if_eq_meaning:NNTF a #2
2291       {#1}
2292       { \prg_variable_get_type_aux:w #2#3 \q_stop }
2293     }
2294   }
```

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

179.9 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.

```
2295 \cs_new_protected_nopar:Npn \prg_define_quicksort:nnn #1#2#3 {
2296   \cs_set:cpx{#1_quicksort:n}##1{
2297     \exp_not:c{#1_quicksort_start_partition:w} ##1
2298     \exp_not:n{#2\q_nil#3\q_stop}
2299   }
2300   \cs_set:cpx{#1_quicksort_braced:n}##1{
2301     \exp_not:c{#1_quicksort_start_partition_braced:n} ##1
2302     \exp_not:N\q_nil\exp_not:N\q_stop
2303   }
2304   \cs_set:cpx {#1_quicksort_start_partition:w} #2 ##1 #3{
2305     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2306     \exp_not:c{#1_quicksort_do_partition_i:nnnw} {##1}{-}{-}
2307   }
2308   \cs_set:cpx {#1_quicksort_start_partition_braced:n} ##1 {
2309     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2310     \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn} {##1}{-}{-}
2311   }
```

Now for doing the partitions.

```
2312 \cs_set:cpx {#1_quicksort_do_partition_i:nnnw} ##1##2##3 #2 ##4 #3 {
2313   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2314   {
2315     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2316     \exp_not:c{#1_quicksort_partition_greater_ii:nnnn}
2317     \exp_not:c{#1_quicksort_partition_less_ii:nnnn}
2318   }
2319   {##1}{##2}{##3}{##4}
2320 }
2321 \cs_set:cpx {#1_quicksort_do_partition_i_braced:nnnn} ##1##2##3##4 {
2322   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2323   {
2324     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2325     \exp_not:c{#1_quicksort_partition_greater_ii_braced:nnnn}
2326     \exp_not:c{#1_quicksort_partition_less_ii_braced:nnnn}
2327   }
2328   {##1}{##2}{##3}{##4}
2329 }
```

```

2330 \cs_set:cpx {#1_quicksort_do_partition_ii:nnnw} ##1##2##3 #2 ##4 #3 {
2331   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2332   {
2333     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2334     \exp_not:c{#1_quicksort_partition_less_i:nnnn}
2335     \exp_not:c{#1_quicksort_partition_greater_i:nnnn}
2336   }
2337   {##1}{##2}{##3}{##4}
2338 }
2339 \cs_set:cpx {#1_quicksort_do_partition_ii_braced:nnnn} ##1##2##3##4 {
2340   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2341   {
2342     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2343     \exp_not:c{#1_quicksort_partition_less_i_braced:nnnn}
2344     \exp_not:c{#1_quicksort_partition_greater_i_braced:nnnn}
2345   }
2346   {##1}{##2}{##3}{##4}
2347 }

```

This part of the code handles the two branches in each sorting. Again we will also have to do it braced.

```

2348 \cs_set:cpx {#1_quicksort_partition_less_i:nnnn} ##1##2##3##4{
2349   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}
2350 \cs_set:cpx {#1_quicksort_partition_less_ii:nnnn} ##1##2##3##4{
2351   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}}
2352 \cs_set:cpx {#1_quicksort_partition_greater_i:nnnn} ##1##2##3##4{
2353   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}}
2354 \cs_set:cpx {#1_quicksort_partition_greater_ii:nnnn} ##1##2##3##4{
2355   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}}
2356 \cs_set:cpx {#1_quicksort_partition_less_i_braced:nnnn} ##1##2##3##4{
2357   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##3}{##4}}
2358 \cs_set:cpx {#1_quicksort_partition_less_ii_braced:nnnn} ##1##2##3##4{
2359   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##3}{##4}}
2360 \cs_set:cpx {#1_quicksort_partition_greater_i_braced:nnnn} ##1##2##3##4{
2361   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##3}{##4}}
2362 \cs_set:cpx {#1_quicksort_partition_greater_ii_braced:nnnn} ##1##2##3##4{
2363   \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.

```

2364 \cs_set:cpx {#1_do_quicksort_braced:nnnnw} ##1##2##3##4\q_stop {
2365   \exp_not:c{#1_quicksort_braced:n}{##2}
2366   \exp_not:c{#1_quicksort_function:n}{##1}
2367   \exp_not:c{#1_quicksort_braced:n}{##3}
2368 }
2369 }

```

(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.

```

2370 \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
2371 \cs_set:Npn \prg_quicksort_function:n {\ERROR}
2372 \cs_set:Npn \prg_quicksort_compare:nnTF {\ERROR}
      (End definition for \prg_quicksort_function:n. This function is documented on page 42.)

```

179.10 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
2373 \*deprecated
2374 \cs_new_protected:Npn \prg_new_map_functions:Nn #1#2 { \deprecated }
2375 \cs_new_protected:Npn \prg_set_map_functions:Nn #1#2 { \deprecated }
2376 \*deprecated
      (End definition for \prg_new_map_functions:Nn. This function is documented on page ??.)
2377 \*initex | package)

```

180 l3quark implementation

The following test files are used for this code: *m3quark001.lvt*.

```

2378 \*initex | package)
2379 \*package)
2380 \ProvidesExplPackage
2381   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2382 \package_check_loaded_expl:
2383 \*package)

\quark_new:N Allocate a new quark.
2384 \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
2385 \quark_new:N \q_nil
2386 \quark_new:N \q_mark
2387 \quark_new:N \q_no_value
2388 \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.

```
2389 \quark_new:N \q_recursion_tail
2390 \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.

```
2391 \cs_new:Npn \quark_if_recursion_tail_stop:N #1
2392 {
2393   \if_meaning:w #1 \q_recursion_tail
2394   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2395   \fi:
2396 }
2397 \cs_new:Npn \quark_if_recursion_tail_stop_do:Nn #1
2398 {
2399   \if_meaning:w #1 \q_recursion_tail
2400   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2401   \else:
2402     \exp_after:wN \use_none:n
2403   \fi:
2404 }
```

(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
2405 \cs_new:Npn \quark_if_recursion_tail_stop:n #1
2406 {
2407   \if_int_compare:w \pdfTeX_strcmp:D
2408     { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2409   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2410   \fi:
2411 }
2412 \cs_new:Npn \quark_if_recursion_tail_stop_do:nn #1
2413 {
2414   \if_int_compare:w \pdfTeX_strcmp:D
2415     { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2416   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2417   \else:
2418     \exp_after:wN \use_none:n
2419   \fi:
```

```

2420 }
2421 \cs_generate_variant:Nn \quark_if_recursion_tail_stop:n { o }
2422 \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

```

2423 \prg_new_conditional:Nnn \quark_if_nil:N { p, T, F, TF }
2424 {
2425   \if_meaning:w \q_nil #1
2426   \prg_return_true:
2427   \else:
2428   \prg_return_false:
2429   \fi:
2430 }
2431 \prg_new_conditional:Nnn \quark_if_no_value:N { p, T, F, TF }
2432 {
2433   \if_meaning:w \q_no_value #1
2434   \prg_return_true:
2435   \else:
2436   \prg_return_false:
2437   \fi:
2438 }
2439 \cs_generate_variant:Nn \quark_if_no_value_p:N { c }
2440 \cs_generate_variant:Nn \quark_if_no_value:NT { c }
2441 \cs_generate_variant:Nn \quark_if_no_value:NF { c }
2442 \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

```

2443 \prg_new_conditional:Nnn \quark_if_nil:n { p, T, F, TF }
2444 {
2445   \if_int_compare:w \pdfTeX_strcmp:D
2446   { \exp_not:N \q_nil } { \exp_not:n {#1} } = \c_zero
2447   \prg_return_true:
2448   \else:
2449   \prg_return_false:
2450   \fi:
2451 }
2452 \prg_new_conditional:Nnn \quark_if_no_value:n { p, T, F, TF }
2453 {
2454   \if_int_compare:w \pdfTeX_strcmp:D
2455   { \exp_not:N \q_no_value } { \exp_not:n {#1} } = \c_zero
2456   \prg_return_true:
2457   \else:

```

⁵It may still loop in special circumstances however!


```

2458     \prg_return_false:
2459     \fi:
2460   }
2461   \cs_generate_variant:Nn \quark_if_nil_p:n { V , o }
2462   \cs_generate_variant:Nn \quark_if_nil:nTF { V , o }
2463   \cs_generate_variant:Nn \quark_if_nil:nT { V , o }
2464   \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.

```

2465   \quark_new:N \q_tl_act_mark
2466   \quark_new:N \q_tl_act_stop

```

(End definition for \q_tl_act_mark and \q_tl_act_stop. These functions are documented on page 94.)

```

2467 \</initex | package>

```

181 l3token implementation

```

2468 <*initex | package>
2469 <*package>
2470 \ProvidesExplPackage
2471   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2472   \package_check_loaded_expl:
2473 </package>

```

181.1 Character tokens

\char_set_catcode:nn Category code changes.

```

\char_value_catcode:n
\char_show_value_catcode:n

```

```

2474 \cs_new_protected_nopar:Npn \char_set_catcode:nn #1#2
2475   { \tex_catcode:D #1 = \int_eval:w #2 \int_eval_end: }
2476 \cs_new_nopar:Npn \char_value_catcode:n #1
2477   { \tex_the:D \tex_catcode:D \int_eval:w #1 \int_eval_end: }
2478 \cs_new_nopar:Npn \char_show_value_catcode:n #1
2479   { \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

```

```

2489 { \char_set_catcode:nn { '#1 } \c_four }
2490 \cs_new_protected_nopar:Npn \char_set_catcode_end_line:N #1
2491 { \char_set_catcode:nn { '#1 } \c_five }
2492 \cs_new_protected_nopar:Npn \char_set_catcode_parameter:N #1
2493 { \char_set_catcode:nn { '#1 } \c_six }
2494 \cs_new_protected_nopar:Npn \char_set_catcode_math_superscript:N #1
2495 { \char_set_catcode:nn { '#1 } \c_seven }
2496 \cs_new_protected_nopar:Npn \char_set_catcode_math_subscript:N #1
2497 { \char_set_catcode:nn { '#1 } \c_eight }
2498 \cs_new_protected_nopar:Npn \char_set_catcode_ignore:N #1
2499 { \char_set_catcode:nn { '#1 } \c_nine }
2500 \cs_new_protected_nopar:Npn \char_set_catcode_space:N #1
2501 { \char_set_catcode:nn { '#1 } \c_ten }
2502 \cs_new_protected_nopar:Npn \char_set_catcode_letter:N #1
2503 { \char_set_catcode:nn { '#1 } \c_eleven }
2504 \cs_new_protected_nopar:Npn \char_set_catcode_other:N #1
2505 { \char_set_catcode:nn { '#1 } \c_twelve }
2506 \cs_new_protected_nopar:Npn \char_set_catcode_active:N #1
2507 { \char_set_catcode:nn { '#1 } \c_thirteen }
2508 \cs_new_protected_nopar:Npn \char_set_catcode_comment:N #1
2509 { \char_set_catcode:nn { '#1 } \c_fourteen }
2510 \cs_new_protected_nopar:Npn \char_set_catcode_invalid:N #1
2511 { \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 2512 \cs_new_protected_nopar:Npn \char_set_catcode_escape:n #1
  \char_set_catcode_group_end:n    2513 { \char_set_catcode:nn {#1} \c_zero }
  \char_set_catcode_math_toggle:n  2514 \cs_new_protected_nopar:Npn \char_set_catcode_group_begin:n #1
  \char_set_catcode_alignment:n     2515 { \char_set_catcode:nn {#1} \c_one }
\char_set_catcode_end_line:n        2516 \cs_new_protected_nopar:Npn \char_set_catcode_group_end:n #1
  \char_set_catcode_parameter:n     2517 { \char_set_catcode:nn {#1} \c_two }
  \char_set_catcode_math_superscript:n 2518 \cs_new_protected_nopar:Npn \char_set_catcode_math_toggle:n #1
  \char_set_catcode_math_subscript:n 2519 { \char_set_catcode:nn {#1} \c_three }
  \char_set_catcode_ignore:n         2520 \cs_new_protected_nopar:Npn \char_set_catcode_alignment:n #1
  \char_set_catcode_space:n          2521 { \char_set_catcode:nn {#1} \c_four }
  \char_set_catcode_letter:n         2522 \cs_new_protected_nopar:Npn \char_set_catcode_end_line:n #1
  \char_set_catcode_other:n          2523 { \char_set_catcode:nn {#1} \c_five }
  \char_set_catcode_active:n         2524 \cs_new_protected_nopar:Npn \char_set_catcode_parameter:n #1
  \char_set_catcode_comment:n        2525 { \char_set_catcode:nn {#1} \c_six }
  \char_set_catcode_invalid:n        2526 \cs_new_protected_nopar:Npn \char_set_catcode_math_superscript:n #1
                                     2527 { \char_set_catcode:nn {#1} \c_seven }
                                     2528 \cs_new_protected_nopar:Npn \char_set_catcode_math_subscript:n #1
                                     2529 { \char_set_catcode:nn {#1} \c_eight }
                                     2530 \cs_new_protected_nopar:Npn \char_set_catcode_ignore:n #1
                                     2531 { \char_set_catcode:nn {#1} \c_nine }
                                     2532 \cs_new_protected_nopar:Npn \char_set_catcode_space:n #1
                                     2533 { \char_set_catcode:nn {#1} \c_ten }
                                     2534 \cs_new_protected_nopar:Npn \char_set_catcode_letter:n #1

```

```

2535 { \char_set_catcode:nn {#1} \c_eleven }
2536 \cs_new_protected_nopar:Npn \char_set_catcode_other:n #1
2537 { \char_set_catcode:nn {#1} \c_twelve }
2538 \cs_new_protected_nopar:Npn \char_set_catcode_active:n #1
2539 { \char_set_catcode:nn {#1} \c_thirteen }
2540 \cs_new_protected_nopar:Npn \char_set_catcode_comment:n #1
2541 { \char_set_catcode:nn {#1} \c_fourteen }
2542 \cs_new_protected_nopar:Npn \char_set_catcode_invalid:n #1
2543 { \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 2544 \cs_new_protected_nopar:Npn \char_set_mathcode:nn #1#2
\char_show_value_mathcode:n 2545 { \tex_mathcode:D #1 = \int_eval:w #2 \int_eval_end: }
\char_set_lccode:nn 2546 \cs_new_nopar:Npn \char_value_mathcode:n #1
\char_value_lccode:n 2547 { \tex_the:D \tex_mathcode:D \int_eval:w #1\int_eval_end: }
\char_show_value_lccode:n 2548 \cs_new_nopar:Npn \char_show_value_mathcode:n #1
\char_set_uccode:nn 2549 { \tex_showthe:D \tex_mathcode:D \int_eval:w #1 \int_eval_end: }
\char_value_uccode:n 2550 \cs_new_protected_nopar:Npn \char_set_lccode:nn #1#2
\char_show_value_uccode:n 2551 { \tex_lccode:D #1 = \int_eval:w #2 \int_eval_end: }
\char_set_sfcode:nn 2552 \cs_new_nopar:Npn \char_value_lccode:n #1
\char_value_sfcode:n 2553 { \tex_the:D \tex_lccode:D \int_eval:w #1\int_eval_end: }
\char_show_value_sfcode:n 2554 \cs_new_nopar:Npn \char_show_value_lccode:n #1
2555 { \tex_showthe:D \tex_lccode:D \int_eval:w #1 \int_eval_end: }
2556 \cs_new_protected_nopar:Npn \char_set_uccode:nn #1#2
2557 { \tex_uccode:D #1 = \int_eval:w #2 \int_eval_end: }
2558 \cs_new_nopar:Npn \char_value_uccode:n #1
2559 { \tex_the:D \tex_uccode:D \int_eval:w #1\int_eval_end: }
2560 \cs_new_nopar:Npn \char_show_value_uccode:n #1
2561 { \tex_showthe:D \tex_uccode:D \int_eval:w #1 \int_eval_end: }
2562 \cs_new_protected_nopar:Npn \char_set_sfcode:nn #1#2
2563 { \tex_sfcode:D #1 = \int_eval:w #2 \int_eval_end: }
2564 \cs_new_nopar:Npn \char_value_sfcode:n #1
2565 { \tex_the:D \tex_sfcode:D \int_eval:w #1\int_eval_end: }
2566 \cs_new_nopar:Npn \char_show_value_sfcode:n #1
2567 { \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.)

181.2 Generic tokens

`\token_new:Nn` Creates a new token.

```

2568 \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 2569 \cs_new_eq:NN \c_group_begin_token {
\c_alignment_token 2570 \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

```

```

2571 \group_begin:
2572   \char_set_catcode_math_toggle:N \*
2573   \token_new:Nn \c_math_toggle_token { * }
2574   \char_set_catcode_alignment:N \*
2575   \token_new:Nn \c_alignment_token { * }
2576   \token_new:Nn \c_parameter_token { # }
2577   \token_new:Nn \c_math_superscript_token { ^ }
2578   \char_set_catcode_math_subscript:N \*
2579   \token_new:Nn \c_math_subscript_token { * }
2580   \token_new:Nn \c_space_token { ~ }
2581   \token_new:Nn \c_catcode_letter_token { a }
2582   \token_new:Nn \c_catcode_other_token { 1 }
2583 \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!

```

2584 \group_begin:
2585   \char_set_catcode_active:N \*
2586   \cs_new_nopar:Npn \c_catcode_active_tl { \exp_not:N * }
2587 \group_end:

```

(End definition for `\c_catcode_active_tl`. This function is documented on page 50.)

181.3 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.

```

2588 \prg_new_conditional:Npnn \token_if_group_begin:N #1 { p , T , F , TF }
2589 {
2590   \if_catcode:w \exp_not:N #1 \c_group_begin_token
2591   \prg_return_true: \else: \prg_return_false: \fi:
2592 }

```

(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.

```

2593 \prg_new_conditional:Npnn \token_if_group_end:N #1 { p , T , F , TF }
2594 {
2595   \if_catcode:w \exp_not:N #1 \c_group_end_token
2596   \prg_return_true: \else: \prg_return_false: \fi:
2597 }

```

(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.

```

2598 \prg_new_conditional:Npnn \token_if_math_toggle:N #1 { p , T , F , TF }
2599 {
2600   \if_catcode:w \exp_not:N #1 \c_math_toggle_token
2601   \prg_return_true: \else: \prg_return_false: \fi:
2602 }

```

(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.

```
2603 \prg_new_conditional:Npnn \token_if_alignment:N #1 { p , T , F , TF }
2604 {
2605     \if_catcode:w \exp_not:N #1 \c_alignment_token
2606     \prg_return_true: \else: \prg_return_false: \fi:
2607 }
```

(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.

```
2608 \group_begin:
2609 \cs_set_eq:NN \c_parameter_token \scan_stop:
2610 \prg_new_conditional:Npnn \token_if_parameter:N #1 { p , T , F , TF }
2611 {
2612     \if_catcode:w \exp_not:N #1 \c_parameter_token
2613     \prg_return_true: \else: \prg_return_false: \fi:
2614 }
2615 \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.

```
2616 \prg_new_conditional:Npnn \token_if_math_superscript:N #1 { p , T , F , TF }
2617 {
2618     \if_catcode:w \exp_not:N #1 \c_math_superscript_token
2619     \prg_return_true: \else: \prg_return_false: \fi:
2620 }
```

(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.

```
2621 \prg_new_conditional:Npnn \token_if_math_subscript:N #1 { p , T , F , TF }
2622 {
2623     \if_catcode:w \exp_not:N #1 \c_math_subscript_token
2624     \prg_return_true: \else: \prg_return_false: \fi:
2625 }
```

(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.

```
2626 \prg_new_conditional:Npnn \token_if_space:N #1 { p , T , F , TF }
2627 {
2628     \if_catcode:w \exp_not:N #1 \c_space_token
2629     \prg_return_true: \else: \prg_return_false: \fi:
2630 }
```

(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.

```
2631 \prg_new_conditional:Npnn \token_if_letter:N #1 { p , T , F , TF }
2632 {
2633   \if_catcode:w \exp_not:N #1 \c_catcode_letter_token
2634   \prg_return_true: \else: \prg_return_false: \fi:
2635 }
```

(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.

```
2636 \prg_new_conditional:Npnn \token_if_other:N #1 { p , T , F , TF }
2637 {
2638   \if_catcode:w \exp_not:N #1 \c_catcode_other_token
2639   \prg_return_true: \else: \prg_return_false: \fi:
2640 }
```

(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.

```
2641 \prg_new_conditional:Npnn \token_if_active:N #1 { p , T , F , TF }
2642 {
2643   \if_catcode:w \exp_not:N #1 \c_catcode_active_tl
2644   \prg_return_true: \else: \prg_return_false: \fi:
2645 }
```

(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.

```
2646 \prg_new_conditional:Npnn \token_if_eq_meaning:NN #1#2 { p , T , F , TF }
2647 {
2648   \if_meaning:w #1 #2
2649   \prg_return_true: \else: \prg_return_false: \fi:
2650 }
```

(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.

```
2651 \prg_new_conditional:Npnn \token_if_eq_catcode:NN #1#2 { p , T , F , TF }
2652 {
2653   \if_catcode:w \exp_not:N #1 \exp_not:N #2
2654   \prg_return_true: \else: \prg_return_false: \fi:
2655 }
```

(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.

```

2656 \prg_new_conditional:Npnn \token_if_eq_charcode:NN #1#2 { p , T , F , TF }
2657 {
2658   \if_charcode:w \exp_not:N #1 \exp_not:N #2
2659   \prg_return_true: \else: \prg_return_false: \fi:
2660 }

```

(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.

```

2661 \group_begin:
2662 \char_set_catcode_other:N \M
2663 \char_set_catcode_other:N \A
2664 \char_set_lccode:nn { '\; } { '\: }
2665 \char_set_lccode:nn { '\T } { '\T }
2666 \char_set_lccode:nn { '\F } { '\F }
2667 \tl_to_lowercase:n
2668 {
2669   \group_end:
2670   \prg_new_conditional:Npnn \token_if_macro:N #1 { p , T , F , TF }
2671   {
2672     \exp_after:wN \token_if_macro_p_aux:w
2673     \token_to_meaning:N #1 MA; \q_stop
2674   }
2675   \cs_new_nopar:Npn \token_if_macro_p_aux:w #1 MA #2 ; #3 \q_stop
2676   {
2677     \if_int_compare:w \pdfTeX_strcmp:D { #2 } { cro } = \c_zero
2678     \prg_return_true:
2679   \else:
2680     \prg_return_false:
2681   \fi:
2682   }
2683 }

```

(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.

```
2684 \prg_new_conditional:Npnn \token_if_cs:N #1 { p , T , F , TF }
2685 {
2686   \if_catcode:w \exp_not:N #1 \scan_stop:
2687   \prg_return_true: \else: \prg_return_false: \fi:
2688 }
```

(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.

```
2689 \prg_new_conditional:Npnn \token_if_expandable:N #1 { p , T , F , TF }
2690 {
2691   \cs_if_exist:NTF #1
2692   {
2693     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
2694     \prg_return_false: \else: \prg_return_true: \fi:
2695   }
2696   { \prg_return_false: }
2697 }
```

(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_long_macro:N
\token_if_dim_register:N
\token_if_skip_register:N
\token_if_int_register:N
\token_if_toks_register:N
\token_if_chardef_p_aux:w
\token_if_mathchardef_p_aux:w
\token_if_int_register_p_aux:w
\token_if_skip_register_p_aux:w
\token_if_dim_register_p_aux:w
\token_if_toks_register_p_aux:w
\token_if_protected_macro_p_aux:w
\token_if_long_macro_p_aux:w
\token_if_protected_long_macro_p_aux:w

2698 \group_begin:
2699 \char_set_lccode:nn { '\T } { '\T }
2700 \char_set_lccode:nn { '\F } { '\F }
2701 \char_set_lccode:nn { '\X } { '\n }
2702 \char_set_lccode:nn { '\Y } { '\t }
2703 \char_set_lccode:nn { '\Z } { '\d }
2704 \char_set_lccode:nn { '\? } { '\\ }
2705 \tl_map_inline:nn { \X \Y \Z \M \C \H \A \R \O \U \S \K \I \P \L \G \P \E }
2706 { \char_set_catcode:nn { '#1 } \c_twelve }
```

We convert the token list to lower case and restore the catcode and lowercase code changes.

```
2707 \tl_to_lowercase:n
2708 {
2709   \group_end:
2710   \prg_new_conditional:Npnn \token_if_chardef:N #1 { p , T , F , TF }
2711   {
2712     \exp_after:wN \token_if_chardef_aux:w
2713     \token_to_meaning:N #1 ?CHAR" \q_stop
```

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⟩`.


```

2714     }
2715     \cs_new_nopar:Npn \token_if_chardef_aux:w #1 ?CHAR" #2 \q_stop
2716     { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
2717     \prg_new_conditional:Npnn \token_if_mathchardef:N #1 { p , T , F , TF }
2718     {
2719         \exp_after:wN \token_if_mathchardef_aux:w
2720         \token_to_meaning:N #1 ?MAYHCHAR" \q_stop
2721     }
2722     \cs_new_nopar:Npn \token_if_mathchardef_aux:w #1 ?MAYHCHAR" #2 \q_stop
2723     { \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.

```

2724     \prg_new_conditional:Npnn \token_if_int_register:N #1 { p , T , F , TF }
2725     {
2726         \if_meaning:w \tex_countdef:D #1
2727         \prg_return_false:
2728     \else:
2729         \exp_after:wN \token_if_int_register_aux:w
2730         \token_to_meaning:N #1 ?COUXY \q_stop
2731     \fi:
2732 }
2733 \cs_new_nopar:Npn \token_if_int_register_aux:w #1 ?COUXY #2 \q_stop
2734 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Skip registers are done the same way as the integer registers.

```

2735     \prg_new_conditional:Npnn \token_if_skip_register:N #1 { p , T , F , TF }
2736     {
2737         \if_meaning:w \tex_skipdef:D #1
2738         \prg_return_false:
2739     \else:
2740         \exp_after:wN \token_if_skip_register_aux:w
2741         \token_to_meaning:N #1 ?SKIP \q_stop
2742     \fi:
2743 }
2744 \cs_new_nopar:Npn \token_if_skip_register_aux:w #1 ?SKIP #2 \q_stop
2745 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Dim registers. No news here

```

2746     \prg_new_conditional:Npnn \token_if_dim_register:N #1 { p , T , F , TF }
2747     {
2748         \if_meaning:w \tex_dimendef:D #1
2749         \c_false_bool
2750     \else:
2751         \exp_after:wN \token_if_dim_register_aux:w
2752         \token_to_meaning:N #1 ?ZIMEX \q_stop
2753     \fi:
2754 }
2755 \cs_new_nopar:Npn \token_if_dim_register_aux:w #1 ?ZIMEX #2 \q_stop
2756 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Toks registers.

```

2757 \prg_new_conditional:Npnn \token_if_toks_register:N #1 { p , T , F , TF }
2758 {
2759   \if_meaning:w \tex_toksdef:D #1
2760   \prg_return_false:
2761   \else:
2762     \exp_after:wN \token_if_toks_register_aux:w
2763     \token_to_meaning:N #1 ?YOKS \q_stop
2764   \fi:
2765 }
2766 \cs_new_nopar:Npn \token_if_toks_register_aux:w #1 ?YOKS #2 \q_stop
2767 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Protected macros.

```

2768 \prg_new_conditional:Npnn \token_if_protected_macro:N #1
2769 { p , T , F , TF }
2770 {
2771   \exp_after:wN \token_if_protected_macro_aux:w
2772   \token_to_meaning:N #1 ?PROYECYEZ~MACRO \q_stop
2773 }
2774 \cs_new_nopar:Npn \token_if_protected_macro_aux:w
2775 #1 ?PROYECYEZ~MACRO #2 \q_stop
2776 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Long macros.

```

2777 \prg_new_conditional:Npnn \token_if_long_macro:N #1 { p , T , F , TF }
2778 {
2779   \exp_after:wN \token_if_long_macro_aux:w
2780   \token_to_meaning:N #1 ?LOXG~MACRO \q_stop
2781 }
2782 \cs_new_nopar:Npn \token_if_long_macro_aux:w #1 ?LOXG~MACRO #2 \q_stop
2783 { \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.

```

2784 \prg_new_conditional:Npnn \token_if_protected_long_macro:N #1
2785 { p , T , F , TF }
2786 {
2787   \exp_after:wN \token_if_protected_long_macro_aux:w
2788   \token_to_meaning:N #1 ?PROYECYEZ?LOXG~MACRO \q_stop
2789 }
2790 \cs_new_nopar:Npn \token_if_protected_long_macro_aux:w
2791 #1 ?PROYECYEZ?LOXG~MACRO #2 \q_stop
2792 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Finally the \tl_to_lowercase:n ends!

```

2793 }

```

(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.

```

2794 \tex_chardef:D \c_token_A_int = 'A ~ %
2795 \group_begin:
2796 \char_set_catcode_other:N \;
2797 \char_set_lccode:nn { '\; } { '\: }
2798 \char_set_lccode:nn { '\T } { '\T }
2799 \char_set_lccode:nn { '\F } { '\F }
2800 \tl_to_lowercase:n {
2801   \group_end:
2802   \prg_new_conditional:Npnn \token_if_primitive:N #1 { p , T , F , TF }
2803   {
2804     \token_if_macro:NTF #1
2805     \prg_return_false:
2806     {
2807       \exp_after:wN \token_if_primitive_aux:NNw
2808       \token_to_meaning:N #1 ; ; ; \q_stop #1
2809     }
2810   }
2811 \cs_new_nopar:Npn \token_if_primitive_aux:NNw #1#2 #3 ; #4 \q_stop
2812 {
2813   \tl_if_empty:oTF { \token_if_primitive_aux_space:w #3 ~ }
2814   { \token_if_primitive_aux_loop:N #3 ; \q_stop }
2815   { \token_if_primitive_aux_nullfont:N }
2816 }
2817 }

```

```

2818 \cs_new_nopar:Npn \token_if_primitive_aux_space:w #1 ~ { }
2819 \cs_new:Npn \token_if_primitive_aux_nullfont:N #1
2820 {
2821   \if_meaning:w \tex_nullfont:D #1
2822     \prg_return_true:
2823   \else:
2824     \prg_return_false:
2825   \fi:
2826 }
2827 \cs_new_nopar:Npn \token_if_primitive_aux_loop:N #1
2828 {
2829   \if_num:w '#1 < \c_token_A_int %
2830     \exp_after:wN \token_if_primitive_auxii:Nw
2831     \exp_after:wN #1
2832   \else:
2833     \exp_after:wN \token_if_primitive_aux_loop:N
2834   \fi:
2835 }
2836 \cs_new_nopar:Npn \token_if_primitive_auxii:Nw #1 #2 \q_stop
2837 {
2838   \if:w : #1
2839     \exp_after:wN \token_if_primitive_aux_undefined:N
2840   \else:
2841     \prg_return_false:
2842     \exp_after:wN \use_none:n
2843   \fi:
2844 }
2845 \cs_new:Npn \token_if_primitive_aux_undefined:N #1
2846 {
2847   \if_cs_exist:N #1
2848     \prg_return_true:
2849   \else:
2850     \prg_return_false:
2851   \fi:
2852 }

```

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

181.4 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` 2853 `\cs_new_eq:NN \l_peek_token ?`
2854 `\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`.
2855 `\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`.
2856 `\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` 2857 `\cs_new_nopar:Npn \peek_true:w { }`
`\peek_false:w` 2858 `\cs_new_nopar:Npn \peek_true_aux:w { }`
`\peek_tmp:w` 2859 `\cs_new_nopar:Npn \peek_false:w { }`
2860 `\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` 2861 `\cs_new_protected_nopar:Npn \peek_after:Nw`
2862 `{ \tex_futurelet:D \l_peek_token }`
2863 `\cs_new_protected_nopar:Npn \peek_gafter:Nw`
2864 `{ \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.
2865 `\cs_new_protected:Npn \peek_true_remove:w`
2866 `{`
2867 `\group_align_safe_end:`
2868 `\tex_afterassignment:D \peek_true_aux:w`
2869 `\cs_set_eq:NN \peek_tmp:w`
2870 `}`
(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.
2871 `\cs_new_protected:Npn \peek_token_generic:NTTF #1#2#3#4`
2872 `{`
2873 `\cs_set_eq:NN \l_peek_search_token #2`
2874 `\tl_set:Nn \l_peek_search_tl {#2}`
2875 `\cs_set_nopar:Npx \peek_true:w`
2876 `{`
2877 `\exp_not:N \group_align_safe_end:`
2878 `\exp_not:n {#3}`
2879 `}`

```

2880     \cs_set_nopar:Npx \peek_false:w
2881     {
2882         \exp_not:N \group_align_safe_end:
2883         \exp_not:n {#4}
2884     }
2885     \group_align_safe_begin:
2886     \peek_after:Nw #1
2887 }
2888 \cs_new_protected:Npn \peek_token_generic:NNT #1#2#3
2889 { \peek_token_generic:NNTF #1 #2 {#3} { } }
2890 \cs_new_protected:Npn \peek_token_generic:NNF #1#2#3
2891 { \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.

```

2892 \cs_new_protected:Npn \peek_token_remove_generic:NNTF #1#2#3#4
2893 {
2894     \cs_set_eq:NN \l_peek_search_token #2
2895     \tl_set:Nn \l_peek_search_tl {#2}
2896     \cs_set_eq:NN \peek_true:w \peek_true_remove:w
2897     \cs_set_nopar:Npx \peek_true_aux:w { \exp_not:n {#3} }
2898     \cs_set_nopar:Npx \peek_false:w
2899     {
2900         \exp_not:N \group_align_safe_end:
2901         \exp_not:n {#4}
2902     }
2903     \group_align_safe_begin:
2904     \peek_after:Nw #1
2905 }
2906 \cs_new_protected:Npn \peek_token_remove_generic:NNT #1#2#3
2907 { \peek_token_remove_generic:NNTF #1 #2 {#3} { } }
2908 \cs_new_protected:Npn \peek_token_remove_generic:NNF #1#2#3
2909 { \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:
2910 \cs_new_nopar:Npn \peek_execute_branches_catcode:
2911 {
2912     \if_catcode:w
2913         \exp_not:N \l_peek_token \exp_not:N \l_peek_search_token
2914         \exp_after:wN \peek_true:w
2915     \else:
2916         \exp_after:wN \peek_false:w
2917     \fi:
2918 }
2919 \cs_new_nopar:Npn \peek_execute_branches_meaning:
2920 {
2921     \if_meaning:w \l_peek_token \l_peek_search_token
2922         \exp_after:wN \peek_true:w

```

```

2923 \else:
2924 \exp_after:wN \peek_false:w
2925 \fi:
2926 }

```

(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.

```

2927 \cs_new_nopar:Npn \peek_execute_branches_charcode:
2928 {
2929 \bool_if:nTF
2930 {
2931 \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token
2932 || \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token
2933 || \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
2934 }
2935 { \peek_false:w }
2936 {
2937 \exp_after:wN \peek_execute_branches_charcode_aux:NN
2938 \l_peek_search_tl
2939 }
2940 }
2941 \cs_new:Npn \peek_execute_branches_charcode_aux:NN #1#2
2942 {
2943 \if:w \exp_not:N #1 \exp_not:N #2
2944 \exp_after:wN \peek_true:w
2945 \else:
2946 \exp_after:wN \peek_false:w
2947 \fi:
2948 #2
2949 }

```

(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.

```

2950 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches:
2951 {
2952 \token_if_eq_meaning:NNTF \l_peek_token \c_space_token
2953 {
2954 \tex_afterassignment:D \peek_ignore_spaces_execute_branches_aux:
2955 \cs_set_eq:NN \peek_tmp:w
2956 }
2957 { \peek_execute_branches: }
2958 }
2959 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches_aux:
2960 { \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.

```

2961 \group_begin:
2962   \cs_set_nopar:Npn \peek_def:nnnn #1#2#3#4
2963   {
2964     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { TF }
2965     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { T }
2966     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { F }
2967   }
2968   \cs_set_nopar:Npn \peek_def_aux:nnnnn #1#2#3#4#5
2969   {
2970     \cs_gset_nopar:cpx { #1 #5 }
2971     {
2972       \tl_if_empty:nF {#2}
2973       { \exp_not:n { \cs_set_eq:NN \peek_execute_branches: #2 } }
2974       \exp_not:c { #3 #5 }
2975       \exp_not:n {#4}
2976     }
2977   }

```

(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 2978 \peek_def:nnnn { peek_catcode:N }
\peek_catcode_remove:N         2979 { }
\peek_catcode_remove_ignore_spaces:N 2980 { peek_token_generic:NN }
                                2981 { \peek_execute_branches_catcode: }
                                2982 \peek_def:nnnn { peek_catcode_ignore_spaces:N }
                                2983 { \peek_execute_branches_catcode: }
                                2984 { peek_token_generic:NN }
                                2985 { \peek_ignore_spaces_execute_branches: }
                                2986 \peek_def:nnnn { peek_catcode_remove:N }
                                2987 { }
                                2988 { peek_token_remove_generic:NN }
                                2989 { \peek_execute_branches_catcode: }
                                2990 \peek_def:nnnn { peek_catcode_remove_ignore_spaces:N }
                                2991 { \peek_execute_branches_catcode: }
                                2992 { peek_token_remove_generic:NN }
                                2993 { \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 2994 \peek_def:nnnn { peek_charcode:N }
\peek_charcode_remove:N        2995 { }
\peek_charcode_remove_ignore_spaces:N 2996 { peek_token_generic:NN }
                                2997 { \peek_execute_branches_charcode: }

```



```

2998 \peek_def:nnnn { peek_charcode_ignore_spaces:N }
2999   { \peek_execute_branches_charcode: }
3000   { peek_token_generic:NN }
3001   { \peek_ignore_spaces_execute_branches: }
3002 \peek_def:nnnn { peek_charcode_remove:N }
3003   { }
3004   { peek_token_remove_generic:NN }
3005   { \peek_execute_branches_charcode: }
3006 \peek_def:nnnn { peek_charcode_remove_ignore_spaces:N }
3007   { \peek_execute_branches_charcode: }
3008   { peek_token_remove_generic:NN }
3009   { \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
\peek_meaning_remove:N
\peek_meaning_remove_ignore_spaces:N
3010 \peek_def:nnnn { peek_meaning:N }
3011   { }
3012   { peek_token_generic:NN }
3013   { \peek_execute_branches_meaning: }
3014 \peek_def:nnnn { peek_meaning_ignore_spaces:N }
3015   { \peek_execute_branches_meaning: }
3016   { peek_token_generic:NN }
3017   { \peek_ignore_spaces_execute_branches: }
3018 \peek_def:nnnn { peek_meaning_remove:N }
3019   { }
3020   { peek_token_remove_generic:NN }
3021   { \peek_execute_branches_meaning: }
3022 \peek_def:nnnn { peek_meaning_remove_ignore_spaces:N }
3023   { \peek_execute_branches_meaning: }
3024   { peek_token_remove_generic:NN }
3025   { \peek_ignore_spaces_execute_branches: }
3026 \group_end:

```

(End definition for \peek_meaning:N and others. These functions are documented on page 56.)

181.5 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.

```

3027 \exp_args:Nno \use:nn
3028   { \cs_new_nopar:Npn \token_get_prefix_arg_replacement_aux:wN #1 }
3029   { \tl_to_str:n { macro : } #2 -> #3 \q_stop #4 }
3030   { #4 {#1} {#2} {#3} }
3031 \cs_new:Npn \token_get_prefix_spec:N #1
3032   {

```

```

3033 \token_if_macro:NTF #1
3034 {
3035   \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3036   \token_to_meaning:N #1 \q_stop \use_i:nnn
3037 }
3038 { \scan_stop: }
3039 }
3040 \cs_new:Npn \token_get_arg_spec:N #1
3041 {
3042   \token_if_macro:NTF #1
3043   {
3044     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3045     \token_to_meaning:N #1 \q_stop \use_ii:nnn
3046   }
3047   { \scan_stop: }
3048 }
3049 \cs_new:Npn \token_get_replacement_spec:N #1
3050 {
3051   \token_if_macro:NTF #1
3052   {
3053     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3054     \token_to_meaning:N #1 \q_stop \use_iii:nnn
3055   }
3056   { \scan_stop: }
3057 }

```

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

181.6 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
3058 \group_begin:
3059 \char_set_catcode_active:N ^^@
3060 \cs_set:Npn \char_tmp:NN #1#2
3061 {
3062   \cs_new:Npn #1 ##1
3063   {
3064     \char_set_catcode_active:n { '##1 }
3065     \group_begin:
3066     \char_set_lccode:nn { '\^^@ } { '##1 }
3067     \tl_to_lowercase:n { \group_end: #2 ^^@ }
3068   }
3069 }
3070 \char_tmp:NN \char_active_set:Npn \cs_set:Npn
3071 \char_tmp:NN \char_active_set:Npx \cs_set:Npx
3072 \char_tmp:NN \char_active_gset:Npn \cs_gset:Npn
3073 \char_tmp:NN \char_active_gset:Npx \cs_gset:Npx
3074 \char_tmp:NN \char_active_set_eq:NN \cs_set_eq:NN
3075 \char_tmp:NN \char_active_gset_eq:NN \cs_gset_eq:NN
3076 \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.

```

3077 \cs_new_protected_nopar:Npn \peek_execute_branches_N_type:
3078 {
3079   \bool_if:nTF
3080     {
3081       \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
3082       \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token ||
3083       \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3084     }
3085     { \peek_false:w }
3086     { \peek_true:w }
3087   }
3088 \cs_new_protected_nopar:Npn \peek_N_type:TF
3089 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }
3090 \cs_new_protected_nopar:Npn \peek_N_type:T
3091 { \peek_token_generic:NNT \peek_execute_branches_N_type: \scan_stop: }
3092 \cs_new_protected_nopar:Npn \peek_N_type:F
3093 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }

```

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

181.7 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\char_set_catcode:w` Primitives renamed.

```

\char_set_mathcode:w 3094 {*deprecated}
\char_set_lccode:w    3095 \cs_new_eq:NN \char_set_catcode:w \tex_catcode:D
\char_set_uccode:w    3096 \cs_new_eq:NN \char_set_mathcode:w \tex_mathcode:D
\char_set_sfcode:w    3097 \cs_new_eq:NN \char_set_lccode:w \tex_lccode:D
                     3098 \cs_new_eq:NN \char_set_uccode:w \tex_uccode:D
                     3099 \cs_new_eq:NN \char_set_sfcode:w \tex_sfcode:D
                     3100 {/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 3101 {*deprecated}
\char_value_mathcode:w     3102 \cs_new_nopar:Npn \char_value_catcode:w { \tex_the:D \char_set_catcode:w }
\char_show_value_mathcode:w 3103 \cs_new_nopar:Npn \char_show_value_catcode:w
\char_value_lccode:w       3104 { \tex_showthe:D \char_set_catcode:w }
\char_show_value_lccode:w   3105 \cs_new_nopar:Npn \char_value_mathcode:w { \tex_the:D \char_set_mathcode:w }
\char_value_uccode:w       3106 \cs_new_nopar:Npn \char_show_value_mathcode:w
\char_show_value_uccode:w   3107 { \tex_showthe:D \char_set_mathcode:w }
\char_value_sfcode:w       3108 \cs_new_nopar:Npn \char_value_lccode:w { \tex_the:D \char_set_lccode:w }
\char_show_value_sfcode:w

```

```

3109 \cs_new_nopar:Npn \char_show_value_lccode:w
3110 { \tex_showthe:D \char_set_lccode:w }
3111 \cs_new_nopar:Npn \char_value_uccode:w { \tex_the:D \char_set_uccode:w }
3112 \cs_new_nopar:Npn \char_show_value_uccode:w
3113 { \tex_showthe:D \char_set_uccode:w }
3114 \cs_new_nopar:Npn \char_value_sfcode:w { \tex_the:D \char_set_sfcode:w }
3115 \cs_new_nopar:Npn \char_show_value_sfcode:w
3116 { \tex_showthe:D \char_set_sfcode:w }
3117 </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
3118 < *deprecated>
3119 \cs_new_eq:NN \peek_after:NN \peek_after:Nw
3120 \cs_new_eq:NN \peek_gafter:NN \peek_gafter:Nw
3121 </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
3122 < *deprecated>
3123 \cs_new_eq:NN \c_alignment_tab_token \c_alignment_token
3124 \cs_new_eq:NN \c_math_shift_token \c_math_toggle_token
3125 \cs_new_eq:NN \c_letter_token \c_catcode_letter_token
3126 \cs_new_eq:NN \c_other_char_token \c_catcode_other_token
3127 </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!

```

3128 < *deprecated>
3129 \cs_new_eq:NN \c_active_char_token \c_catcode_active_tl
3130 </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
3131 < *deprecated>
3132 \cs_new_eq:NN \char_make_escape:N \char_set_catcode_escape:N
3133 \cs_new_eq:NN \char_make_begin_group:N \char_set_catcode_group_begin:N
3134 \cs_new_eq:NN \char_make_end_group:N \char_set_catcode_group_end:N
3135 \cs_new_eq:NN \char_make_math_shift:N \char_set_catcode_math_toggle:N
3136 \cs_new_eq:NN \char_make_alignment_tab:N \char_set_catcode_alignment:N
3137 \cs_new_eq:NN \char_make_end_line:N \char_set_catcode_end_line:N
3138 \cs_new_eq:NN \char_make_parameter:N \char_set_catcode_parameter:N
3139 \cs_new_eq:NN \char_make_math_superscript:N
3140 \char_set_catcode_math_superscript:N
3141 \cs_new_eq:NN \char_make_math_subscript:N
3142 \char_set_catcode_math_subscript:N
3143 \cs_new_eq:NN \char_make_ignore:N \char_set_catcode_ignore:N
3144 \cs_new_eq:NN \char_make_space:N \char_set_catcode_space:N

```

```

3145 \cs_new_eq:NN \char_make_letter:N \char_set_catcode_letter:N
3146 \cs_new_eq:NN \char_make_other:N \char_set_catcode_other:N
3147 \cs_new_eq:NN \char_make_active:N \char_set_catcode_active:N
3148 \cs_new_eq:NN \char_make_comment:N \char_set_catcode_comment:N
3149 \cs_new_eq:NN \char_make_invalid:N \char_set_catcode_invalid:N
3150 \cs_new_eq:NN \char_make_escape:n \char_set_catcode_escape:n
3151 \cs_new_eq:NN \char_make_begin_group:n \char_set_catcode_group_begin:n
3152 \cs_new_eq:NN \char_make_end_group:n \char_set_catcode_group_end:n
3153 \cs_new_eq:NN \char_make_math_shift:n \char_set_catcode_math_toggle:n
3154 \cs_new_eq:NN \char_make_alignment_tab:n \char_set_catcode_alignment:n
3155 \cs_new_eq:NN \char_make_end_line:n \char_set_catcode_end_line:n
3156 \cs_new_eq:NN \char_make_parameter:n \char_set_catcode_parameter:n
3157 \cs_new_eq:NN \char_make_math_superscript:n
3158 \char_set_catcode_math_superscript:n
3159 \cs_new_eq:NN \char_make_math_subscript:n
3160 \char_set_catcode_math_subscript:n
3161 \cs_new_eq:NN \char_make_ignore:n \char_set_catcode_ignore:n
3162 \cs_new_eq:NN \char_make_space:n \char_set_catcode_space:n
3163 \cs_new_eq:NN \char_make_letter:n \char_set_catcode_letter:n
3164 \cs_new_eq:NN \char_make_other:n \char_set_catcode_other:n
3165 \cs_new_eq:NN \char_make_active:n \char_set_catcode_active:n
3166 \cs_new_eq:NN \char_make_comment:n \char_set_catcode_comment:n
3167 \cs_new_eq:NN \char_make_invalid:n \char_set_catcode_invalid:n
3168 </deprecated>

```

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

```

\token_if_alignment_tab:N
\token_if_math_shift:N
\token_if_other_char:N
\token_if_active_char:N

```

```

3169 < *deprecated >
3170 \cs_new_eq:NN \token_if_alignment_tab_p:N \token_if_alignment_p:N
3171 \cs_new_eq:NN \token_if_alignment_tab_NT \token_if_alignment_NT
3172 \cs_new_eq:NN \token_if_alignment_tab_NF \token_if_alignment_NF
3173 \cs_new_eq:NN \token_if_alignment_tab_NTF \token_if_alignment_NTF
3174 \cs_new_eq:NN \token_if_math_shift_p:N \token_if_math_toggle_p:N
3175 \cs_new_eq:NN \token_if_math_shift_NT \token_if_math_toggle_NT
3176 \cs_new_eq:NN \token_if_math_shift_NF \token_if_math_toggle_NF
3177 \cs_new_eq:NN \token_if_math_shift_NTF \token_if_math_toggle_NTF
3178 \cs_new_eq:NN \token_if_other_char_p:N \token_if_other_p:N
3179 \cs_new_eq:NN \token_if_other_char_NT \token_if_other_NT
3180 \cs_new_eq:NN \token_if_other_char_NF \token_if_other_NF
3181 \cs_new_eq:NN \token_if_other_char_NTF \token_if_other_NTF
3182 \cs_new_eq:NN \token_if_active_char_p:N \token_if_active_p:N
3183 \cs_new_eq:NN \token_if_active_char_NT \token_if_active_NT
3184 \cs_new_eq:NN \token_if_active_char_NF \token_if_active_NF
3185 \cs_new_eq:NN \token_if_active_char_NTF \token_if_active_NTF
3186 </deprecated>

```

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

```

3187 </initex | package>

```

182 l3int implementation

```

3188 <*initex | package>
      The following test files are used for this code: m3int001,m3int002,m3int03.
3189 <*package>
3190 \ProvidesExplPackage
3191   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3192 \package_check_loaded_expl:
3193 </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
\int_eval_end: 3194 \cs_new_eq:NN \int_value:w \tex_number:D
\if_num:w       3195 \cs_new_eq:NN \int_eval:w \etex_numexpr:D
\if_int_odd:w   3196 \cs_new_eq:NN \int_eval_end: \tex_relax:D
\if_case:w      3197 \cs_new_eq:NN \if_num:w \tex_ifnum:D
                3198 \cs_new_eq:NN \if_int_odd:w \tex_ifodd:D
                3199 \cs_new_eq:NN \if_case:w \tex_ifcase:D
      (End definition for \int_value:w. This function is documented on page 69.)

```

182.1 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.

```

3200 <*initex>
3201 \cs_set:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3202 </initex>
3203 <*package>
3204 \cs_new:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3205 </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  3206 \cs_new:Npn \int_abs:n #1
            {
            3207   \int_value:w
            3208   \if_int_compare:w \int_eval:w #1 < \c_zero
            3209   -
            3210   \fi:
            3211   \int_eval:w #1 \int_eval_end:
            3212 }
            3213 \cs_new:Npn \int_max:nn #1#2
            {
            3214   \int_value:w \int_eval:w
            3215   \if_int_compare:w
            3216   \int_eval:w #1 > \int_eval:w #2 \int_eval_end:
            3217
            3218

```

```

3219         #1
3220     \else:
3221         #2
3222     \fi:
3223 \int_eval_end:
3224 }
3225 \cs_new:Npn \int_min:nn #1#2
3226 {
3227     \int_value:w \int_eval:w
3228     \if_int_compare:w
3229         \int_eval:w #1 < \int_eval:w #2 \int_eval_end:
3230     #1
3231 \else:
3232     #2
3233 \fi:
3234 \int_eval_end:
3235 }

```

(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.

```

3236 \cs_new:Npn \int_div_truncate:nn #1#2
3237 {
3238     \int_value:w \int_eval:w
3239     \if_int_compare:w \int_eval:w #1 = \c_zero
3240     0
3241 \else:
3242     ( #1 % )
3243     \if_int_compare:w \int_eval:w #1 < \c_zero
3244     \if_int_compare:w \int_eval:w #2 < \c_zero
3245     - ( #2 + % )
3246 \else:
3247     + ( #2 - % )
3248 \fi:
3249 \else:
3250     \if_int_compare:w \int_eval:w #2 < \c_zero
3251     + ( #2 + % )
3252 \else:
3253     - ( #2 - % )
3254 \fi:
3255 \fi: % ( (
3256     1 ) / 2 )
3257 \fi:
3258 / ( #2 )
3259 \int_eval_end:
3260 }

```

For the sake of completeness:

```

3261 \cs_new:Npn \int_div_round:nn #1#2 { \int_eval:n { ( #1 ) / ( #2 ) } }

```

Finally there's the modulus operation.

```

3262 \cs_new:Npn \int_mod:nn #1#2
3263 {
3264   \int_value:w \int_eval:w
3265     #1 - \int_div_truncate:nn {#1} {#2} * ( #2 )
3266   \int_eval_end:
3267 }

```

(End definition for `\int_div_truncate:nn`. This function is documented on page 60.)

182.2 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 3268 <*package>
3269 \cs_new_protected_nopar:Npn \int_new:N #1
3270 {
3271   \chk_if_free_cs:N #1
3272   \newcount #1
3273 }
3274 </package>
3275 \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 3276 \cs_new_protected_nopar:Npn \int_const:Nn #1#2
3277 {
3278   \int_compare:nNnTF {#2} > \c_minus_one
3279   {
3280     \int_compare:nNnTF {#2} > \c_max_register_int
3281     {
3282       \int_new:N #1
3283       \int_gset:Nn #1 {#2}
3284     }
3285     {
3286       \chk_if_free_cs:N #1
3287       \tex_global:D \tex_mathchardef:D #1 =
3288         \int_eval:w #2 \int_eval_end:
3289     }
3290   }
3291   {
3292     \int_new:N #1
3293     \int_gset:Nn #1 {#2}
3294   }
3295 }
3296 \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` 3297 `\cs_new_protected_nopar:Npn \int_zero:N #1 { #1 = \c_zero }`

`\int_gzero:N` 3298 `\cs_new_protected_nopar:Npn \int_gzero:N #1 { \tex_global:D #1 = \c_zero }`

`\int_gzero:c` 3299 `\cs_generate_variant:Nn \int_zero:N { c }`

3300 `\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` 3301 `\cs_new_protected_nopar:Npn \int_set_eq:NN #1#2 { #1 = #2 }`

`\int_set_eq:Nc` 3302 `\cs_generate_variant:Nn \int_set_eq:NN { c }`

`\int_set_eq:cc` 3303 `\cs_generate_variant:Nn \int_set_eq:NN { Nc , cc }`

`\int_gset_eq:NN` 3304 `\cs_new_protected_nopar:Npn \int_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }`

`\int_gset_eq:cN` 3305 `\cs_generate_variant:Nn \int_gset_eq:NN { c }`

`\int_gset_eq:Nc` 3306 `\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 ??.)

182.3 Setting and incrementing integers

`\int_add:Nn` Adding and subtracting to and from a counter ...

`\int_add:cn` 3307 `\cs_new_protected_nopar:Npn \int_add:Nn #1#2`

`\int_gadd:Nn` 3308 `{ \tex_advance:D #1 by \int_eval:w #2 \int_eval_end: }`

`\int_gadd:cn` 3309 `\cs_new_protected_nopar:Npn \int_sub:Nn #1#2`

`\int_sub:Nn` 3310 `{ \tex_advance:D #1 by - \int_eval:w #2 \int_eval_end: }`

`\int_sub:cn` 3311 `\cs_new_protected_nopar:Npn \int_gadd:Nn`

`\int_gsub:Nn` 3312 `{ \tex_global:D \int_add:Nn }`

`\int_gsub:cn` 3313 `\cs_new_protected_nopar:Npn \int_gsub:Nn`

3314 `{ \tex_global:D \int_sub:Nn }`

3315 `\cs_generate_variant:Nn \int_add:Nn { c }`

3316 `\cs_generate_variant:Nn \int_gadd:Nn { c }`

3317 `\cs_generate_variant:Nn \int_sub:Nn { c }`

3318 `\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` 3319 `\cs_new_protected_nopar:Npn \int_incr:N #1`

`\int_gincr:N` 3320 `{ \tex_advance:D #1 \c_one }`

`\int_gincr:c` 3321 `\cs_new_protected_nopar:Npn \int_decr:N #1`

`\int_decr:N` 3322 `{ \tex_advance:D #1 \c_minus_one }`

`\int_decr:c` 3323 `\cs_new_protected_nopar:Npn \int_gincr:N`

`\int_gdecr:N` 3324 `{ \tex_global:D \int_incr:N }`

`\int_gdecr:c` 3325 `\cs_new_protected_nopar:Npn \int_gdecr:N`

3326 `{ \tex_global:D \int_decr:N }`

3327 `\cs_generate_variant:Nn \int_incr:N { c }`

3328 `\cs_generate_variant:Nn \int_decr:N { c }`

3329 `\cs_generate_variant:Nn \int_gincr:N { c }`

3330 `\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` 3331 `\cs_new_protected_nopar:Npn \int_set:Nn #1#2`
`\int_gset:cn` 3332 `{ #1 ~ \int_eval:w #2\int_eval_end: }`
3333 `\cs_new_protected_nopar:Npn \int_gset:Nn { \tex_global:D \int_set:Nn }`
3334 `\cs_generate_variant:Nn \int_set:Nn { c }`
3335 `\cs_generate_variant:Nn \int_gset:Nn { c }`
(End definition for \int_set:Nn and \int_set:cn. These functions are documented on page ??.)

182.4 Using integers

`\int_use:N` Here is how counters are accessed:
`\int_use:c` 3336 `\cs_new_eq:NN \int_use:N \tex_the:D`
3337 `\cs_new_nopar: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 ??.)

182.5 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_=:w      \int_compare_p:n { 5 + \l_tmpa_int != 4 - \l_tmpb_int }
```

`int_compare_!=:w` In other words, we want to somehow add the missing `\int_eval:w` where required.
`int_compare_<:w` We can start evaluating from the left using `\int_eval:w`, and we know that since the
`int_compare_>:w` relation symbols `<`, `>`, `=` and `!` are not allowed in such expressions, they will terminate
`int_compare_<=:w` the expression. Therefore, we first let TeX evaluate this left hand side of the (in)equality.
`int_compare_>=:w`

```
3338 \prg_new_conditional:Npnn \int_compare:n #1 { p , T , F , TF }
3339 { \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.

```
3340 \cs_new:Npn \int_compare_aux:nw #1#2 \q_stop
3341 {
3342   \exp_after:wN \int_compare_aux:Nw
3343   \int_to_roman:w
3344   \if:w #1 -
3345   \else:
3346     -
3347   \fi:
3348   #1#2 \q_mark #1#2 \q_stop
3349 }
```

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.

```

3350 \cs_new:Npn \int_compare_aux:Nw #1#2#3 \q_mark
3351 { \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:

```

3352 \cs_new:cpn { int_compare_=:w } #1 = #2 \q_stop
3353 {
3354   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3355   \prg_return_true:
3356   \else:
3357   \prg_return_false:
3358   \fi:
3359 }

```

So is the one using == we just have to use == in the parameter text.

```

3360 \cs_new:cpn { int_compare_==:w } #1 == #2 \q_stop
3361 {
3362   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3363   \prg_return_true:
3364   \else:
3365   \prg_return_false:
3366   \fi:
3367 }

```

Not equal is just about reversing the truth value.

```

3368 \cs_new:cpn { int_compare_!=:w } #1 != #2 \q_stop
3369 {
3370   \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3371   \prg_return_false:
3372   \else:
3373   \prg_return_true:
3374   \fi:
3375 }

```

Less than and greater than are also straight forward.

```

3376 \cs_new:cpn { int_compare_<:w } #1 < #2 \q_stop
3377 {
3378   \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3379   \prg_return_true:
3380   \else:
3381   \prg_return_false:
3382   \fi:
3383 }
3384 \cs_new:cpn { int_compare_>:w } #1 > #2 \q_stop
3385 {
3386   \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3387   \prg_return_true:

```

```

3388     \else:
3389         \prg_return_false:
3390     \fi:
3391 }

```

The less than or equal operation is just the opposite of the greater than operation. *Vice versa* for less than or equal.

```

3392 \cs_new:cpn { int_compare_<=:w } #1 <= #2 \q_stop
3393 {
3394     \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3395         \prg_return_false:
3396     \else:
3397         \prg_return_true:
3398     \fi:
3399 }
3400 \cs_new:cpn { int_compare_>=:w } #1 >= #2 \q_stop
3401 {
3402     \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3403         \prg_return_false:
3404     \else:
3405         \prg_return_true:
3406     \fi:
3407 }

```

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

`\int_compare:nNn` More efficient but less natural in typing.

```

3408 \prg_new_conditional:Npnn \int_compare:nNn #1#2#3 { p , T , F , TF}
3409 {
3410     \if_int_compare:w \int_eval:w #1 #2 \int_eval:w #3 \int_eval_end:
3411         \prg_return_true:
3412     \else:
3413         \prg_return_false:
3414     \fi:
3415 }

```

(End definition for \int_compare:nNn. This function is documented on page 62.)

`\int_if_odd:n` A predicate function.

`\int_if_even:n`

```

3416 \prg_new_conditional:Npnn \int_if_odd:n #1 { p , T , F , TF}
3417 {
3418     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3419         \prg_return_true:
3420     \else:
3421         \prg_return_false:
3422     \fi:
3423 }
3424 \prg_new_conditional:Npnn \int_if_even:n #1 { p , T , F , TF}
3425 {
3426     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3427         \prg_return_false:

```

```

3428     \else:
3429         \prg_return_true:
3430     \fi:
3431 }

```

(End definition for \int_if_odd:n. This function is documented on page 62.)

182.6 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 3432 \cs_new:Npn \int_while_do:nn #1#2
\int_until_do:nn 3433 {
\int_do_while:nn 3434     \int_compare:nT {#1}
\int_do_until:nn 3435     {
3436         #2
3437         \int_while_do:nn {#1} {#2}
3438     }
3439 }
3440 \cs_new:Npn \int_until_do:nn #1#2
3441 {
3442     \int_compare:nF {#1}
3443     {
3444         #2
3445         \int_until_do:nn {#1} {#2}
3446     }
3447 }
3448 \cs_new:Npn \int_do_while:nn #1#2
3449 {
3450     #2
3451     \int_compare:nT {#1}
3452     { \int_do_while:nn {#1} {#2} }
3453 }
3454 \cs_new:Npn \int_do_until:nn #1#2
3455 {
3456     #2
3457     \int_compare:nF {#1}
3458     { \int_do_until:nn {#1} {#2} }
3459 }

```

(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 3460 \cs_new:Npn \int_while_do:nNnn #1#2#3#4
\int_until_do:nNnn 3461 {
\int_do_while:nNnn 3462     \int_compare:nNnT {#1} #2 {#3}
\int_do_until:nNnn 3463     {
3464         #4
3465         \int_while_do:nNnn {#1} #2 {#3} {#4}
3466     }
3467 }

```

```

3468 \cs_new:Npn \int_until_do:nNnn #1#2#3#4
3469 {
3470   \int_compare:nNnF {#1} #2 {#3}
3471   {
3472     #4
3473     \int_until_do:nNnn {#1} #2 {#3} {#4}
3474   }
3475 }
3476 \cs_new:Npn \int_do_while:nNnn #1#2#3#4
3477 {
3478   #4
3479   \int_compare:nNnT {#1} #2 {#3}
3480   { \int_do_while:nNnn {#1} #2 {#3} {#4} }
3481 }
3482 \cs_new:Npn \int_do_until:nNnn #1#2#3#4
3483 {
3484   #4
3485   \int_compare:nNnF {#1} #2 {#3}
3486   { \int_do_until:nNnn {#1} #2 {#3} {#4} }
3487 }

```

(End definition for \int_while_do:nNnn. This function is documented on page 63.)

182.7 Formatting integers

\int_to_arabic:n Nothing exciting here.

```

3488 \cs_new_nopar: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.

```

3489 \cs_new:Npn \int_to_symbols:nnn #1#2#3
3490 {
3491   \int_compare:nNnTF {#1} > {#2}
3492   {
3493     \exp_args:NNo \exp_args:No \int_to_symbols_aux:nnnn
3494     {
3495       \prg_case_int:nnn
3496       { 1 + \int_mod:nn { #1 - 1 } {#2} }
3497       {#3} { }
3498     }
3499     {#1} {#2} {#3}
3500   }
3501   { \prg_case_int:nnn {#1} {#3} { } }

```

```

3502 }
3503 \cs_new:Npn \int_to_symbols_aux:nnnn #1#2#3#4
3504 {
3505   \exp_args:Nf \int_to_symbols:nnn
3506     { \int_div_truncate:nn { #2 - 1 } {#3} } {#3} {#4}
3507   #1
3508 }

```

(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.

```

3509 \cs_new:Npn \int_to_alph:n #1
3510 {
3511   \int_to_symbols:nnn {#1} { 26 }
3512   {
3513     { 1 } { a }
3514     { 2 } { b }
3515     { 3 } { c }
3516     { 4 } { d }
3517     { 5 } { e }
3518     { 6 } { f }
3519     { 7 } { g }
3520     { 8 } { h }
3521     { 9 } { i }
3522     { 10 } { j }
3523     { 11 } { k }
3524     { 12 } { l }
3525     { 13 } { m }
3526     { 14 } { n }
3527     { 15 } { o }
3528     { 16 } { p }
3529     { 17 } { q }
3530     { 18 } { r }
3531     { 19 } { s }
3532     { 20 } { t }
3533     { 21 } { u }
3534     { 22 } { v }
3535     { 23 } { w }
3536     { 24 } { x }
3537     { 25 } { y }
3538     { 26 } { z }
3539   }
3540 }
3541 \cs_new:Npn \int_to_Alph:n #1
3542 {
3543   \int_to_symbols:nnn {#1} { 26 }
3544   {
3545     { 1 } { A }
3546     { 2 } { B }

```

```

3547      { 3 } { C }
3548      { 4 } { D }
3549      { 5 } { E }
3550      { 6 } { F }
3551      { 7 } { G }
3552      { 8 } { H }
3553      { 9 } { I }
3554      { 10 } { J }
3555      { 11 } { K }
3556      { 12 } { L }
3557      { 13 } { M }
3558      { 14 } { N }
3559      { 15 } { O }
3560      { 16 } { P }
3561      { 17 } { Q }
3562      { 18 } { R }
3563      { 19 } { S }
3564      { 20 } { T }
3565      { 21 } { U }
3566      { 22 } { V }
3567      { 23 } { W }
3568      { 24 } { X }
3569      { 25 } { Y }
3570      { 26 } { Z }
3571    }
3572  }

```

(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`

```

3573 \cs_new:Npn \int_to_base:nn #1
3574 { \exp_args:Nf \int_to_base_aux_i:nn { \int_eval:n {#1} } }
3575 \cs_new:Npn \int_to_base_aux_i:nn #1#2
3576 {
3577   \int_compare:nNnTF {#1} < \c_zero
3578   { \exp_args:No \int_to_base_aux_ii:nnN { \use_none:n #1 } {#2} - }
3579   { \int_to_base_aux_ii:nnN {#1} {#2} \c_empty_tl }
3580 }

```

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.

```

3581 \cs_new:Npn \int_to_base_aux_ii:nnN #1#2#3
3582 {

```



```

3583 \int_compare:nNnTF {#1} < {#2}
3584 { \exp_last_unbraced:Nf #3 { \int_to_letter:n {#1} } }
3585 {
3586   \exp_args:Nf \int_to_base_aux_iii:nnnN
3587   { \int_to_letter:n { \int_mod:nn {#1} {#2} } }
3588   {#1}
3589   {#2}
3590   #3
3591 }
3592 }
3593 \cs_new:Npn \int_to_base_aux_iii:nnnN #1#2#3#4
3594 {
3595   \exp_args:Nf \int_to_base_aux_ii:nnN
3596   { \int_div_truncate:nn {#2} {#3} }
3597   {#3}
3598   #4
3599   #1
3600 }

```

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:`.

```

3601 \cs_new:Npn \int_to_letter:n #1
3602 {
3603   \exp_after:wN \exp_after:wN
3604   \if_case:w \int_eval:w #1 - \c_ten \int_eval_end:
3605   A
3606   \or: B
3607   \or: C
3608   \or: D
3609   \or: E
3610   \or: F
3611   \or: G
3612   \or: H
3613   \or: I
3614   \or: J
3615   \or: K
3616   \or: L
3617   \or: M
3618   \or: N
3619   \or: O
3620   \or: P
3621   \or: Q
3622   \or: R
3623   \or: S
3624   \or: T
3625   \or: U

```

```

3626     \or: V
3627     \or: W
3628     \or: X
3629     \or: Y
3630     \or: Z
3631     \else: \int_value:w \int_eval:w #1 \exp_after:wN \int_eval_end:
3632     \fi:
3633 }

```

(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

```

```

3634 \cs_new:Npn \int_to_binary:n #1
3635 { \int_to_base:nn {#1} { 2 } }
3636 \cs_new:Npn \int_to_hexadecimal:n #1
3637 { \int_to_base:nn {#1} { 16 } }
3638 \cs_new:Npn \int_to_octal:n #1
3639 { \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 3640 \cs_new_nopar:Npn \int_to_roman:n #1
\int_to_roman_v:w 3641 {
\int_to_roman_x:w 3642     \exp_after:wN \int_to_roman_aux:N
\int_to_roman_l:w 3643     \int_to_roman:w \int_eval:n {#1} Q
\int_to_roman_c:w 3644 }
\int_to_roman_d:w 3645 \cs_new_nopar:Npn \int_to_roman_aux:N #1
\int_to_roman_m:w 3646 {
\int_to_roman_Q:w 3647     \use:c { int_to_roman_ #1 :w }
\int_to_Roman_i:w 3648     \int_to_roman_aux:N
\int_to_Roman_v:w 3649 }
\int_to_Roman_x:w 3650 \cs_new_nopar:Npn \int_to_Roman:n #1
\int_to_Roman_l:w 3651 {
\int_to_Roman_c:w 3652     \exp_after:wN \int_to_Roman_aux:N
\int_to_Roman_d:w 3653     \int_to_roman:w \int_eval:n {#1} Q
\int_to_Roman_m:w 3654 }
\int_to_Roman_Q:w 3655 \cs_new_nopar:Npn \int_to_Roman_aux:N #1
3656 {
3657     \use:c { int_to_Roman_ #1 :w }
3658     \int_to_Roman_aux:N
3659 }
3660 \cs_new_nopar:Npn \int_to_roman_i:w { i }
3661 \cs_new_nopar:Npn \int_to_roman_v:w { v }
3662 \cs_new_nopar:Npn \int_to_roman_x:w { x }
3663 \cs_new_nopar:Npn \int_to_roman_l:w { l }
3664 \cs_new_nopar:Npn \int_to_roman_c:w { c }

```

```

3665 \cs_new_nopar:Npn \int_to_roman_d:w { d }
3666 \cs_new_nopar:Npn \int_to_roman_m:w { m }
3667 \cs_new_nopar:Npn \int_to_roman_Q:w #1 { }
3668 \cs_new_nopar:Npn \int_to_Roman_i:w { I }
3669 \cs_new_nopar:Npn \int_to_Roman_v:w { V }
3670 \cs_new_nopar:Npn \int_to_Roman_x:w { X }
3671 \cs_new_nopar:Npn \int_to_Roman_l:w { L }
3672 \cs_new_nopar:Npn \int_to_Roman_c:w { C }
3673 \cs_new_nopar:Npn \int_to_Roman_d:w { D }
3674 \cs_new_nopar:Npn \int_to_Roman_m:w { M }
3675 \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 ??.)

182.8 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.

```

3676 \cs_new:Npn \int_get_sign:n #1
3677 {
3678   \int_get_sign_and_digits_aux:nNNN {#1}
3679   \c_true_bool \c_true_bool \c_false_bool
3680 }
3681 \cs_new:Npn \int_get_digits:n #1
3682 {
3683   \int_get_sign_and_digits_aux:nNNN {#1}
3684   \c_true_bool \c_false_bool \c_true_bool
3685 }

```

The auxiliary loops through, finding sign tokens and removing them. The sign itself is carried through as a flag.

```

3686 \cs_new:Npn \int_get_sign_and_digits_aux:nNNN #1#2#3#4
3687 {
3688   \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} -
3689   {
3690     \bool_if:NTF #2
3691     {
3692       \int_get_sign_and_digits_aux:oNNN
3693       { \use_none:n #1 } \c_false_bool #3#4
3694     }
3695     {
3696       \int_get_sign_and_digits_aux:oNNN
3697       { \use_none:n #1 } \c_true_bool #3#4
3698     }
3699   }
3700   {
3701     \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} +

```

```

3702         { \int_get_sign_and_digits_aux:oNNN { \use_none:n #1 } #2#3#4 }
3703         {
3704             \bool_if:NT #3 { \bool_if:NF #2 - }
3705             \bool_if:NT #4 {#1}
3706         }
3707     }
3708 }
3709 \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`

```

3710 \cs_new:Npn \int_from_alph:n #1
3711 {
3712     \int_eval:n
3713     {
3714         \int_get_sign:n {#1}
3715         \exp_args:Nf \int_from_alph_aux:n { \int_get_digits:n {#1} }
3716     }
3717 }
3718 \cs_new:Npn \int_from_alph_aux:n #1
3719 { \int_from_alph_aux:nN { 0 } #1 \q_nil }
3720 \cs_new:Npn \int_from_alph_aux:nN #1#2
3721 {
3722     \quark_if_nil:NTF #2
3723     {#1}
3724     {
3725         \exp_args:Nf \int_from_alph_aux:nN
3726         { \int_eval:n { #1 * 26 + \int_from_alph_aux:N #2 } }
3727     }
3728 }
3729 \cs_new:Npn \int_from_alph_aux:N #1
3730 { \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`

```

3731 \cs_new:Npn \int_from_base:nn #1#2
3732 {
3733     \int_eval:n
3734     {
3735         \int_get_sign:n {#1}
3736         \exp_args:Nf \int_from_base_aux:nn
3737         { \int_get_digits:n {#1} } {#2}
3738     }
3739 }
3740 \cs_new:Npn \int_from_base_aux:nn #1#2
3741 { \int_from_base_aux:nnN { 0 } { #2 } #1 \q_nil }

```

```

3742 \cs_new:Npn \int_from_base_aux:nnN #1#2#3
3743 {
3744   \quark_if_nil:NTF #3
3745   {#1}
3746   {
3747     \exp_args:Nf \int_from_base_aux:nnN
3748     { \int_eval:n { #1 * #2 + \int_from_base_aux:N #3 } }
3749     {#2}
3750   }
3751 }

```

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.

```

3752 \cs_new:Npn \int_from_base_aux:N #1
3753 {
3754   \int_compare:nNnTF { '#1 } < { 58 }
3755   {#1}
3756   {
3757     \int_eval:n
3758     { '#1 - \int_compare:nNnTF { '#1 } < { 91 } { 55 } { 87 } }
3759   }
3760 }

```

(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.

```

3761 \cs_new:Npn \int_from_binary:n #1
3762 { \int_from_base:nn {#1} \c_two }
3763 \cs_new:Npn \int_from_hexadecimal:n #1
3764 { \int_from_base:nn {#1} \c_sixteen }
3765 \cs_new:Npn \int_from_octal:n #1
3766 { \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.

```

3767 \int_const:cn { c_int_from_roman_i_int } { 1 }
3768 \int_const:cn { c_int_from_roman_v_int } { 5 }
3769 \int_const:cn { c_int_from_roman_x_int } { 10 }
3770 \int_const:cn { c_int_from_roman_l_int } { 50 }
3771 \int_const:cn { c_int_from_roman_c_int } { 100 }
3772 \int_const:cn { c_int_from_roman_d_int } { 500 }
3773 \int_const:cn { c_int_from_roman_m_int } { 1000 }
3774 \int_const:cn { c_int_from_roman_I_int } { 1 }
3775 \int_const:cn { c_int_from_roman_V_int } { 5 }
3776 \int_const:cn { c_int_from_roman_X_int } { 10 }
3777 \int_const:cn { c_int_from_roman_L_int } { 50 }
3778 \int_const:cn { c_int_from_roman_C_int } { 100 }
3779 \int_const:cn { c_int_from_roman_D_int } { 500 }
3780 \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`

```

3781 \cs_new_nopar:Npn \int_from_roman:n #1
3782 {
3783   \tl_if_blank:nF {#1}
3784   {
3785     \exp_after:wN \int_from_roman_end:w
3786     \int_value:w \int_eval:w
3787     \int_from_roman_aux:NN #1 Q \q_stop
3788   }
3789 }
3790 \cs_new_nopar:Npn \int_from_roman_aux:NN #1#2
3791 {
3792   \str_if_eq:nnTF {#1} { Q }
3793   {#1#2}
3794   {
3795     \str_if_eq:nnTF {#2} { Q }
3796     {
3797       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3798       { \int_from_roman_clean_up:w }
3799       +
3800       \use:c { c_int_from_roman_ #1 _int }
3801       #2
3802     }
3803     {
3804       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3805       { \int_from_roman_clean_up:w }
3806       \cs_if_exist:cF { c_int_from_roman_ #2 _int }
3807       { \int_from_roman_clean_up:w }
3808       \int_compare:nNnTF
3809       { \use:c { c_int_from_roman_ #1 _int } }
3810       <
3811       { \use:c { c_int_from_roman_ #2 _int } }
3812       {
3813         + \use:c { c_int_from_roman_ #2 _int }
3814         - \use:c { c_int_from_roman_ #1 _int }
3815         \int_from_roman_aux:NN
3816       }
3817       {
3818         + \use:c { c_int_from_roman_ #1 _int }
3819         \int_from_roman_aux:NN #2
3820       }
3821     }
3822   }
3823 }
3824 \cs_new_nopar:Npn \int_from_roman_end:w #1 Q #2 \q_stop
3825 { \tl_if_empty:nTF {#2} {#1} {#2} }

```

```
3826 \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 ??.)
```

182.9 Viewing integer

```
\int_show:N
\int_show:c 3827 \cs_new_eq:NN \int_show:N \kernel_register_show:N
3828 \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 ??.)
```

182.10 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` Low-number values not previously defined.
`\c_one`
`\c_sixteen`
`\c_two`

```
3829 \int_const:Nn \c_one      { 1 }
3830 \int_const:Nn \c_two      { 2 }
3831 \int_const:Nn \c_three    { 3 }
3832 \int_const:Nn \c_four     { 4 }
3833 \int_const:Nn \c_five     { 5 }
3834 \int_const:Nn \c_eight    { 8 }
3835 \int_const:Nn \c_nine     { 9 }
3836 \int_const:Nn \c_ten      { 10 }
3837 \int_const:Nn \c_eleven   { 11 }
3838 \int_const:Nn \c_thirteen { 13 }
3839 \int_const:Nn \c_fourteen { 14 }
3840 \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.

```
3841 \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`

```
3842 \int_const:Nn \c_two_hundred_fifty_five { 255 }
3843 \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` 3844 `\int_const:Nn \c_one_hundred { 100 }`
`\c_ten_thousand` 3845 `\int_const:Nn \c_one_thousand { 1000 }`
3846 `\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 $2^{31} - 1$
3847 `\int_const:Nn \c_max_int { 2 147 483 647 }`
(End definition for `\c_max_int`. This function is documented on page 67.)

182.11 Scratch integers

`\l_tmpa_int` We provide four local and two global scratch counters, maybe we need more or less.
`\l_tmpb_int` 3848 `\int_new:N \l_tmpa_int`
`\l_tmpc_int` 3849 `\int_new:N \l_tmpb_int`
`\g_tmpa_int` 3850 `\int_new:N \l_tmpc_int`
`\g_tmpb_int` 3851 `\int_new:N \g_tmpa_int`
3852 `\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.)

182.12 Registers for earlier modules

Needed from other modules:

3853 `\int_new:N \g_seq_nesting_depth_int`
3854 `\int_new:N \g_tl_inline_level_int`

182.13 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` 3855 `\cs_new_eq:NN \int_convert_from_base_ten:nn \int_to_base:nn`
`\int_convert_to_base_ten:nn` 3856 `\cs_new_eq:NN \int_convert_to_symbols:nnn \int_to_symbols:nnn`
3857 `\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` 3858 `\cs_new_nopar:Npn \int_to_symbol:n`
`\int_to_symbol_text:n` 3859 `{`
3860 `\scan_align_safe_stop:`
3861 `\mode_if_math:TF`
3862 `{ \int_to_symbol_math:n }`
3863 `{ \int_to_symbol_text:n }`
3864 `}`
3865 `\cs_new:Npn \int_to_symbol_math:n #1`
3866 `{`


```

3867 \int_to_symbols:nnn {#1} { 9 }
3868 {
3869   { 1 } {          * }
3870   { 2 } {        \dagger }
3871   { 3 } {       \ddagger }
3872   { 4 } {     \mathsection }
3873   { 5 } {   \mathparagraph }
3874   { 6 } {         \l }
3875   { 7 } {         ** }
3876   { 8 } {   \dagger \dagger }
3877   { 9 } { \ddagger \ddagger }
3878 }
3879 }
3880 \cs_new:Npn \int_to_symbol_text:n #1
3881 {
3882   \int_to_symbols:nnn {#1} { 9 }
3883   {
3884     { 1 } {          \textasteriskcentered }
3885     { 2 } {        \textdagger }
3886     { 3 } {       \textdaggerdbl }
3887     { 4 } {     \textsection }
3888     { 5 } {   \textparagraph }
3889     { 6 } {         \textbardbl }
3890     { 7 } { \textasteriskcentered \textasteriskcentered }
3891     { 8 } {        \textdagger \textdagger }
3892     { 9 } {       \textdaggerdbl \textdaggerdbl }
3893   }
3894 }
3895 \end{definition for \int_to_symbol:n. This function is documented on page ??.)
\initex | package

```

183 l3skip implementation

```

3896 \*initex | package
3897 \*package
3898 \ProvidesExplPackage
3899   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3900 \package_check_loaded_expl:
3901 \package

```

183.1 Length primitives renamed

```

\if_dim:w Primitives renamed.
\dim_eval:w
\dim_eval_end:
3902 \cs_new_eq:NN \if_dim:w \tex_ifdim:D
3903 \cs_new_eq:NN \dim_eval:w \etex_dimexpr:D
3904 \cs_new_eq:NN \dim_eval_end: \tex_relax:D
3905 \end{definition for \if_dim:w. This function is documented on page ??.)

```

183.2 Creating and initialising dim variables

`\dim_new:N` Allocating $\langle dim \rangle$ registers ...

```
\dim_new:c 3905 \*package>
3906 \cs_new_protected_nopar:Npn \dim_new:N #1
3907 {
3908     \chk_if_free_cs:N #1
3909     \newdimen #1
3910 }
3911 \</package>
3912 \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 3913 \cs_new_protected_nopar:Npn \dim_zero:N #1 { #1 \c_zero_dim }
\dim_gzero:N 3914 \cs_new_protected_nopar:Npn \dim_gzero:N { \tex_global:D \dim_zero:N }
\dim_gzero:c 3915 \cs_generate_variant:Nn \dim_zero:N { c }
3916 \cs_generate_variant:Nn \dim_gzero:N { c }
(End definition for \dim_zero:N and \dim_zero:c. These functions are documented on page ??.)
```

183.3 Setting dim variables

`\dim_set:Nn` Setting dimensions is easy enough.

```
\dim_set:cn 3917 \cs_new_protected_nopar:Npn \dim_set:Nn #1#2
\dim_gset:Nn 3918 { #1 ~ \dim_eval:w #2 \dim_eval_end: }
\dim_gset:cn 3919 \cs_new_protected_nopar:Npn \dim_gset:Nn { \tex_global:D \dim_set:Nn }
3920 \cs_generate_variant:Nn \dim_set:Nn { c }
3921 \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 3922 \cs_new_protected_nopar:Npn \dim_set_eq:NN #1#2 { #1 = #2 }
\dim_set_eq:Nc 3923 \cs_generate_variant:Nn \dim_set_eq:NN { c }
\dim_set_eq:cc 3924 \cs_generate_variant:Nn \dim_set_eq:NN { Nc , cc }
\dim_gset_eq:NN 3925 \cs_new_protected_nopar:Npn \dim_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\dim_gset_eq:cN 3926 \cs_generate_variant:Nn \dim_gset_eq:NN { c }
\dim_gset_eq:Nc 3927 \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 3928 \cs_new_protected_nopar:Npn \dim_set_max:Nn #1#2
\dim_set_min:cn 3929 { \dim_compare:nNnT {#1} < {#2} { \dim_set:Nn #1 {#2} } }
\dim_gset_max:Nn 3930 \cs_new_protected_nopar:Npn \dim_gset_max:Nn #1#2
\dim_gset_max:cn 3931 { \dim_compare:nNnT {#1} < {#2} { \dim_gset:Nn #1 {#2} } }
\dim_gset_min:Nn 3932 \cs_new_protected_nopar:Npn \dim_set_min:Nn #1#2
\dim_gset_min:cn 3933 { \dim_compare:nNnT {#1} > {#2} { \dim_set:Nn #1 {#2} } }
3934 \cs_new_protected_nopar:Npn \dim_gset_min:Nn #1#2
```

```

3935 { \dim_compare:nNnT {#1} > {#2} { \dim_gset:Nn #1 {#2} } }
3936 \cs_generate_variant:Nn \dim_set_max:Nn { c }
3937 \cs_generate_variant:Nn \dim_gset_max:Nn { c }
3938 \cs_generate_variant:Nn \dim_set_min:Nn { c }
3939 \cs_generate_variant:Nn \dim_gset_min:Nn { c }

```

(End definition for `\dim_set_max:Nn` and `\dim_set_max:cn`. These functions are documented on page ??.)

`\dim_add:Nn` Using by here deals with the (incorrect) case `\dimen123`.

```

\dim_add:cn
\dim_gadd:Nn
\dim_gadd:cn
\dim_sub:Nn
\dim_sub:cn
\dim_gsub:Nn
\dim_gsub:cn
3940 \cs_new_protected_nopar:Npn \dim_add:Nn #1#2
3941 { \tex_advance:D #1 by \dim_eval:w #2 \dim_eval_end: }
3942 \cs_new_protected_nopar:Npn \dim_gadd:Nn { \tex_global:D \dim_add:Nn }
3943 \cs_generate_variant:Nn \dim_add:Nn { c }
3944 \cs_generate_variant:Nn \dim_gadd:Nn { c }
3945 \cs_new_protected_nopar:Npn \dim_sub:Nn #1#2
3946 { \tex_advance:D #1 by - \dim_eval:w #2 \dim_eval_end: }
3947 \cs_new_protected_nopar:Npn \dim_gsub:Nn { \tex_global:D \dim_sub:Nn }
3948 \cs_generate_variant:Nn \dim_sub:Nn { c }
3949 \cs_generate_variant:Nn \dim_gsub:Nn { c }

```

(End definition for `\dim_add:Nn` and `\dim_add:cn`. These functions are documented on page ??.)

183.4 Utilities for dimension calculations

`\dim_ratio:nn` `\dim_ratio_aux:n` With dimension expressions, something like `10 pt * (5 pt / 10 pt)` will not work. Instead, the ratio part needs to be converted to an integer expression. Using `\int_value:w` forces everything into `sp`, avoiding any decimal parts.

```

3950 \cs_new_nopar:Npn \dim_ratio:nn #1#2
3951 { \dim_ratio_aux:n {#1} / \dim_ratio_aux:n {#2} }
3952 \cs_new_nopar:Npn \dim_ratio_aux:n #1
3953 { \int_value:w \dim_eval:w #1 \dim_eval_end: }

```

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

183.5 Dimension expression conditionals

```

\dim_compare_p:nNn
\dim_compare:nNn
3954 \prg_new_conditional:Npnn \dim_compare:nNn #1#2#3 { p , T , F , TF }
3955 {
3956   \if_dim:w \dim_eval:w #1 #2 \dim_eval:w #3 \dim_eval_end:
3957   \prg_return_true: \else: \prg_return_false: \fi:
3958 }

```

(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.] Comparison tests using a simple syntax where only one set of braces is required and additional operators such as `!=` and `>=` are supported. First some notes on the idea behind this.

`\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_>=:nw`

`\dim_compare_>=:nw` `\dim_compare_>=:nw`

In other words, we want to somehow add the missing `\dim_eval:w` where required. We can start evaluating from the left using `\dim_use:N \dim_eval:w`, and we know that since the relation symbols `<`, `>`, `=` and `!` are not allowed in such expressions, they will terminate the expression. Therefore, we first let \TeX evaluate this left hand side of the (in)equality.

Eventually, we will convert the relation symbol to the appropriate version of `\if_dim:w`, and add `\dim_eval:w` after it. We optimize by placing the end-code already here: this avoids needless grabbing of arguments later.

```

3959 \prg_new_conditional:Npnn \dim_compare:n #1 { p , T , F , TF }
3960 {
3961   \exp_after:wN \dim_compare_aux:wNN \dim_use:N \dim_eval:w #1
3962   \dim_eval_end:
3963   \prg_return_true:
3964   \else:
3965     \prg_return_false:
3966   \fi:
3967 }

```

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.

```

3968 \exp_args:Nno \use:nn
3969 { \cs_new:Npn \dim_compare_aux:wNN #1 }
3970 { \tl_to_str:n { pt } }
3971 #2 #3
3972 {
3973   \use:c
3974   {
3975     dim_compare_ #2
3976     \if_meaning:w = #3 = \fi:
3977     :nw
3978   }
3979   { #1 pt } #3
3980 }

```

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.

```

3981 \cs_new:cpn { dim_compare_<:nw } #1 { \if_dim:w #1 < \dim_eval:w }
3982 \cs_new:cpn { dim_compare_=:nw } #1 { \if_dim:w #1 = \dim_eval:w }
3983 \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 =.

```

3984 \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.

```

3985 \cs_new:cpn {dim_compare_<=:nw} #1#2 {\reverse_if:N \if_dim:w #1 > \dim_eval:w}
3986 \cs_new:cpn {dim_compare_!=:nw} #1#2 {\reverse_if:N \if_dim:w #1 = \dim_eval:w}
3987 \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 ??.)

183.6 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_while_do:nn
\dim_until_do:nn
\dim_do_while:nn
\dim_do_until:nn
3988 \cs_set:Npn \dim_while_do:nn #1#2
3989 {
3990   \dim_compare:nT {#1}
3991   {
3992     #2
3993     \dim_while_do:nn {#1} {#2}
3994   }
3995 }
3996 \cs_set:Npn \dim_until_do:nn #1#2
3997 {
3998   \dim_compare:nF {#1}
3999   {
4000     #2
4001     \dim_until_do:nn {#1} {#2}
4002   }
4003 }
4004 \cs_set:Npn \dim_do_while:nn #1#2
4005 {
4006   #2
4007   \dim_compare:nT {#1}
4008   { \dim_do_while:nn {#1} {#2} }
4009 }
4010 \cs_set:Npn \dim_do_until:nn #1#2
4011 {
4012   #2
4013   \dim_compare:nF {#1}
4014   { \dim_do_until:nn {#1} {#2} }
4015 }

```

(End definition for \dim_while_do:nn. This function is documented on page 73.)

`\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
4016 \cs_set:Npn \dim_while_do:nNnn #1#2#3#4
4017 {
4018   \dim_compare:nNnT {#1} #2 {#3}
4019   {
4020     #4
4021     \dim_while_do:nNnn {#1} #2 {#3} {#4}
4022   }
4023 }
4024 \cs_set:Npn \dim_until_do:nNnn #1#2#3#4
4025 {
4026   \dim_compare:nNnF {#1} #2 {#3}
4027   {
4028     #4
4029     \dim_until_do:nNnn {#1} #2 {#3} {#4}
4030   }
4031 }
4032 \cs_set:Npn \dim_do_while:nNnn #1#2#3#4
4033 {
4034   #4
4035   \dim_compare:nNnT {#1} #2 {#3}
4036   { \dim_do_while:nNnn {#1} #2 {#3} {#4} }
4037 }
4038 \cs_set:Npn \dim_do_until:nNnn #1#2#3#4
4039 {
4040   #4
4041   \dim_compare:nNnF {#1} #2 {#3}
4042   { \dim_do_until:nNnn {#1} #2 {#3} {#4} }
4043 }

```

(End definition for `\dim_while_do:nNnn`. This function is documented on page 73.)

183.7 Using dim expressions and variables

`\dim_eval:n` Evaluating a dimension expression expandably.

```

4044 \cs_new_nopar:Npn \dim_eval:n #1
4045 { \dim_use:N \dim_eval:w #1 \dim_eval_end: }

```

(End definition for `\dim_eval:n`. This function is documented on page 74.)

`\dim_use:N` Accessing a $\langle dim \rangle$.

```

\dim_use:c
4046 \cs_new_eq:NN \dim_use:N \tex_the:D
4047 \cs_generate_variant:Nn \dim_use:N { c }

```

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

183.8 Viewing dim variables

`\dim_show:N` Diagnostics.
`\dim_show:c`

```

4048 \cs_new_eq:NN \dim_show:N \kernel_register_show:N
4049 \cs_generate_variant:Nn \dim_show:N { c }

```

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

183.9 Constant dimensions

`\c_zero_dim` The source for these depends on whether we are in package mode.

`\c_max_dim`

```

4050 \*initex
4051 \dim_new:N \c_zero_dim
4052 \dim_new:N \c_max_dim
4053 \dim_set:Nn \c_max_dim { 16383.99999 pt }
4054 \*initex
4055 \*package
4056 \cs_new_eq:NN \c_zero_dim \z@
4057 \cs_new_eq:NN \c_max_dim \maxdimen
4058 \*package

```

(End definition for \c_zero_dim. This function is documented on page 74.)

183.10 Scratch dimensions

`\l_tmpa_dim` We provide three local and two global scratch registers, maybe we need more or less.

`\l_tmpb_dim`

```

4059 \dim_new:N \l_tmpa_dim

```

`\l_tmpc_dim`

```

4060 \dim_new:N \l_tmpb_dim

```

`\g_tmpa_dim`

```

4061 \dim_new:N \l_tmpc_dim

```

`\g_tmpb_dim`

```

4062 \dim_new:N \g_tmpa_dim
4063 \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.)

183.11 Creating and initialising skip variables

`\skip_new:N` Allocation of a new internal registers.

`\skip_new:c`

```

4064 \*package
4065 \cs_new_protected_nopar:Npn \skip_new:N #1
4066 {
4067   \chk_if_free_cs:N #1
4068   \newskip #1
4069 }
4070 \*package
4071 \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` 4072 `\cs_new_protected_nopar:Npn \skip_zero:N #1 { #1 \c_zero_skip }`

`\skip_gzero:N` 4073 `\cs_new_protected_nopar:Npn \skip_gzero:N { \tex_global:D \skip_zero:N }`

`\skip_gzero:c` 4074 `\cs_generate_variant:Nn \skip_zero:N { c }`

4075 `\cs_generate_variant:Nn \skip_gzero:N { c }`

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

183.12 Setting skip variables

`\skip_set:Nn` Much the same as for dimensions.

`\skip_set:cn` 4076 `\cs_new_protected_nopar:Npn \skip_set:Nn #1#2`

`\skip_gset:Nn` 4077 `{ #1 ~ \etex_glueexpr:D #2 \scan_stop: }`

`\skip_gset:cn` 4078 `\cs_new_protected_nopar:Npn \skip_gset:Nn { \tex_global:D \skip_set:Nn }`

4079 `\cs_generate_variant:Nn \skip_set:Nn { c }`

4080 `\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` 4081 `\cs_new_protected_nopar:Npn \skip_set_eq:NN #1#2 { #1 = #2 }`

`\skip_set_eq:Nc` 4082 `\cs_generate_variant:Nn \skip_set_eq:NN { c }`

`\skip_set_eq:cc` 4083 `\cs_generate_variant:Nn \skip_set_eq:NN { Nc , cc }`

`\skip_gset_eq:NN` 4084 `\cs_new_protected_nopar:Npn \skip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }`

`\skip_gset_eq:cN` 4085 `\cs_generate_variant:Nn \skip_gset_eq:NN { c }`

`\skip_gset_eq:Nc` 4086 `\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` 4087 `\cs_new_protected_nopar:Npn \skip_add:Nn #1#2`

`\skip_gadd:Nn` 4088 `{ \tex_advance:D #1 by \etex_glueexpr:D #2 \scan_stop: }`

`\skip_gadd:cn` 4089 `\cs_new_protected_nopar:Npn \skip_gadd:Nn { \tex_global:D \skip_add:Nn }`

`\skip_sub:Nn` 4090 `\cs_generate_variant:Nn \skip_add:Nn { c }`

`\skip_sub:cn` 4091 `\cs_generate_variant:Nn \skip_gadd:Nn { c }`

`\skip_gsub:Nn` 4092 `\cs_new_protected_nopar:Npn \skip_sub:Nn #1#2`

`\skip_gsub:cn` 4093 `{ \tex_advance:D #1 by - \etex_glueexpr:D #2 \scan_stop: }`

4094 `\cs_new_protected_nopar:Npn \skip_gsub:Nn { \tex_global:D \skip_sub:Nn }`

4095 `\cs_generate_variant:Nn \skip_sub:Nn { c }`

4096 `\cs_generate_variant:Nn \skip_gsub:Nn { c }`

(End definition for `\skip_add:Nn` and `\skip_add:cn`. These functions are documented on page ??.)

183.13 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.

```

4097 \prg_new_conditional:Npnn \skip_if_eq:nn #1#2 { p , T , F , TF }
4098 {
4099   \if_int_compare:w
4100     \pdfTeX_strcmp:D { \skip_eval:n { #1 } } { \skip_eval:n { #2 } }
4101     = \c_zero
4102     \prg_return_true:
4103   \else:
4104     \prg_return_false:
4105   \fi:
4106 }

```

(End definition for `\skip_if_eq:nn`. This function is documented on page 76.)

`\skip_if_infinite_glue:n` With ε -TeX we all of a sudden get access to a lot of information we should otherwise consider ourselves lucky to get. One is the stretch and shrink components of a skip register and the order of those components. `\csskip_if_infinite_glue:nTF` tests it directly by looking at the stretch and shrink order. If either of the predicate functions return $\langle true \rangle$, `\bool_if:nTF` will return $\langle true \rangle$ and the logic test will take the true branch.

```

4107 \prg_new_conditional:Npnn \skip_if_infinite_glue:n #1 { p , T , F , TF }
4108 {
4109   \bool_if:nTF
4110     {
4111       \int_compare_p:nNn { \etex_gluestretchorder:D #1 } > \c_zero ||
4112       \int_compare_p:nNn { \etex_glueshrinkorder:D #1 } > \c_zero
4113     }
4114     { \prg_return_true: }
4115     { \prg_return_false: }
4116 }

```

(End definition for `\skip_if_infinite_glue:n`. This function is documented on page 76.)

183.14 Using skip expressions and variables

`\skip_eval:n` Evaluating a skip expression expandably.

```

4117 \cs_new_nopar:Npn \skip_eval:n #1
4118 { \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 $\langle skip \rangle$.

```

4119 \cs_new_eq:NN \skip_use:N \tex_the:D
4120 \cs_generate_variant:Nn \skip_use:N { c }

```

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

183.15 Inserting skips into the output

```

\skip_horizontal:N Inserting skips.
\skip_horizontal:c 4121 \cs_new_eq:NN \skip_horizontal:N \tex_hskip:D
\skip_horizontal:n 4122 \cs_new_nopar:Npn \skip_horizontal:n #1
                    4123 { \skip_horizontal:N \etex_glueexpr:D #1 \scan_stop: }
\skip_vertical:N 4124 \cs_new_eq:NN \skip_vertical:N \tex_vskip:D
\skip_vertical:c 4125 \cs_new_nopar:Npn \skip_vertical:n #1
\skip_vertical:n 4126 { \skip_vertical:N \etex_glueexpr:D #1 \scan_stop: }
                  4127 \cs_generate_variant:Nn \skip_horizontal:N { c }
                  4128 \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 ??.)

```

183.16 Viewing skip variables

```

\skip_show:N Diagnostics.
\skip_show:c 4129 \cs_new_eq:NN \skip_show:N \kernel_register_show:N
              4130 \cs_generate_variant:Nn \skip_show:N { c }

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

```

183.17 Constant skips

```

\c_zero_skip Skips with no rubber component are just dimensions
\c_max_skip 4131 \cs_new_eq:NN \c_zero_skip \c_zero_dim
              4132 \cs_new_eq:NN \c_max_skip \c_max_dim

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

```

183.18 Scratch skips

```

\l_tmpa_skip We provide three local and two global scratch registers, maybe we need more or less.
\l_tmpb_skip 4133 \skip_new:N \l_tmpa_skip
\l_tmpc_skip 4134 \skip_new:N \l_tmpb_skip
\g_tmpa_skip 4135 \skip_new:N \l_tmpc_skip
\g_tmpb_skip 4136 \skip_new:N \g_tmpa_skip
              4137 \skip_new:N \g_tmpb_skip

(End definition for \l_tmpa_skip, \l_tmpb_skip, and \l_tmpc_skip. These functions are docu-
mented on page ??.)

```

183.19 Creating and initialising muskip variables

```

\muskip_new:N And then we add muskips.
\muskip_new:c 4138 <*package>
              4139 \cs_new_protected_nopar:Npn \muskip_new:N #1
              4140 {
              4141   \chk_if_free_cs:N #1
              4142   \newmuskip #1

```

```

4143 }
4144 \end{package}
4145 \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 4146 \cs_new_protected_nopar:Npn \muskip_zero:N #1
\muskip_gzero:N 4147 { #1 \c_zero_muskip }
\muskip_gzero:c 4148 \cs_new_protected_nopar:Npn \muskip_gzero:N { \tex_global:D \muskip_zero:N }
4149 \cs_generate_variant:Nn \muskip_zero:N { c }
4150 \cs_generate_variant:Nn \muskip_gzero:N { c }
(End definition for \muskip_zero:N and \muskip_zero:c. These functions are documented on
page ??.)

```

183.20 Setting muskip variables

`\muskip_set:Nn` This should be pretty familiar.

```

\muskip_set:cn 4151 \cs_new_protected_nopar:Npn \muskip_set:Nn #1#2
\muskip_gset:Nn 4152 { #1 ~ \etex_muexpr:D #2 \scan_stop: }
\muskip_gset:cn 4153 \cs_new_protected_nopar:Npn \muskip_gset:Nn { \tex_global:D \muskip_set:Nn }
4154 \cs_generate_variant:Nn \muskip_set:Nn { c }
4155 \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 4156 \cs_new_protected_nopar:Npn \muskip_set_eq:NN #1#2 { #1 = #2 }
\muskip_set_eq:Nc 4157 \cs_generate_variant:Nn \muskip_set_eq:NN { c }
\muskip_set_eq:cc 4158 \cs_generate_variant:Nn \muskip_set_eq:NN { Nc , cc }
\muskip_gset_eq:NN 4159 \cs_new_protected_nopar:Npn \muskip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\muskip_gset_eq:cN 4160 \cs_generate_variant:Nn \muskip_gset_eq:NN { c }
\muskip_gset_eq:Nc 4161 \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 4162 \cs_new_protected_nopar:Npn \muskip_add:Nn #1#2
\muskip_gadd:Nn 4163 { \tex_advance:D #1 by \etex_muexpr:D #2 \scan_stop: }
\muskip_gadd:cn 4164 \cs_new_protected_nopar:Npn \muskip_gadd:Nn { \tex_global:D \muskip_add:Nn }
\muskip_sub:Nn 4165 \cs_generate_variant:Nn \muskip_add:Nn { c }
\muskip_sub:cn 4166 \cs_generate_variant:Nn \muskip_gadd:Nn { c }
\muskip_gsub:Nn 4167 \cs_new_protected_nopar:Npn \muskip_sub:Nn #1#2
\muskip_gsub:cn 4168 { \tex_advance:D #1 by - \etex_muexpr:D #2 \scan_stop: }
4169 \cs_new_protected_nopar:Npn \muskip_gsub:Nn { \tex_global:D \muskip_sub:Nn }
4170 \cs_generate_variant:Nn \muskip_sub:Nn { c }
4171 \cs_generate_variant:Nn \muskip_gsub:Nn { c }
(End definition for \muskip_add:Nn and \muskip_add:cn. These functions are documented on
page ??.)

```

183.21 Using muskip expressions and variables

`\muskip_eval:n` Evaluating a muskip expression expandably.

```
4172 \cs_new_nopar:Npn \muskip_eval:n #1
4173 { \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 4174 \cs_new_eq:NN \muskip_use:N \tex_the:D
4175 \cs_generate_variant:Nn \muskip_use:N { c }
(End definition for \muskip_use:N and \muskip_use:c. These functions are documented on page ??.)
```

183.22 Viewing muskip variables

`\muskip_show:N` Diagnostics.

```
\muskip_show:c 4176 \cs_new_eq:NN \muskip_show:N \kernel_register_show:N
4177 \cs_generate_variant:Nn \muskip_show:N { c }
(End definition for \muskip_show:N and \muskip_show:c. These functions are documented on page ??.)
```

183.23 Experimental skip functions

`\skip_split_finite_else_action:nnNN` This macro is useful when performing error checking in certain circumstances. If the $\langle skip \rangle$ register holds finite glue it sets #3 and #4 to the stretch and shrink component, resp. If it holds infinite glue set #3 and #4 to zero and issue the special action #2 which is probably an error message. Assignments are global.

```
4178 \cs_new_nopar:Npn \skip_split_finite_else_action:nnNN #1#2#3#4
4179 {
4180   \skip_if_infinite_glue:nTF {#1}
4181   {
4182     #3 = \c_zero_skip
4183     #4 = \c_zero_skip
4184     #2
4185   }
4186   {
4187     #3 = \etex_gluestretch:D #1 \scan_stop:
4188     #4 = \etex_glueshrink:D #1 \scan_stop:
4189   }
4190 }
(End definition for \skip_split_finite_else_action:nnNN. This function is documented on page 80.)
4191 \</initex | package>
```

184 l3tl implementation

```

4192 <*initex | package>
4193 <*package>
4194 \ProvidesExplPackage
4195   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4196 \package_check_loaded_expl:
4197 </package>

```

A token list variable is a $\text{T}_{\text{E}}\text{X}$ macro that holds tokens. By using the $\varepsilon\text{-T}_{\text{E}}\text{X}$ primitive `\unexpanded` inside a $\text{T}_{\text{E}}\text{X}$ `\edef` it is possible to store any tokens, including `#`, in this way.

184.1 Functions

`\tl_new:N` Creating new token list variables is a case of checking for an existing definition and if free doing the definition.

```

4198 \cs_new_protected_nopar:Npn \tl_new:N #1
4199 {
4200   \chk_if_free_cs:N #1
4201   \cs_gset_eq:NN #1 \c_empty_tl
4202 }
4203 \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 4204 \cs_new_protected:Npn \tl_const:Nn #1#2
\tl_const:cn 4205 {
\tl_const:cx 4206   \chk_if_free_cs:N #1
4207   \cs_gset_nopar:Npx #1 { \exp_not:n {#2} }
4208 }
4209 \cs_new_protected:Npn \tl_const:Nx #1#2
4210 {
4211   \chk_if_free_cs:N #1
4212   \cs_gset_nopar:Npx #1 {#2}
4213 }
4214 \cs_generate_variant:Nn \tl_const:Nn { c }
4215 \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 4216 \cs_new_protected_nopar:Npn \tl_clear:N #1
\tl_gclear:N 4217 { \tl_set_eq:NN #1 \c_empty_tl }
\tl_gclear:c 4218 \cs_new_protected_nopar:Npn \tl_gclear:N #1
4219 { \tl_gset_eq:NN #1 \c_empty_tl }
4220 \cs_generate_variant:Nn \tl_clear:N { c }
4221 \cs_generate_variant:Nn \tl_gclear:N { c }

```

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

`\tl_clear_new:N` Clearing a token list variable means setting it to an empty value. Error checking will be sorted out by the parent function.

`\tl_clear_new:c`

`\tl_gclear_new:N` 4222 `\cs_new_protected_nopar:Npn \tl_clear_new:N #1`
4223 `{ \cs_if_exist:NTF #1 { \tl_clear:N #1 } { \tl_new:N #1 } }`

`\tl_gclear_new:c` 4224 `\cs_new_protected_nopar:Npn \tl_gclear_new:N #1`
4225 `{ \cs_if_exist:NTF #1 { \tl_gclear:N #1 } { \tl_new:N #1 } }`
4226 `\cs_generate_variant:Nn \tl_clear_new:N { c }`
4227 `\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` 4228 `\cs_new_eq:NN \tl_set_eq:NN \cs_set_eq:NN`
`\tl_set_eq:cN` 4229 `\cs_new_eq:NN \tl_set_eq:cN \cs_set_eq:cN`
`\tl_set_eq:cc` 4230 `\cs_new_eq:NN \tl_set_eq:Nc \cs_set_eq:Nc`
`\tl_gset_eq:NN` 4231 `\cs_new_eq:NN \tl_set_eq:cc \cs_set_eq:cc`
`\tl_gset_eq:Nc` 4232 `\cs_new_eq:NN \tl_gset_eq:NN \cs_gset_eq:NN`
`\tl_gset_eq:cN` 4233 `\cs_new_eq:NN \tl_gset_eq:cN \cs_gset_eq:cN`
`\tl_gset_eq:cc` 4234 `\cs_new_eq:NN \tl_gset_eq:Nc \cs_gset_eq:Nc`
4235 `\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 ??.)

184.2 Adding to token list variables

`\tl_set:Nn` By using `\exp_not:n` token list variables can contain # tokens, which makes the token list registers provided by T_EX more or less redundant. The `\tl_set:No` version is done “by hand” as it is used quite a lot.

`\tl_set:Nv`

`\tl_set:No` 4236 `\cs_new_protected:Npn \tl_set:Nn #1#2`
`\tl_set:Nf` 4237 `{ \cs_set_nopar:Npx #1 { \exp_not:n {#2} } }`
`\tl_set:Nx` 4238 `\cs_new_protected:Npn \tl_set:No #1#2`
`\tl_set:cn` 4239 `{ \cs_set_nopar:Npx #1 { \exp_not:o {#2} } }`
`\tl_set:Nv` 4240 `\cs_new_protected:Npn \tl_set:Nx #1#2`
`\tl_set:Nv` 4241 `{ \cs_set_nopar:Npx #1 {#2} }`
`\tl_set:co` 4242 `\cs_new_protected:Npn \tl_gset:Nn #1#2`
`\tl_set:cf` 4243 `{ \cs_gset_nopar:Npx #1 { \exp_not:n {#2} } }`
`\tl_set:cx` 4244 `\cs_new_protected:Npn \tl_gset:No #1#2`
4245 `{ \cs_gset_nopar:Npx #1 { \exp_not:o {#2} } }`
`\tl_gset:Nn` 4246 `\cs_new_protected:Npn \tl_gset:Nx #1#2`
`\tl_gset:Nv` 4247 `{ \cs_gset_nopar:Npx #1 {#2} }`
`\tl_gset:Nv` 4248 `\cs_generate_variant:Nn \tl_set:Nn { NV , Nv , Nf }`
`\tl_gset:No` 4249 `\cs_generate_variant:Nn \tl_set:Nx { c }`
`\tl_gset:Nf` 4250 `\cs_generate_variant:Nn \tl_set:Nn { c , co , cV , cv , cf }`
`\tl_gset:Nx` 4251 `\cs_generate_variant:Nn \tl_gset:Nn { NV , Nv , Nf }`
`\tl_gset:cn` 4252 `\cs_generate_variant:Nn \tl_gset:Nx { c }`
`\tl_gset:Nv` 4253 `\cs_generate_variant:Nn \tl_gset:Nn { c , co , cV , cv , cf }`
(End definition for \tl_set:Nn and others. These functions are documented on page ??.)
`\tl_gset:Nv`
`\tl_gset:co`
`\tl_gset:cf`
`\tl_gset:cx`

\tl_put_left:Nn Adding to the left is done directly to gain a little performance.

```

\tl_put_left:NV 4254 \cs_new_protected:Npn \tl_put_left:Nn #1#2
\tl_put_left:No 4255 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_put_left:Nx 4256 \cs_new_protected:Npn \tl_put_left:NV #1#2
\tl_put_left:cn 4257 { \cs_set_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_put_left:cV 4258 \cs_new_protected:Npn \tl_put_left:No #1#2
\tl_put_left:co 4259 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_put_left:cx 4260 \cs_new_protected:Npn \tl_put_left:Nx #1#2
\tl_gput_left:Nn 4261 { \cs_set_nopar:Npx #1 { #2 \exp_not:o #1 } }
\tl_gput_left:NV 4262 \cs_new_protected:Npn \tl_gput_left:Nn #1#2
\tl_gput_left:No 4263 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_gput_left:Nx 4264 \cs_new_protected:Npn \tl_gput_left:NV #1#2
\tl_gput_left:cn 4265 { \cs_gset_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_gput_left:cV 4266 \cs_new_protected:Npn \tl_gput_left:No #1#2
\tl_gput_left:co 4267 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_gput_left:cx 4268 \cs_new_protected:Npn \tl_gput_left:Nx #1#2
4269 { \cs_gset_nopar:Npx #1 { #2 \exp_not:o {#1} } }
4270 \cs_generate_variant:Nn \tl_put_left:Nn { c }
4271 \cs_generate_variant:Nn \tl_put_left:NV { c }
4272 \cs_generate_variant:Nn \tl_put_left:No { c }
4273 \cs_generate_variant:Nn \tl_put_left:Nx { c }
4274 \cs_generate_variant:Nn \tl_gput_left:Nn { c }
4275 \cs_generate_variant:Nn \tl_gput_left:NV { c }
4276 \cs_generate_variant:Nn \tl_gput_left:No { c }
4277 \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 4278 \cs_new_protected:Npn \tl_put_right:Nn #1#2
\tl_put_right:No 4279 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_put_right:Nx 4280 \cs_new_protected:Npn \tl_put_right:NV #1#2
\tl_put_right:cn 4281 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_put_right:cV 4282 \cs_new_protected:Npn \tl_put_right:No #1#2
\tl_put_right:co 4283 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_put_right:cx 4284 \cs_new_protected:Npn \tl_put_right:Nx #1#2
\tl_gput_right:Nn 4285 { \cs_set_nopar:Npx #1 { \exp_not:o #1 #2 } }
\tl_gput_right:NV 4286 \cs_new_protected:Npn \tl_gput_right:Nn #1#2
\tl_gput_right:No 4287 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_gput_right:Nx 4288 \cs_new_protected:Npn \tl_gput_right:NV #1#2
\tl_gput_right:cn 4289 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_gput_right:cV 4290 \cs_new_protected:Npn \tl_gput_right:No #1#2
\tl_gput_right:co 4291 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_gput_right:cx 4292 \cs_new_protected:Npn \tl_gput_right:Nx #1#2
4293 { \cs_gset_nopar:Npx #1 { \exp_not:o {#1} #2 } }
4294 \cs_generate_variant:Nn \tl_put_right:Nn { c }
4295 \cs_generate_variant:Nn \tl_put_right:NV { c }
4296 \cs_generate_variant:Nn \tl_put_right:No { c }
4297 \cs_generate_variant:Nn \tl_put_right:Nx { c }
4298 \cs_generate_variant:Nn \tl_gput_right:Nn { c }

```

```

4299 \cs_generate_variant:Nn \tl_gput_right:NV { c }
4300 \cs_generate_variant:Nn \tl_gput_right:No { c }
4301 \cs_generate_variant:Nn \tl_gput_right:Nx { c }

```

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

184.3 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.

```

4302 \group_begin:
4303   \tex_lccode:D '\A = '\@ \scan_stop:
4304   \tex_lccode:D '\B = '\@ \scan_stop:
4305   \tex_catcode:D '\A = 8 \scan_stop:
4306   \tex_catcode:D '\B = 3 \scan_stop:
4307   \tex_lowercase:D
4308   {
4309     \group_end:
4310     \tl_const:Nn \c_tl_rescan_marker_tl { A B }
4311   }

```

(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.

```

4312 \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 as a file, and without the correct settings a TeX error occurs:

```

\tl_set_rescan:Nno
\tl_set_rescan:cnn
\tl_set_rescan:cno

```

! File ended while scanning definition of ...

`\tl_gset_rescan:Nnn` When expanding a token list this can be handled using `\exp_not:N` but this fails if the token list is not being expanded. So instead a delimited argument is used with an end marker which cannot appear within the token list which is scanned: two @ symbols with different category codes. The rescanned token list cannot contain the end marker, because all @ present in the token list are read with the same category code. As every character with charcode `\newlinechar` is replaced by the `\endlinechar`, and an extra `\endlinechar` is added at the end, we need to set both of those to -1, “unprintable”.

`\tl_gset_rescan:Nno`

`\tl_gset_rescan:cnn`

`\tl_gset_rescan:cno`

`\tl_set_rescan_aux:NNnn`

`\tl_rescan_aux:w`

```

4313 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnn
4314 { \tl_set_rescan_aux:NNnn \tl_set:Nn }
4315 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnn
4316 { \tl_set_rescan_aux:NNnn \tl_gset:Nn }
4317 \cs_new_protected:Npn \tl_set_rescan_aux:NNnn #1#2#3#4
4318 {
4319   \group_begin:
4320   \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl }
4321   \tex_endlinechar:D \c_minus_one
4322   \tex_newlinechar:D \c_minus_one
4323   #3

```



```

4324 \tl_clear:N \l_tl_rescan_tl
4325 \exp_after:wN \tl_rescan_aux:w \etex_scantokens:D {#4}
4326 \exp_args:NNNo \group_end:
4327 #1 #2 \l_tl_rescan_tl
4328 }
4329 \cs_new_nopar:Npx \tl_rescan_aux:w
4330 {
4331 \cs_set_protected:Npn \exp_not:N \tl_rescan_aux:w ##1
4332 \c_tl_rescan_marker_tl
4333 { \tl_set:Nn \exp_not:N \l_tl_rescan_tl {##1} }
4334 }
4335 \tl_rescan_aux:w
4336 \cs_generate_variant:Nn \tl_set_rescan:Nnn { Nno }
4337 \cs_generate_variant:Nn \tl_set_rescan:Nnn { c , cno }
4338 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nno }
4339 \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
\tl_set_rescan:cnx
\tl_gset_rescan:Nnx
\tl_gset_rescan:cnx
\tl_set_rescan_aux:NNnx

With x-type expansion the \everyeof method does apply and the code is simple.

```

4340 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnx
4341 { \tl_set_rescan_aux:NNnx \tl_set:Nn }
4342 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnx
4343 { \tl_set_rescan_aux:NNnx \tl_gset:Nn }
4344 \cs_new_protected_nopar:Npn \tl_set_rescan_aux:NNnx #1#2#3#4
4345 {
4346 \group_begin:
4347 \etex_everyeof:D { \exp_not:N }
4348 \tex_endlinechar:D \c_minus_one
4349 \tex_newlinechar:D \c_minus_one
4350 #3
4351 \tl_set:Nx \l_tl_rescan_tl { \etex_scantokens:D {#4} }
4352 \exp_args:NNNo \group_end:
4353 #1 #2 \l_tl_rescan_tl
4354 }
4355 \cs_generate_variant:Nn \tl_set_rescan:Nnx { c }
4356 \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.

```

4357 \cs_new_protected:Npn \tl_rescan:nn #1#2
4358 {
4359 \group_begin:
4360 \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl }
4361 \tex_endlinechar:D \c_minus_one
4362 \tex_newlinechar:D \c_minus_one
4363 #1
4364 \exp_after:wN \tl_rescan_aux:w \etex_scantokens:D {#2}
4365 \exp_args:No \group_end:

```

```

4366 \l_tl_rescan_tl
4367 }

```

(End definition for `\tl_rescan:nn`. This function is documented on page 85.)

184.4 Reassigning token list character codes

`\tl_to_lowercase:n` Just some names for a few primitives.
`\tl_to_uppercase:n`

```

4368 \cs_new_eq:NN \tl_to_lowercase:n \tex_lowercase:D
4369 \cs_new_eq:NN \tl_to_uppercase:n \tex_uppercase:D

```

(End definition for `\tl_to_lowercase:n`. This function is documented on page 85.)

184.5 Modifying token list variables

`\l_tl_replace_tl` A scratch variable for doing token replacement.

```

4370 \tl_new:N \l_tl_replace_tl

```

(End definition for `\l_tl_replace_tl`. This function is documented on page ??.)

`\tl_replace_all:Nnn` All of the replace functions are based on `\tl_replace_aux:NNNnn`, whose arguments are:
`\tl_replace_all:cnn` $\langle function \rangle$, $\langle tl_g \rangle \text{set:Nx}$, $\langle tl_var \rangle$, $\langle search\ tokens \rangle$, $\langle replacement\ tokens \rangle$.
`\tl_greplace_all:Nnn`
`\tl_greplace_all:cnn`
`\tl_replace_once:Nnn`
`\tl_replace_once:cnn`
`\tl_greplace_once:Nnn`
`\tl_greplace_once:cnn`
`\tl_replace_aux:NNNnn`
`\tl_replace_aux:ii:w`
`\tl_replace_all_aux:`
`\tl_replace_once_aux:`
`\tl_replace_once_aux_end:w`

```

4371 \cs_new_protected_nopar:Npn \tl_replace_once:Nnn
4372 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_set:Nx }
4373 \cs_new_protected_nopar:Npn \tl_greplace_once:Nnn
4374 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_gset:Nx }
4375 \cs_new_protected_nopar:Npn \tl_replace_all:Nnn
4376 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_set:Nx }
4377 \cs_new_protected_nopar:Npn \tl_greplace_all:Nnn
4378 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_gset:Nx }
4379 \cs_generate_variant:Nn \tl_replace_once:Nnn { c }
4380 \cs_generate_variant:Nn \tl_greplace_once:Nnn { c }
4381 \cs_generate_variant:Nn \tl_replace_all:Nnn { c }
4382 \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.

```

4383 \cs_new_protected:Npn \tl_replace_aux:NNNnn #1#2#3#4#5

```

```

4384 {
4385   \tl_if_empty:nTF {#4}
4386   {
4387     \msg_kernel_error:nxx { tl } { empty-search-pattern }
4388     { \tl_to_str:n {#5} }
4389   }
4390   {
4391     \cs_set:Npx \tl_tmp:w ##1##2 #4
4392     {
4393       ##2
4394       \exp_not:N \q_mark
4395       \exp_not:N \use_none_delimit_by_q_stop:w
4396       \exp_not:n { \exp_not:n {#5} }
4397       ##1
4398     }
4399     #2 #3
4400     {
4401       \exp_after:wN #1
4402       #3 \q_mark #4 \q_stop
4403     }
4404   }
4405 }
4406 \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.

```

4407 \cs_new:Npn \tl_replace_all_aux:
4408 {
4409   \exp_after:wN \tl_replace_aux_ii:w
4410   \tl_tmp:w \tl_replace_all_aux: \prg_do_nothing:
4411 }
4412 \cs_new_nopar:Npn \tl_replace_once_aux:
4413 {
4414   \exp_after:wN \tl_replace_aux_ii:w
4415   \tl_tmp:w { \tl_replace_once_aux_end:w \prg_do_nothing: } \prg_do_nothing:
4416 }
4417 \cs_new:Npn \tl_replace_once_aux_end:w #1 \q_mark #2 \q_stop
4418 { \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
\tl_remove_once:cn
\tl_gremove_once:Nn
\tl_gremove_once:cn

```

Removal is just a special case of replacement.

```

4419 \cs_new_protected_nopar:Npn \tl_remove_once:Nn #1#2
4420 { \tl_replace_once:Nnn #1 {#2} { } }

```

```

4421 \cs_new_protected_nopar:Npn \tl_gremove_once:Nn #1#2
4422 { \tl_greplace_once:Nnn #1 {#2} { } }
4423 \cs_generate_variant:Nn \tl_remove_once:Nn { c }
4424 \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
4425 \cs_new_protected_nopar:Npn \tl_remove_all:Nn #1#2
\tl_gremove_all:Nn
4426 { \tl_replace_all:Nnn #1 {#2} { } }
\tl_gremove_all:cn
4427 \cs_new_protected_nopar:Npn \tl_gremove_all:Nn #1#2
4428 { \tl_greplace_all:Nnn #1 {#2} { } }
4429 \cs_generate_variant:Nn \tl_remove_all:Nn { c }
4430 \cs_generate_variant:Nn \tl_gremove_all:Nn { c }

```

184.6 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.

```

4431 \prg_new_conditional:Npnn \tl_if_blank:n #1 { p , T , F , TF }
4432 { \tl_if_empty_return:o { \use_none:n #1 ? } }
4433 \cs_generate_variant:Nn \tl_if_blank_p:n { V }
4434 \cs_generate_variant:Nn \tl_if_blank:nT { V }
4435 \cs_generate_variant:Nn \tl_if_blank:nF { V }
4436 \cs_generate_variant:Nn \tl_if_blank:nTF { V }
4437 \cs_generate_variant:Nn \tl_if_blank_p:n { o }
4438 \cs_generate_variant:Nn \tl_if_blank:nT { o }
4439 \cs_generate_variant:Nn \tl_if_blank:nF { o }
4440 \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`

```

4441 \prg_set_conditional:Npnn \tl_if_empty:N #1 { p , T , F , TF }
4442 {
4443   \if_meaning:w #1 \c_empty_tl
4444   \prg_return_true:
4445   \else:
4446   \prg_return_false:
4447   \fi:
4448 }

```

```

4449 \cs_generate_variant:Nn \tl_if_empty_p:N { c }
4450 \cs_generate_variant:Nn \tl_if_empty:NT { c }
4451 \cs_generate_variant:Nn \tl_if_empty:NF { c }
4452 \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.

```

4453 \prg_new_conditional:Npnn \tl_if_empty:n #1 { p , TF , T , F }
4454 {
4455   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil \tl_to_str:n {#1} \q_nil
4456   \prg_return_true:
4457   \else:
4458   \prg_return_false:
4459   \fi:
4460 }
4461 \cs_generate_variant:Nn \tl_if_empty_p:n { V }
4462 \cs_generate_variant:Nn \tl_if_empty:nTF { V }
4463 \cs_generate_variant:Nn \tl_if_empty:nT { V }
4464 \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.

```

4465 \cs_new:Npn \tl_if_empty_return:o #1
4466 {
4467   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil
4468   \tl_to_str:n \exp_after:wN {#1} \q_nil
4469   \prg_return_true:
4470   \else:
4471   \prg_return_false:
4472   \fi:
4473 }
4474 \prg_new_conditional:Npnn \tl_if_empty:o #1 { p , TF , T , F }
4475 { \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.

```

4476 \prg_new_conditional:Npnn \tl_if_eq:NN #1#2 { p , T , F , TF }
4477 {
4478   \if_meaning:w #1 #2
4479   \prg_return_true:
4480   \else:
4481     \prg_return_false:
4482   \fi:
4483 }
4484 \cs_generate_variant:Nn \tl_if_eq:p:NN { Nc , c , cc }
4485 \cs_generate_variant:Nn \tl_if_eq:NNTF { Nc , c , cc }
4486 \cs_generate_variant:Nn \tl_if_eq:NNT { Nc , c , cc }
4487 \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.

```

4488 \prg_new_protected_conditional:Npnn \tl_if_eq:nn #1#2 { T , F , TF }
4489 {
4490   \group_begin:
4491     \tl_set:Nn \l_tl_tmpa_tl {#1}
4492     \tl_set:Nn \l_tl_tmpb_tl {#2}
4493     \if_meaning:w \l_tl_tmpa_tl \l_tl_tmpb_tl
4494     \group_end:
4495     \prg_return_true:
4496   \else:
4497     \group_end:
4498     \prg_return_false:
4499   \fi:
4500 }
4501 \tl_new:N \l_tl_tmpa_tl
4502 \tl_new:N \l_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 variable and pass it to `\tl_if_in:nn(TF)`.

```

4503 \cs_new_protected_nopar:Npn \tl_if_in:NnT { \exp_args:No \tl_if_in:nnT }
4504 \cs_new_protected_nopar:Npn \tl_if_in:NnF { \exp_args:No \tl_if_in:nnF }
4505 \cs_new_protected_nopar:Npn \tl_if_in:NnTF { \exp_args:No \tl_if_in:nnTF }
4506 \cs_generate_variant:Nn \tl_if_in:NnT { c }
4507 \cs_generate_variant:Nn \tl_if_in:NnF { c }
4508 \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` removes tokens until the first occurrence of `#2`. If this does not appear in `#1`, then the final `#2` is removed, leaving an empty token list. Otherwise some tokens remain, and the 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.

```

4509 \prg_new_protected_conditional:Npnn \tl_if_in:nn #1#2 { T , F , TF }
4510 {
4511   \cs_set:Npn \tl_tmp:w ##1 #2 { }
4512   \tl_if_empty:oTF { \tl_tmp:w #1 {} {} } #2 {
4513     { \prg_return_false: } { \prg_return_true: }
4514   }
4515   \cs_generate_variant:Nn \tl_if_in:nnT { V , o , no }
4516   \cs_generate_variant:Nn \tl_if_in:nnF { V , o , no }
4517   \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 ??.)

184.7 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:Nn #1#2
\tl_map_function:NN \tl_map_function:NN #1#2
\tl_map_function:cN \tl_map_function:cN #1#2
\tl_map_function_aux:NN
4518 \cs_new:Npn \tl_map_function:nN #1#2
4519 { \tl_map_function_aux:Nn #2 #1 \q_recursion_tail \q_recursion_stop }
4520 \cs_new_nopar:Npn \tl_map_function:NN #1#2
4521 {
4522   \exp_after:wN \tl_map_function_aux:Nn
4523   \exp_after:wN #2 #1 \q_recursion_tail \q_recursion_stop
4524 }
4525 \cs_new:Npn \tl_map_function_aux:Nn #1#2
4526 {
4527   \quark_if_recursion_tail_stop:n {#2}
4528   #1 {#2} \tl_map_function_aux:Nn #1
4529 }
4530 \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 \g_tl_inline_level_int
\tl_map_inline:cn \g_tl_inline_level_int
\tl_map_inline_aux:n
4531 \cs_new_protected:Npn \tl_map_inline:nn #1#2
4532 {
4533   \int_gincr:N \g_tl_inline_level_int
4534   \cs_gset:cpn { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4535   ##1 {#2}
4536   \exp_args:Nc \tl_map_function_aux:Nn
4537   { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4538   #1 \q_recursion_tail \q_recursion_stop

```

```

4539 \int_gdecr:N \g_tl_inline_level_int
4540 }
4541 \cs_new_protected:Npn \tl_map_inline:Nn #1#2
4542 {
4543 \int_gincr:N \g_tl_inline_level_int
4544 \cs_gset:cpn { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4545 ##1 {#2}
4546 \exp_last_unbraced:NcV \tl_map_function_aux:Nn
4547 { tl_map_inline_ \int_use:N \g_tl_inline_level_int :n }
4548 #1 \q_recursion_tail\q_recursion_stop
4549 \int_gdecr:N \g_tl_inline_level_int
4550 }
4551 \cs_generate_variant:Nn \tl_map_inline:Nn { c }

```

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

\tl_map_variable:nNn \tl_map_variable:NnN <token list> <temp> <action> assigns <temp> to each element and executes <action>.

\tl_map_variable:cNn \tl_map_variable_aux:NnN

```

4552 \cs_new_protected:Npn \tl_map_variable:nNn #1#2#3
4553 { \tl_map_variable_aux:Nnn #2 {#3} #1 \q_recursion_tail \q_recursion_stop }
4554 \cs_new_protected_nopar:Npn \tl_map_variable:NnN
4555 { \exp_args:No \tl_map_variable:nNn }
4556 \cs_new_protected:Npn \tl_map_variable_aux:Nnn #1#2#3
4557 {
4558 \tl_set:Nn #1 {#3}
4559 \quark_if_recursion_tail_stop:N #1
4560 #2 \tl_map_variable_aux:Nnn #1 {#2}
4561 }
4562 \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.

```

4563 \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 ??.)

184.8 Using token lists

\tl_to_str:n Another name for a primitive.

```

4564 \cs_new_eq:NN \tl_to_str:n \etex_detokenize:D

```

(End definition for \tl_to_str:n. This function is documented on page 88.)

\tl_to_str:N These functions return the replacement text of a token list as a string.

\tl_to_str:c

```

4565 \cs_new_nopar:Npn \tl_to_str:N #1 { \etex_detokenize:D \exp_after:wN {#1} }
4566 \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 for ones equal to `\scan_stop:` so instead a test is made and if there is an issue an error is forced.

```

4567 \cs_new_eq:NN \tl_use:N \prg_do_nothing:
4568 \cs_new_nopar:Npn \tl_use:c #1
4569 {
4570   \if_cs_exist:w #1 \cs_end:
4571   \cs:w #1 \exp_after:wN \cs_end:
4572   \else:
4573     \msg_expandable_error:n { Undefined-variable-name~'#1'! }
4574   \fi:
4575 }

```

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

184.9 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 4576 \cs_new:Npn \tl_length:n #1
\tl_length:V 4577 {
\tl_length:o 4578   \int_eval:n
\tl_length:N 4579   { 0 \tl_map_function:nN {#1} \tl_length_aux:n }
\tl_length:c 4580 }
\tl_length_aux:n 4581 \cs_new_nopar:Npn \tl_length:N #1
4582 {
4583   \int_eval:n
4584   { 0 \tl_map_function:NN #1 \tl_length_aux:n }
4585 }
4586 \cs_new:Npn \tl_length_aux:n #1 { + 1 }
4587 \cs_generate_variant:Nn \tl_length:n { V , o }
4588 \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`.

```

4589 \cs_new:Npn \tl_reverse_items:n #1
4590 { \tl_reverse_items_aux:nw #1 \q_recursion_tail \q_recursion_stop }
4591 \cs_new:Npn \tl_reverse_items_aux:nw #1 #2 \q_recursion_stop
4592 {
4593   \quark_if_recursion_tail_stop_do:nn {#1} { \use_none:n }
4594   \tl_reverse_items_aux:nw #2 \q_recursion_stop
4595   {#1}
4596 }

```

(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.

```

4597 \cs_set:Npn \tl_tmp:w #1
4598 {
4599   \cs_new:Npn \tl_trim_spaces:n ##1
4600   {
4601     \etex_unexpanded:D
4602     \tl_trim_spaces_aux_i:w
4603     \q_mark
4604     ##1
4605     \q_nil
4606     \q_mark #1 { }
4607     \q_mark \tl_trim_spaces_aux_ii:w
4608     \tl_trim_spaces_aux_iii:w
4609     #1 \q_nil
4610     \tl_trim_spaces_aux_iv:w
4611     \q_stop
4612   }
4613 \cs_new:Npn \tl_trim_spaces_aux_i:w ##1 \q_mark #1 ##2 \q_mark ##3
4614 {
4615   ##3
4616   \tl_trim_spaces_aux_i:w
4617   \q_mark
4618   ##2
4619   \q_mark #1 {##1}
4620 }
4621 \cs_new:Npn \tl_trim_spaces_aux_ii:w ##1 \q_mark \q_mark ##2
4622 {
4623   \tl_trim_spaces_aux_iii:w
4624   ##2
4625 }
4626 \cs_new:Npn \tl_trim_spaces_aux_iii:w ##1 #1 \q_nil ##2
4627 {
4628   ##2
4629   ##1 \q_nil
4630   \tl_trim_spaces_aux_iii:w
4631 }
4632 \cs_new:Npn \tl_trim_spaces_aux_iv:w ##1 \q_nil ##2 \q_stop

```

```

4633     { \exp_after:wN { \use_none:n ##1 } }
4634   }
4635   \tl_tmp:w { ~ }
4636   \cs_new_protected:Npn \tl_trim_spaces:N #1
4637     { \tl_set:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4638   \cs_new_protected:Npn \tl_gtrim_spaces:N #1
4639     { \tl_gset:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4640   \cs_generate_variant:Nn \tl_trim_spaces:N { c }
4641   \cs_generate_variant:Nn \tl_gtrim_spaces:N { c }

```

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

184.10 The first token from a token list

`\tl_head:n` These functions pick up either the head or the tail of a list. The empty brace groups in `\tl_head:n` and `\tl_tail:n` ensure that a blank argument gives an empty result.

```

\tl_head:n 4642 \cs_new:Npn \tl_head:w #1#2 \q_stop {#1}
\tl_head:v 4643 \cs_new:Npn \tl_tail:w #1#2 \q_stop {#2}
\tl_head:f 4644 \cs_new:Npn \tl_head:n #1
\tl_head:w 4645   { \tl_head:w #1 { } \q_stop }
\tl_tail:n 4646 \cs_new:Npn \tl_tail:n #1
\tl_tail:v 4647   { \tl_tail_aux:w #1 \q_mark { } \q_mark \q_stop }
\tl_tail:f 4648 \cs_new:Npn \tl_tail_aux:w #1 #2 \q_mark #3 \q_stop { #2 }
\tl_tail:w 4649 \cs_generate_variant:Nn \tl_head:n { V , v , f }
           4650 \cs_generate_variant:Nn \tl_tail:n { V , v , f }

```

(End definition for `\tl_head:n` and others. These functions are documented on page 91.)

`\str_head:n` After `\tl_to_str:n`, we have a list of character tokens, all with category code 12, except
`\str_tail:n` the space, which has category code 10. Directly using `\tl_head:w` would thus lose leading
`\str_head_aux:w` spaces. Instead, we take an argument delimited by an explicit space, and then only use `\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.

```

4651 \cs_new:Npn \str_head:n #1
4652   {
4653     \exp_after:wN \str_head_aux:w
4654     \tl_to_str:n {#1}
4655     { { } } ~ \q_stop
4656   }
4657 \cs_new_nopar:Npn \str_head_aux:w #1 ~ %
4658   { \tl_head:w #1 { ~ } }
4659 \cs_new:Npn \str_tail:n #1

```

```

4660 {
4661   \exp_after:wN \str_tail_aux:w
4662   \reverse_if:N \if_charcode:w
4663   \scan_stop: \tl_to_str:n {#1} X X \q_stop
4664 }
4665 \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
\tl_if_head_eq_charcode:fN
\tl_if_head_eq_catcode:nN Forgetting temporarily about this issue we would use the following test in \tl_if_head_eq_charcode:nN. Here, an empty #1 argument yields \q_nil, otherwise the first token of the token list.

```

\tl_if_charcode:w
  \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop
  \exp_not:N #2

```

The special cases are detected using \tl_if_head_N_type:n (the extra ? takes care of empty arguments). In those cases, the first token is a character, and since we only care about its character code, we can use \str_head:n to access it (this works even if it is a space character).

```

4666 \prg_new_conditional:Npnn \tl_if_head_eq_charcode:nN #1#2 { p , T , F , TF }
4667 {
4668   \if_charcode:w
4669     \exp_not:N #2
4670     \tl_if_head_N_type:nTF { #1 ? }
4671     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4672     { \str_head:n {#1} }
4673   \prg_return_true:
4674   \else:
4675     \prg_return_false:
4676   \fi:
4677 }
4678 \cs_generate_variant:Nn \tl_if_head_eq_charcode_p:nN { f }
4679 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNTF { f }
4680 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNT { f }
4681 \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.

```

4682 \prg_new_conditional:Npnn \tl_if_head_eq_catcode:nN #1 #2 { p , T , F , TF }
4683 {
4684   \if_catcode:w
4685     \exp_not:N #2
4686     \tl_if_head_N_type:nTF { #1 ? }

```

```

4687         { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4688         {
4689             \tl_if_head_group:nTF {#1}
4690             { \c_group_begin_token }
4691             { \c_space_token }
4692         }
4693         \prg_return_true:
4694     \else:
4695         \prg_return_false:
4696     \fi:
4697 }

```

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.

```

4698 \prg_new_conditional:Npnn \tl_if_head_eq_meaning:nN #1#2 { p , T , F , TF }
4699 {
4700     \tl_if_head_N_type:nTF { #1 ? }
4701     { \tl_if_head_eq_meaning_aux_normal:nN }
4702     { \tl_if_head_eq_meaning_aux_special:nN }
4703     {#1} #2
4704 }
4705 \cs_new:Npn \tl_if_head_eq_meaning_aux_normal:nN #1 #2
4706 {
4707     \exp_after:wN \if_meaning:w \tl_head:w #1 \q_nil \q_stop #2
4708     \prg_return_true:
4709     \else:
4710         \prg_return_false:
4711     \fi:
4712 }
4713 \cs_new:Npn \tl_if_head_eq_meaning_aux_special:nN #1 #2
4714 {
4715     \if_charcode:w \str_head:n {#1} \exp_not:N #2
4716     \exp_after:wN \use:n
4717     \else:
4718         \prg_return_false:
4719         \exp_after:wN \use_none:n
4720     \fi:
4721     {
4722         \if_catcode:w \exp_not:N #2
4723         \tl_if_head_group:nTF {#1}
4724         { \c_group_begin_token }
4725         { \c_space_token }
4726         \prg_return_true:
4727         \else:
4728             \prg_return_false:
4729         \fi:
4730     }

```

```

4731 }
      (End definition for \tl_if_head_eq_meaning:nN. This function is documented on page 91.)

```

`\tl_if_head_N_type:n` The first token of a token list can be either an N-type argument, a begin-group token (catcode 1), or an explicit space token (catcode 10 and charcode 32). These two cases are characterized by the fact that `\use:n` removes some tokens from `#1`, hence changing its string representation (no token can have an empty string representation). The extra brace group covers the case of an empty argument, whose head is not “normal”.

```

4732 \prg_new_conditional:Npnn \tl_if_head_N_type:n #1 { p , T , F , TF }
4733 { \str_if_eq_return:on { \use:n #1 { } } { #1 { } } }
      (End definition for \tl_if_head_N_type:n. This function is documented on page 92.)

```

`\tl_if_head_group:n` Pass the first token of `#1` through `\token_to_str:N`, then check for the brace balance. The extra `?` caters for an empty argument.⁶

```

4734 \prg_new_conditional:Npnn \tl_if_head_group:n #1 { p , T , F , TF }
4735 {
4736   \if_predicate:w
4737     \exp_after:wN \use_none:n
4738     \exp_after:wN {
4739       \exp_after:wN {
4740         \token_to_str:N #1 ?
4741       }
4742       \c_false_bool
4743     }
4744     \c_true_bool
4745     \prg_return_false:
4746   \else:
4747     \prg_return_true:
4748   \fi:
4749 }
      (End definition for \tl_if_head_group:n. This function is documented on page 92.)

```

`\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.

```

4750 \prg_new_conditional:Npnn \tl_if_head_space:n #1 { p , T , F , TF }
4751 {
4752   \if_int_compare:w
4753     \pdfTeX_strcmp:D
4754     { }
4755     { \tl_if_head_space_aux:w \prg_do_nothing: #1 ? ~ }
4756     = \c_zero
4757     \prg_return_true:
4758   \else:

```

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.

```

4759     \prg_return_false:
4760     \fi:
4761   }
4762   \cs_new:Npn \tl_if_head_space_aux:w #1 ~ %
4763   {
4764     \exp_not:o {#1}
4765     \if_false: { \fi: }
4766     \exp_after:wN \use_none:n \exp_after:wN { \if_false: } \fi:
4767   }

```

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

184.11 Viewing token lists

`\tl_show:N` Showing token list variables is done directly: at the moment do not worry if they are defined.

```

4768 \cs_new_protected:Npn \tl_show:N #1 { \cs_show:N #1 }
4769 \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.

```

4770 \cs_new_eq:NN \tl_show:n \etex_showtokens:D

```

(End definition for `\tl_show:n`. This function is documented on page 93.)

184.12 Constant token lists

`\c_job_name_tl` Inherited from the L^AT_EX3 name for the primitive: this needs to actually contain the text of the job name rather than the name of the primitive, of course. Lua_TE_X does not quote file names containing spaces, whereas pdf_TE_X and X_T_L_AT_EX do. So there may be a correction to make in the Lua_TE_X case.

```

4771 ⟨*initex⟩
4772 \tex_everyjob:D \exp_after:wN
4773 {
4774   \tex_the:D \tex_everyjob:D
4775   \luatex_if_engine:T
4776   {
4777     \lua_now:x
4778     { dofile ( assert ( kpse.find_file ("lualatexquotejobname.lua" ) ) ) }
4779   }
4780 }
4781 ⟨/initex⟩
4782 \tl_const:Nx \c_job_name_tl { \tex_jobname:D }

```

(End definition for `\c_job_name_tl`. This function is documented on page 93.)

`\c_empty_tl` Never full.

```

4783 \tl_const:Nn \c_empty_tl { }

```

(End definition for `\c_empty_tl`. This function is documented on page 93.)

`\c_space_tl` A space as a token list (as opposed to as a character).

```
4784 \tl_const:Nn \c_space_tl { ~ }
```

(End definition for `\c_space_tl`. This function is documented on page 93.)

184.13 Scratch token lists

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

```
4785 \tl_new:N \g_tmpa_tl
```

```
4786 \tl_new:N \g_tmpb_tl
```

(End definition for `\g_tmpa_tl` and `\g_tmpb_tl`. These functions are documented on page 93.)

`\l_tmpa_tl` These are local temporary token list variables. Be sure not to assume that the value you put into them will survive for long—see discussion above.

```
4787 \tl_new:N \l_tmpa_tl
```

```
4788 \tl_new:N \l_tmpb_tl
```

(End definition for `\l_tmpa_tl` and `\l_tmpb_tl`. These functions are documented on page 93.)

184.14 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.

```
4789 \cs_new:Npn \str_if_eq_return:on #1 #2
```

```
4790 {
```

```
4791   \if_int_compare:w
```

```
4792     \pdfTeX_strcmp:D { \exp_not:o {#1} } { \exp_not:n {#2} }
```

```
4793     = \c_zero
```

```
4794     \prg_return_true:
```

```
4795   \else:
```

```
4796     \prg_return_false:
```

```
4797   \fi:
```

```
4798 }
```

(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`.

```
4799 \cs_new:Npn \tl_if_single_p:N { \exp_args:No \tl_if_single_p:n }
```

```
4800 \cs_new:Npn \tl_if_single:NT { \exp_args:No \tl_if_single:nT }
```

```
4801 \cs_new:Npn \tl_if_single:NF { \exp_args:No \tl_if_single:nF }
```

```
4802 \cs_new:Npn \tl_if_single:NTF { \exp_args:No \tl_if_single:nTF }
```

(End definition for `\tl_if_single:N`. This function is documented on page 86.)

`\tl_if_single:n` A token list has exactly one item if it is either a single token surrounded by optional explicit spaces, or a single brace group surrounded by optional explicit spaces. The naive version of this test would do `\use_none:n #1`, and test if the result is empty. However, this will fail when the token list is empty. Furthermore, it does not allow optional trailing spaces.

```
4803 \prg_new_conditional:Npnn \tl_if_single:n #1 { p , T , F , TF }
4804 { \str_if_eq_return:on { \use_none:nn #1 ?? } {?} }
      (End definition for \tl_if_single:n. This function is documented on page 87.)
```

`\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.

```
4805 \prg_new_conditional:Npnn \tl_if_single_token:n #1 { p , T , F , TF }
4806 {
4807   \tl_if_head_N_type:nTF {#1}
4808   { \str_if_eq_return:on { \use_none:n #1 } { } }
4809   { \str_if_eq_return:on { ~ } { #1 } }
4810 }
      (End definition for \tl_if_single_token:n. This function is documented on page 87.)
```

`\q_tl_act_mark` The `\tl_act` functions may be applied to any token list. Hence, we use two private quarks, to allow any token, even quarks, in the token list. Only `\q_tl_act_mark` and `\q_tl_act_stop` may not appear in the token lists manipulated by `\tl_act` functions. The quarks are effectively defined in `l3quark`.

(End definition for `\q_tl_act_mark` and `\q_tl_act_stop`. These functions are documented on page 94.)

`\tl_act:NNNnn` To help control the expansion, `\tl_act:NNNnn` starts with `\romannumeral` and ends by producing `\c_zero` once the result has been obtained. Then loop over tokens, groups, and spaces in #5. The marker `\q_tl_act_mark` is used both to avoid losing outer braces and to detect the end of the token list more easily. The result is stored as an argument for the dummy function `\tl_act_result:n`.

```
\tl_act_aux:NNNnn
\tl_act_output:n
\tl_act_reverse_output:n
\tl_act_group_recurse:Nnn
\tl_act_loop:w
\tl_act_normal:NwnNNN
\tl_act_group:nwnNNN
\tl_act_space:wwnNNN
\tl_act_end:w
4811 \cs_new:Npn \tl_act:NNNnn { \tex_romannumeral:D \tl_act_aux:NNNnn }
4812 \cs_new:Npn \tl_act_aux:NNNnn #1 #2 #3 #4 #5
4813 {
4814   \tl_act_loop:w #5 \q_tl_act_mark \q_tl_act_stop
4815   {#4} #1 #2 #3
4816   \tl_act_result:n { }
4817 }
```

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.

```

4818 \cs_new:Npn \tl_act_loop:w #1 \q_tl_act_stop
4819 {
4820   \tl_if_head_N_type:nTF {#1}
4821   { \tl_act_normal:NwnNNN }
4822   {
4823     \tl_if_head_group:nTF {#1}
4824     { \tl_act_group:nwnNNN }
4825     { \tl_act_space:wwnNNN }
4826   }
4827   #1 \q_tl_act_stop
4828 }
4829 \cs_new:Npn \tl_act_normal:NwnNNN #1 #2 \q_tl_act_stop #3#4
4830 {
4831   \if_meaning:w \q_tl_act_mark #1
4832   \exp_after:wN \tl_act_end:wn
4833   \fi:
4834   #4 {#3} #1
4835   \tl_act_loop:w #2 \q_tl_act_stop
4836   {#3} #4
4837 }
4838 \cs_new:Npn \tl_act_end:wn #1 \tl_act_result:n #2 { \c_zero #2 }
4839 \cs_new:Npn \tl_act_group:nwnNNN #1 #2 \q_tl_act_stop #3#4#5
4840 {
4841   #5 {#3} {#1}
4842   \tl_act_loop:w #2 \q_tl_act_stop
4843   {#3} #4 #5
4844 }
4845 \exp_last_unbraced:NNo
4846 \cs_new:Npn \tl_act_space:wwnNNN \c_space_tl #1 \q_tl_act_stop #2#3#4#5
4847 {
4848   #5 {#2}
4849   \tl_act_loop:w #1 \q_tl_act_stop
4850   {#2} #3 #4 #5
4851 }

```

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.

```

4852 \cs_new:Npn \tl_act_output:n #1 #2 \tl_act_result:n #3
4853 { #2 \tl_act_result:n { #3 #1 } }
4854 \cs_new:Npn \tl_act_reverse_output:n #1 #2 \tl_act_result:n #3
4855 { #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.

```

4856 \cs_new:Npn \tl_act_group_recurse:Nnn #1#2#3
4857 {
4858   \exp_args:Nf #1
4859   { \exp_after:wN \exp_after:wN \exp_after:wN { #2 {#3} } }
4860 }

```

(End definition for `\tl_act:NNNnn` and `\tl_act_aux:NNNnn`. These functions are documented on page ??.)

`\tl_reverse_tokens:n` The goal is to reverse a token list. This is done by feeding `\tl_act_aux:NNNnn` three functions, an empty fourth argument (we don't use it for `\tl_act_reverse_tokens:n`), and as a fifth argument the token list to be reversed. Spaces and normal tokens are output to the left of the current output. For groups, we must recursively apply `\tl_act_reverse_tokens:n` to the group, and output, still on the left. Note that in all three cases, we throw one argument away: this *parameter* is where for instance the upper/lowercasing action stores the information of whether it is uppercasing or lowercasing.

```

4861 \cs_new:Npn \tl_reverse_tokens:n
4862 {
4863   \tex_romannumeral:D
4864   \tl_act_aux:NNNnn
4865   \tl_act_reverse_normal:nN
4866   \tl_act_reverse_group:nn
4867   \tl_act_reverse_space:n
4868   { }
4869 }
4870 \cs_new:Npn \tl_act_reverse_space:n #1
4871 { \tl_act_reverse_output:n {~} }
4872 \cs_new:Npn \tl_act_reverse_normal:nN #1 #2
4873 { \tl_act_reverse_output:n {#2} }
4874 \cs_new:Npn \tl_act_reverse_group:nn #1
4875 {
4876   \tl_act_group_recurse:Nnn
4877   \tl_act_reverse_output:n
4878   { \tl_reverse_tokens:n }
4879 }

```

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

`\tl_reverse:n` The goal here is to reverse without losing spaces nor braces. The only difference with
`\tl_reverse:o` `\tl_reverse_tokens:n` is that we now simply output groups without entering them.
`\tl_reverse:V`
`\tl_reverse_group_preserve:nn`

```

4880 \cs_new:Npn \tl_reverse:n
4881 {
4882   \tex_romannumeral:D
4883   \tl_act_aux:NNNnn
4884   \tl_act_reverse_normal:nN
4885   \tl_act_reverse_group_preserve:nn
4886   \tl_act_reverse_space:n
4887   { }
4888 }
4889 \cs_new:Npn \tl_act_reverse_group_preserve:nn #1 #2
4890 { \tl_act_reverse_output:n { {#2} } }
4891 \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` primitive.
`\tl_reverse:c`

```
4892 \cs_new_protected_nopar:Npn \tl_reverse:N #1
4893 { \tl_set:No #1 { \etex_unexpanded:D \tl_reverse:o { #1 { } } } }
4894 \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 the *left*, into the integer expression, and the sum is ended by the `\c_zero` inserted by `\tl_act_end:wn`. Somewhat a hack.
`\tl_act_length_normal:nN`
`\tl_act_length_group:nn`
`\tl_act_length_space:n`

```
4895 \cs_new:Npn \tl_length_tokens:n #1
4896 {
4897   \int_eval:n
4898   {
4899     \tl_act_aux:NNNnn
4900     \tl_act_length_normal:nN
4901     \tl_act_length_group:nn
4902     \tl_act_length_space:n
4903     { }
4904     {#1}
4905   }
4906 }
4907 \cs_new:Npn \tl_act_length_normal:nN #1 #2 { 1 + }
4908 \cs_new:Npn \tl_act_length_space:n #1 { 1 + }
4909 \cs_new:Npn \tl_act_length_group:nn #1 #2
4910 { 2 + \tl_length_tokens:n {#2} + }
(End definition for \tl_length_tokens:n. This function is documented on page ??.)
```

`\c_tl_act_uppercase_tl` These constants contain the correspondance between lowercase and uppercase letters, in the form `aAbBcC...` and `AaBbCc...` respectively.
`\c_tl_act_lowercase_tl`

```
4911 \tl_const:Nn \c_tl_act_uppercase_tl
4912 {
4913   aA bB cC dD eE fF gG hH iI jJ kK lL mM
4914   nN oO pP qQ rR sS tT uU vV wW xX yY zZ
4915 }
4916 \tl_const:Nn \c_tl_act_lowercase_tl
4917 {
4918   Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm
4919   Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz
4920 }
(End definition for \c_tl_act_uppercase_tl and \c_tl_act_lowercase_tl. These functions are documented on page ??.)
```

`\tl_expandable_uppercase:n` The only difference between uppercasing and lowercasing is the table of correspondance that is used. As for other token list actions, we feed `\tl_act_aux:NNNnn` three functions, and this time, we use the *parameters* argument to carry which case-changing we are applying. A space is simply output. A normal token is compared to each letter in
`\tl_act_case_normal:nN`
`\tl_act_case_group:nn`
`\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.

```

4921 \cs_new:Npn \tl_expandable_uppercase:n
4922 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_uppercase_tl } }
4923 \cs_new:Npn \tl_expandable_lowercase:n
4924 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_lowercase_tl } }
4925 \cs_new:Npn \tl_act_case_aux:nn
4926 {
4927   \tl_act_aux:NNNnn
4928   \tl_act_case_normal:nN
4929   \tl_act_case_group:nn
4930   \tl_act_case_space:n
4931 }
4932 \cs_new:Npn \tl_act_case_space:n #1 { \tl_act_output:n {~} }
4933 \cs_new:Npn \tl_act_case_normal:nN #1 #2
4934 {
4935   \exp_args:Nf \tl_act_output:n
4936   {
4937     \exp_args:NNo \prg_case_str:nnn #2 {#1}
4938     { \exp_stop_f: #2 }
4939   }
4940 }
4941 \cs_new:Npn \tl_act_case_group:nn #1 #2
4942 {
4943   \exp_after:wN \tl_act_output:n \exp_after:wN
4944   { \exp_after:wN { \tex_romannumeral:D \tl_act_case_aux:nn {#1} {#2} } }
4945 }

```

(End definition for `\tl_expandable_uppercase:n` and `\tl_expandable_lowercase:n`. These functions are documented on page ??.)

184.15 Deprecated functions

`\tl_new:Nn` Use either `\tl_const:Nn` or `\tl_new:N`.

```

\tl_new:cn 4946 <deprecated>
\tl_new:Nx 4947 \cs_new_protected:Npn \tl_new:Nn #1#2
4948 {
4949   \tl_new:N #1
4950   \tl_gset:Nn #1 {#2}
4951 }
4952 \cs_generate_variant:Nn \tl_new:Nn { c }
4953 \cs_generate_variant:Nn \tl_new:Nn { Nx }
4954 </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 4955 <deprecated>

```

```

4956 \cs_new_protected_nopar:Npn \tl_gset:Nc
4957 { \tex_global:D \tl_set:Nc }
4958 \cs_new_protected_nopar:Npn \tl_set:Nc #1#2
4959 { \tl_set:No #1 { \cs:w #2 \cs_end: } }
4960 </deprecated>

```

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

\tl_replace_in:Nnn These are renamed.

```

\tl_replace_in:cnn 4961 <*deprecated>
\tl_greplace_in:Nnn 4962 \cs_new_eq:NN \tl_replace_in:Nnn \tl_replace_once:Nnn
\tl_greplace_in:cnn 4963 \cs_new_eq:NN \tl_replace_in:cnn \tl_replace_once:cnn
\tl_replace_all_in:Nnn 4964 \cs_new_eq:NN \tl_greplace_in:Nnn \tl_greplace_once:Nnn
\tl_replace_all_in:cnn 4965 \cs_new_eq:NN \tl_greplace_in:cnn \tl_greplace_once:cnn
\tl_greplace_all_in:Nnn 4966 \cs_new_eq:NN \tl_replace_all_in:Nnn \tl_replace_all:Nnn
\tl_greplace_all_in:cnn 4967 \cs_new_eq:NN \tl_replace_all_in:cnn \tl_replace_all:cnn
4968 \cs_new_eq:NN \tl_greplace_all_in:Nnn \tl_greplace_all:Nnn
4969 \cs_new_eq:NN \tl_greplace_all_in:cnn \tl_greplace_all:cnn
4970 </deprecated>

```

(End definition for \tl_replace_in:Nnn and \tl_replace_in:cnn. These functions are documented on page ??.)

\tl_remove_in:Nn Also renamed.

```

\tl_remove_in:cn 4971 <*deprecated>
\tl_gremove_in:Nn 4972 \cs_new_eq:NN \tl_remove_in:Nn \tl_remove_once:Nn
\tl_gremove_in:cn 4973 \cs_new_eq:NN \tl_remove_in:cn \tl_remove_once:cn
\tl_remove_all_in:Nn 4974 \cs_new_eq:NN \tl_gremove_in:Nn \tl_gremove_once:Nn
\tl_remove_all_in:cn 4975 \cs_new_eq:NN \tl_gremove_in:cn \tl_gremove_once:cn
\tl_gremove_all_in:Nn 4976 \cs_new_eq:NN \tl_remove_all_in:Nn \tl_remove_all:Nn
\tl_gremove_all_in:cn 4977 \cs_new_eq:NN \tl_remove_all_in:cn \tl_remove_all:cn
4978 \cs_new_eq:NN \tl_gremove_all_in:Nn \tl_gremove_all:Nn
4979 \cs_new_eq:NN \tl_gremove_all_in:cn \tl_gremove_all:cn
4980 </deprecated>

```

(End definition for \tl_remove_in:Nn and \tl_remove_in:cn. These functions are documented on page ??.)

\tl_elt_count:n Another renaming job.

```

\tl_elt_count:V 4981 <*deprecated>
\tl_elt_count:o 4982 \cs_new_eq:NN \tl_elt_count:n \tl_length:n
\tl_elt_count:N 4983 \cs_new_eq:NN \tl_elt_count:V \tl_length:V
\tl_elt_count:c 4984 \cs_new_eq:NN \tl_elt_count:o \tl_length:o
4985 \cs_new_eq:NN \tl_elt_count:N \tl_length:N
4986 \cs_new_eq:NN \tl_elt_count:c \tl_length:c
4987 </deprecated>

```

(End definition for \tl_elt_count:n, \tl_elt_count:V, and \tl_elt_count:o. These functions are documented on page ??.)

\tl_head_i:n Two renames, and a few that are rather too specialised.

```

\tl_head_i:w 4988 <*deprecated>
\tl_head_iii:n 4989 \cs_new_eq:NN \tl_head_i:n \tl_head:n
\tl_head_iii:f
\tl_head_iii:w

```

```

4990 \cs_new_eq:NN \tl_head_i:w \tl_head:w
4991 \cs_new:Npn \tl_head_iii:n #1 { \tl_head_iii:w #1 \q_stop }
4992 \cs_generate_variant:Nn \tl_head_iii:n { f }
4993 \cs_new:Npn \tl_head_iii:w #1#2#3#4 \q_stop {#1#2#3}
4994 \deprecated
      (End definition for \tl_head_i:n. This function is documented on page ??.)
4995 \initex | package)

```

185 l3seq implementation

The following test files are used for this code: *m3seq002,m3seq003*.

```

4996 \*initex | package)
4997 \*package)
4998 \ProvidesExplPackage
4999   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5000 \package_check_loaded_expl:
5001 \package)

```

A sequence is a control sequence whose top-level expansion is of the form “`\seq_item:n {⟨item0⟩} ... \seq_item:n {⟨itemn-1⟩}`”. An earlier implementation used the structure “`\seq_elt:w ⟨item1⟩ \seq_elt_end: ... \seq_elt:w ⟨itemn⟩ \seq_elt_end:`”. This allows rapid searching using a delimited function, but is not suitable for items containing `{`, `}` and `#` tokens, and also leads to the loss of surrounding braces around items.

`\seq_item:n` The delimiter is always defined, but when used incorrectly simply removes its argument and hits an undefined control sequence to raise an error.

```

5002 \cs_new:Npn \seq_item:n
5003 {
5004   \msg_expandable_error:n { A~sequence~was~used~incorrectly. }
5005   \use_none:n
5006 }
      (End definition for \seq_item:n. This function is documented on page 103.)

```

`\l_seq_tmpa_tl` Scratch space for various internal uses.

```

\l_seq_tmpp_tl
5007 \tl_new:N \l_seq_tmpa_tl
5008 \tl_new:N \l_seq_tmpp_tl
      (End definition for \l_seq_tmpa_tl and \l_seq_tmpp_tl. These functions are documented on
page ??.)

```

185.1 Allocation and initialisation

`\seq_new:N` Internally, sequences are just token lists.

`\seq_new:c` 5009 `\cs_new_eq:NN \seq_new:N \tl_new:N`
 5010 `\cs_new_eq:NN \seq_new:c \tl_new:c`

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

`\seq_clear:N` Clearing sequences is just the same as clearing token lists.

`\seq_clear:c` 5011 `\cs_new_eq:NN \seq_clear:N \tl_clear:N`
`\seq_gclear:N` 5012 `\cs_new_eq:NN \seq_clear:c \tl_clear:c`
`\seq_gclear:c` 5013 `\cs_new_eq:NN \seq_gclear:N \tl_gclear:N`
 5014 `\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` 5015 `\cs_new_eq:NN \seq_clear_new:N \tl_clear_new:N`
`\seq_gclear_new:N` 5016 `\cs_new_eq:NN \seq_clear_new:c \tl_clear_new:c`
`\seq_gclear_new:c` 5017 `\cs_new_eq:NN \seq_gclear_new:N \tl_gclear_new:N`
 5018 `\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` 5019 `\cs_new_eq:NN \seq_set_eq:NN \tl_set_eq:NN`
`\seq_set_eq:Nc` 5020 `\cs_new_eq:NN \seq_set_eq:Nc \tl_set_eq:Nc`
`\seq_set_eq:cc` 5021 `\cs_new_eq:NN \seq_set_eq:cN \tl_set_eq:cN`
`\seq_gset_eq:NN` 5022 `\cs_new_eq:NN \seq_set_eq:cc \tl_set_eq:cc`
`\seq_gset_eq:cN` 5023 `\cs_new_eq:NN \seq_gset_eq:NN \tl_gset_eq:NN`
`\seq_gset_eq:Nc` 5024 `\cs_new_eq:NN \seq_gset_eq:Nc \tl_gset_eq:Nc`
`\seq_gset_eq:cN` 5025 `\cs_new_eq:NN \seq_gset_eq:cN \tl_gset_eq:cN`
`\seq_gset_eq:cc` 5026 `\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` 5027 `\cs_new_protected_nopar:Npn \seq_concat:NNN #1#2#3`
`\seq_gconcat:NNN` 5028 `{ \tl_set:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }`
`\seq_gconcat:ccc` 5029 `\cs_new_protected_nopar:Npn \seq_gconcat:NNN #1#2#3`
 5030 `{ \tl_gset:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }`
 5031 `\cs_generate_variant:Nn \seq_concat:NNN { ccc }`
 5032 `\cs_generate_variant:Nn \seq_gconcat:NNN { ccc }`

(End definition for \seq_concat:NNN and \seq_concat:ccc. These functions are documented on page ??.)

185.2 Appending data to either end

```

\seq_put_left:Nn
\seq_put_left:NV
\seq_put_left:Nv
\seq_put_left:No
\seq_put_left:Nx
\seq_put_left:cn
\seq_put_left:cV
\seq_put_left:cv
\seq_put_left:co
\seq_put_left:cx

```

The code here is just a wrapper for adding to token lists.

```

5033 \cs_new_protected:Npn \seq_put_left:Nn #1#2
5034 { \tl_put_left:Nn #1 { \seq_item:n {#2} } }
5035 \cs_new_protected:Npn \seq_put_right:Nn #1#2
5036 { \tl_put_right:Nn #1 { \seq_item:n {#2} } }
5037 \cs_generate_variant:Nn \seq_put_left:Nn { NV , Nv , No , Nx }
5038 \cs_generate_variant:Nn \seq_put_left:Nn { c , cV , cv , co , cx }
5039 \cs_generate_variant:Nn \seq_put_right:Nn { NV , Nv , No , Nx }
5040 \cs_generate_variant:Nn \seq_put_right:Nn { c , cV , cv , co , cx }

```

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

```

\seq_gput_left:Nn
\seq_gput_left:NV
\seq_gput_left:Nv
\seq_gput_left:No
\seq_gput_left:Nx
\seq_gput_left:cn
\seq_gput_left:cV
\seq_gput_left:cv
\seq_gput_left:co
\seq_gput_left:cx

```

The same for global addition.

```

5041 \cs_new_protected:Npn \seq_gput_left:Nn #1#2
5042 { \tl_gput_left:Nn #1 { \seq_item:n {#2} } }
5043 \cs_new_protected:Npn \seq_gput_right:Nn #1#2
5044 { \tl_gput_right:Nn #1 { \seq_item:n {#2} } }
5045 \cs_generate_variant:Nn \seq_gput_left:Nn { NV , Nv , No , Nx }
5046 \cs_generate_variant:Nn \seq_gput_left:Nn { c , cV , cv , co , cx }
5047 \cs_generate_variant:Nn \seq_gput_right:Nn { NV , Nv , No , Nx }
5048 \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 ??.)

185.3 Modifying sequences

```

\seq_remove_duplicates:N
\seq_remove_duplicates:cV
\seq_remove_duplicates:c
\seq_remove_duplicates:cn
\seq_remove_duplicates:cv
\seq_remove_duplicates:co
\seq_remove_duplicates:cx
\seq_remove_duplicates_aux:NN

```

An internal sequence for the removal routines.

```

5049 \seq_new:N \l_seq_remove_seq

```

(End definition for `\l_seq_remove_seq`. This function is documented on page ??.)

Removing duplicates means making a new list then copying it.

```

5050 \cs_new_protected:Npn \seq_remove_duplicates:N
5051 { \seq_remove_duplicates_aux:NN \seq_set_eq:NN }
5052 \cs_new_protected:Npn \seq_gremove_duplicates:N
5053 { \seq_remove_duplicates_aux:NN \seq_gset_eq:NN }
5054 \cs_new_protected:Npn \seq_remove_duplicates_aux:NN #1#2
5055 {
5056   \seq_clear:N \l_seq_remove_seq
5057   \seq_map_inline:Nn #2
5058   {
5059     \seq_if_in:NnF \l_seq_remove_seq {##1}
5060     { \seq_put_right:Nn \l_seq_remove_seq {##1} }
5061   }
5062   #1 #2 \l_seq_remove_seq
5063 }
5064 \cs_generate_variant:Nn \seq_remove_duplicates:N { c }
5065 \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.

```

5066 \cs_new_protected:Npn \seq_remove_all:Nn
5067 { \seq_remove_all_aux:NNn \tl_set:Nx }
5068 \cs_new_protected:Npn \seq_gremove_all:Nn
5069 { \seq_remove_all_aux:NNn \tl_gset:Nx }
5070 \cs_new_protected:Npn \seq_remove_all_aux:NNn #1#2#3
5071 {
5072   \seq_push_item_def:n
5073   {
5074     \str_if_eq:nnT {##1} {#3}
5075     {
5076       \if_false: { \fi: }
5077       \tl_set:Nn \l_seq_tmpb_tl {##1}
5078       #1 #2
5079       { \if_false: } \fi:
5080       \exp_not:o {#2}
5081       \tl_if_eq:NNT \l_seq_tmpa_tl \l_seq_tmpb_tl
5082       { \use_none:nn }
5083     }
5084     \exp_not:n { \seq_item:n {##1} }
5085   }
5086   \tl_set:Nn \l_seq_tmpa_tl {#3}
5087   #1 #2 {#2}
5088   \seq_pop_item_def:
5089 }
5090 \cs_generate_variant:Nn \seq_remove_all:Nn { c }
5091 \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 ??.)

185.4 Sequence conditionals

`\seq_if_empty:N` Simple copies from the token list variable material.

```

\seq_if_empty:c
5092 \prg_new_eq_conditional:NNn \seq_if_empty:N \tl_if_empty:N
5093 { p , T , F , TF }
5094 \prg_new_eq_conditional:NNn \seq_if_empty:c \tl_if_empty:c
5095 { p , T , F , TF }

```

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

`\seq_if_in:Nn` The approach here is to define `\seq_item:n` to compare its argument with the test
`\seq_if_in:NV` sequence. If the two items are equal, the mapping is terminated and `\prg_return_-`
`\seq_if_in:Nv` `true:` is inserted. On the other hand, if there is no match then the loop will break
`\seq_if_in:No` returning `\prg_return_false:`. In either case, `\seq_break_point:n` ensures that the
`\seq_if_in:Nx` group ends before the logical value is returned. Everything is inside a group so that
`\seq_if_in:cn` `\seq_item:n` is preserved in nested situations.
`\seq_if_in:cV` 5096 `\prg_new_protected_conditional:Npnn \seq_if_in:Nn #1#2`
`\seq_if_in:cv` 5097 `{ T , F , TF }`
`\seq_if_in:co` 5098 `{`
`\seq_if_in:cx` 5099 `\group_begin:`
`\seq_if_in_aux:` 5100 `\tl_set:Nn \l_seq_tmpa_tl {#2}`
5101 `\cs_set_protected:Npn \seq_item:n ##1`
5102 `{`
5103 `\tl_set:Nn \l_seq_tmpb_tl {##1}`
5104 `\if_meaning:w \l_seq_tmpa_tl \l_seq_tmpb_tl`
5105 `\exp_after:wN \seq_if_in_aux:`
5106 `\fi:`
5107 `}`
5108 `#1`
5109 `\seq_break:n { \prg_return_false: }`
5110 `\seq_break_point:n { \group_end: }`
5111 `}`
5112 `\cs_new_nopar:Npn \seq_if_in_aux: { \seq_break:n { \prg_return_true: } }`
5113 `\cs_generate_variant:Nn \seq_if_in:NnT { NV , Nv , No , Nx }`
5114 `\cs_generate_variant:Nn \seq_if_in:NnT { c , cV , cv , co , cx }`
5115 `\cs_generate_variant:Nn \seq_if_in:NnF { NV , Nv , No , Nx }`
5116 `\cs_generate_variant:Nn \seq_if_in:NnF { c , cV , cv , co , cx }`
5117 `\cs_generate_variant:Nn \seq_if_in:NnTF { NV , Nv , No , Nx }`
5118 `\cs_generate_variant:Nn \seq_if_in:NnTF { c , cV , cv , co , cx }`
(End definition for \seq_if_in:Nn and others. These functions are documented on page ??.)

185.5 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` 5119 `\cs_new_protected_nopar:Npn \seq_get_left:NN #1#2`
5120 `{`
5121 `\seq_if_empty_err_break:N #1`
5122 `\exp_after:wN \seq_get_left_aux:NnwN #1 \q_stop #2`
5123 `\seq_break_point:n { }`
5124 `}`
5125 `\cs_new_protected:Npn \seq_get_left_aux:NnwN \seq_item:n #1#2 \q_stop #3`
5126 `{ \tl_set:Nn #3 {#1} }`
5127 `\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`

```

5128 \cs_new_protected_nopar:Npn \seq_pop_left:NN
5129 { \seq_pop_left_aux:NNN \tl_set:Nn }
5130 \cs_new_protected_nopar:Npn \seq_gpop_left:NN
5131 { \seq_pop_left_aux:NNN \tl_gset:Nn }
5132 \cs_new_protected_nopar:Npn \seq_pop_left_aux:NNN #1#2#3
5133 {
5134   \seq_if_empty_err_break:N #2
5135   \exp_after:wN \seq_pop_left_aux:NnwNNN #2 \q_stop #1#2#3
5136   \seq_break_point:n { }
5137 }
5138 \cs_new_protected:Npn \seq_pop_left_aux:NnwNNN \seq_item:n #1#2 \q_stop #3#4#5
5139 {
5140   #3 #4 {#2}
5141   \tl_set:Nn #5 {#1}
5142 }
5143 \cs_generate_variant:Nn \seq_pop_left:NN { c }
5144 \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`.

```

5145 \cs_new_protected_nopar:Npn \seq_get_right:NN #1#2
5146 {
5147   \seq_if_empty_err_break:N #1
5148   \seq_get_right_aux:NN #1#2
5149   \seq_break_point:n { }
5150 }
5151 \cs_new_protected_nopar:Npn \seq_get_right_aux:NN #1#2
5152 {
5153   \seq_push_item_def:n { }
5154   \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5155   \exp_after:wN \use_none:n #1
5156   { \tl_set:Nn #2 }
5157   { }
5158   {
5159     \seq_pop_item_def:
5160     \seq_break:
5161   }
5162 }
5163 \cs_new:Npn \seq_get_right_loop:nn #1#2
5164 {

```

```

5165     #2 {#1}
5166     \seq_get_right_loop:nn
5167   }
5168   \cs_generate_variant:Nn \seq_get_right:NN { c }

```

(End definition for `\seq_get_right:NN` and `\seq_get_right:cN`. These functions are documented on page ??.)

```

\seq_pop_right:NN
\seq_pop_right:cN
\seq_gpop_right:NN
\seq_gpop_right:cN
\seq_pop_right_aux:NNN
\seq_pop_right_aux_ii:NNN

```

The approach to popping from the right is a bit more involved, but does use some of the same ideas as getting from the right. What is needed is a “flexible length” way to set a token list variable. This is supplied by the `{ \if_false: } \fi: ... \if_false: { \fi: }` construct. Using an x-type expansion and a “non-expanding” definition for `\seq_item:n`, the left-most $n - 1$ entries in a sequence of n items will be stored back in the sequence. That needs a loop of unknown length, hence using the strange `\if_false:` way of including brackets. When the last item of the sequence is reached, the closing bracket for the assignment is inserted, and `\tl_set:Nn #3` is inserted in front of the final entry. This therefore does the pop assignment, then a final loop clears up the code.

```

5169   \cs_new_protected_nopar:Npn \seq_pop_right:NN
5170     { \seq_pop_right_aux:NNN \tl_set:Nx }
5171   \cs_new_protected_nopar:Npn \seq_gpop_right:NN
5172     { \seq_pop_right_aux:NNN \tl_gset:Nx }
5173   \cs_new_protected_nopar:Npn \seq_pop_right_aux:NNN #1#2#3
5174     {
5175       \seq_if_empty_err_break:N #2
5176       \seq_pop_right_aux_ii:NNN #1 #2 #3
5177       \seq_break_point:n { }
5178     }
5179   \cs_new_protected_nopar:Npn \seq_pop_right_aux_ii:NNN #1#2#3
5180     {
5181       \seq_push_item_def:n { \exp_not:n { \seq_item:n {##1} } }
5182       #1 #2 { \if_false: } \fi:
5183       \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5184       \exp_after:wN \use_none:n #2
5185       {
5186         \if_false: { \fi: }
5187         \tl_set:Nn #3
5188       }
5189       { }
5190       {
5191         \seq_pop_item_def:
5192         \seq_break:
5193       }
5194     }
5195   \cs_generate_variant:Nn \seq_pop_right:NN { c }
5196   \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 ??.)

185.6 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.

```
5197 \cs_new:Npn \seq_break: #1 \seq_break_point:n #2 {#2}
5198 \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 104.)
```

`\seq_map_break:` Semantically-logical copies of the break functions for use inside mappings.

```
\seq_map_break:n
5199 \cs_new_eq:NN \seq_map_break: \seq_break:
5200 \cs_new_eq:NN \seq_map_break:n \seq_break:n
      (End definition for \seq_map_break:.. This function is documented on page 100.)
```

`\seq_break_point:n` Normally, the marker token will not be executed, but if it is then the end code is simply inserted.

```
5201 \cs_new_eq:NN \seq_break_point:n \use:n
      (End definition for \seq_break_point:n. This function is documented on page 104.)
```

`\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.

```
5202 \cs_new_protected_nopar:Npn \seq_if_empty_err_break:N #1
5203 {
5204   \if_meaning:w #1 \c_empty_tl
5205     \msg_kernel_error:nxx { seq } { empty-sequence } { \token_to_str:N #1 }
5206     \exp_after:wN \seq_break:
5207   \fi:
5208 }
      (End definition for \seq_if_empty_err_break:N. This function is documented on page 103.)
```

`\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`

```
5209 \cs_new:Npn \seq_map_function:NN #1#2
5210 {
5211   \exp_after:wN \seq_map_function_aux:NNn \exp_after:wN #2 #1
5212   { ? \seq_map_break: } { }
5213   \seq_break_point:n { }
5214 }
5215 \cs_new:Npn \seq_map_function_aux:NNn #1#2#3
5216 {
5217   \use_none:n #2
5218   #1 {#3}
5219   \seq_map_function_aux:NNn #1
5220 }
5221 \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:`

```

5222 \cs_new_protected:Npn \seq_push_item_def:n
5223 {
5224   \seq_push_item_def_aux:
5225   \cs_gset:Npn \seq_item:n ##1
5226 }
5227 \cs_new_protected:Npn \seq_push_item_def:x
5228 {
5229   \seq_push_item_def_aux:
5230   \cs_gset:Npx \seq_item:n ##1
5231 }
5232 \cs_new_protected:Npn \seq_push_item_def_aux:
5233 {
5234   \cs_gset_eq:cN { seq_item_ \int_use:N \g_seq_nesting_depth_int :n }
5235   \seq_item:n
5236   \int_gincr:N \g_seq_nesting_depth_int
5237 }
5238 \cs_new_protected_nopar:Npn \seq_pop_item_def:
5239 {
5240   \int_gdecr:N \g_seq_nesting_depth_int
5241   \cs_gset_eq:Nc \seq_item:n
5242   { seq_item_ \int_use:N \g_seq_nesting_depth_int :n }
5243 }

```

(End definition for `\seq_push_item_def:n` and `\seq_push_item_def:x`. These functions are documented on page ??.)

`\seq_map_inline:Nn`

`\seq_map_inline:cn`

The idea here is that `\seq_item:n` is already “applied” to each item in a sequence, and so an in-line mapping is just a case of redefining `\seq_item:n`.

```

5244 \cs_new_protected:Npn \seq_map_inline:Nn #1#2
5245 {
5246   \seq_push_item_def:n {#2}
5247   #1
5248   \seq_break_point:n { \seq_pop_item_def: }
5249 }
5250 \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`

`\seq_map_variable:Ncn`

`\seq_map_variable:cNn`

`\seq_map_variable:ccn`

This is just a specialised version of the in-line mapping function, using an `x`-type expansion for the code set up so that the number of `#` tokens required is as expected.

```

5251 \cs_new_protected:Npn \seq_map_variable:NNn #1#2#3

```

```

5252 {
5253   \seq_push_item_def:x
5254   {
5255     \tl_set:Nn \exp_not:N #2 {##1}
5256     \exp_not:n {#3}
5257   }
5258   #1
5259   \seq_break_point:n { \seq_pop_item_def: }
5260 }
5261 \cs_generate_variant:Nn \seq_map_variable:NNn { Nc }
5262 \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 ??.)

185.7 Sequence stacks

The same functions as for sequences, but with the correct naming.

\seq_push:Nn Pushing to a sequence is the same as adding on the left.

```

\seq_push:NV 5263 \cs_new_eq:NN \seq_push:Nn \seq_put_left:Nn
\seq_push:Nv 5264 \cs_new_eq:NN \seq_push:NV \seq_put_left:NV
\seq_push:No 5265 \cs_new_eq:NN \seq_push:Nv \seq_put_left:Nv
\seq_push:Nx 5266 \cs_new_eq:NN \seq_push:No \seq_put_left:No
\seq_push:cn 5267 \cs_new_eq:NN \seq_push:Nx \seq_put_left:Nx
\seq_push:cV 5268 \cs_new_eq:NN \seq_push:cn \seq_put_left:cn
\seq_push:cV 5269 \cs_new_eq:NN \seq_push:cV \seq_put_left:cV
\seq_push:cv 5270 \cs_new_eq:NN \seq_push:cn \seq_put_left:cn
\seq_push:co 5271 \cs_new_eq:NN \seq_push:co \seq_put_left:co
\seq_push:cx 5272 \cs_new_eq:NN \seq_push:cx \seq_put_left:cx
\seq_gpush:Nn 5273 \cs_new_eq:NN \seq_gpush:Nn \seq_gput_left:Nn
\seq_gpush:NV 5274 \cs_new_eq:NN \seq_gpush:NV \seq_gput_left:NV
\seq_gpush:Nv 5275 \cs_new_eq:NN \seq_gpush:Nv \seq_gput_left:Nv
\seq_gpush:No 5276 \cs_new_eq:NN \seq_gpush:No \seq_gput_left:No
\seq_gpush:Nx 5277 \cs_new_eq:NN \seq_gpush:Nx \seq_gput_left:Nx
\seq_gpush:cn 5278 \cs_new_eq:NN \seq_gpush:cn \seq_gput_left:cn
\seq_gpush:cV 5279 \cs_new_eq:NN \seq_gpush:cV \seq_gput_left:cV
\seq_gpush:cv 5280 \cs_new_eq:NN \seq_gpush:cv \seq_gput_left:cv
\seq_gpush:co 5281 \cs_new_eq:NN \seq_gpush:co \seq_gput_left:co
\seq_gpush:cx 5282 \cs_new_eq:NN \seq_gpush:cx \seq_gput_left:cx

```

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

\seq_get:NN In most cases, getting items from the stack does not need to specify that this is from the left. So alias are provided.

```

\seq_get:cN 5283 \cs_new_eq:NN \seq_get:NN \seq_get_left:NN
\seq_pop:NN 5284 \cs_new_eq:NN \seq_get:cN \seq_get_left:cN
\seq_pop:cN 5285 \cs_new_eq:NN \seq_pop:NN \seq_pop_left:NN
\seq_gpop:NN 5286 \cs_new_eq:NN \seq_pop:cN \seq_pop_left:cN
\seq_gpop:cN 5287 \cs_new_eq:NN \seq_gpop:NN \seq_gpop_left:NN
5288 \cs_new_eq:NN \seq_gpop:cN \seq_gpop_left:cN

```

(End definition for \seq_get:NN and \seq_get:cN. These functions are documented on page ??.)

185.8 Viewing sequences

`\l_seq_show_tl` Used to store the material for display.

```
5289 \tl_new:N \l_seq_show_tl
```

(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 T_EX error if there is an empty sequence, a simple test is

`\seq_show_aux:w` used to keep the output “clean”.

```
5290 \cs_new_protected_nopar:Npn \seq_show:N #1
5291 {
5292   \seq_if_empty:NTF #1
5293   {
5294     \iow_term:x { Sequence~\token_to_str:N #1~is~empty }
5295     \tl_show:n { }
5296   }
5297   {
5298     \iow_term:x
5299     {
5300       Sequence~\token_to_str:N #1~
5301       contains~the~items~(without~outer~braces):
5302     }
5303     \tl_set:Nx \l_seq_show_tl
5304     { \seq_map_function:NN #1 \seq_show_aux:n }
5305     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
5306     { \exp_after:wN \seq_show_aux:w \l_seq_show_tl }
5307   }
5308 }
5309 \cs_new:Npn \seq_show_aux:n #1
5310 {
5311   \iow_newline: > \c_space_tl \c_space_tl
5312   \iow_char:N \{ \exp_not:n {#1} \iow_char:N \}
5313 }
5314 \cs_new:Npn \seq_show_aux:w #1 > ~ { }
5315 \cs_generate_variant:Nn \seq_show:N { c }
```

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

185.9 Experimental functions

`\seq_if_empty_break_return_false:N` The name says it all: of the sequence is empty, returns logical **false**.

```
5316 \cs_new_nopar:Npn \seq_if_empty_break_return_false:N #1
5317 {
5318   \if_meaning:w #1 \c_empty_tl
5319   \prg_return_false:
5320   \exp_after:wN \seq_break:
5321   \fi:
5322 }
```

(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 5323 \prg_new_protected_conditional:Npnn \seq_get_left:NN #1 #2 { T , F , TF }
 \seq_get_right:NN 5324 {
 \seq_get_right:cN 5325 \seq_if_empty_break_return_false:N #1
 5326 \exp_after:wN \seq_get_left_aux:Nw #1 \q_stop #2
 5327 \prg_return_true:
 5328 \seq_break:
 5329 \seq_break_point:n { }
 5330 }
 5331 \prg_new_protected_conditional:Npnn \seq_get_right:NN #1#2 { T , F , TF }
 5332 {
 5333 \seq_if_empty_break_return_false:N #1
 5334 \seq_get_right_aux:NN #1#2
 5335 \prg_return_true: \seq_break:
 5336 \seq_break_point:n { }
 5337 }
 5338 \cs_generate_variant:Nn \seq_get_left:NNT { c }
 5339 \cs_generate_variant:Nn \seq_get_left:NNF { c }
 5340 \cs_generate_variant:Nn \seq_get_left:NNTF { c }
 5341 \cs_generate_variant:Nn \seq_get_right:NNT { c }
 5342 \cs_generate_variant:Nn \seq_get_right:NNF { c }
 5343 \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 5344 \prg_new_protected_conditional:Npnn \seq_pop_left:NN #1#2 { T , F , TF }
 \seq_gpop_left:NN 5345 {
 \seq_gpop_left:cN 5346 \seq_if_empty_break_return_false:N #1
 \seq_pop_right:NN 5347 \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_set:Nn #1#2
 \seq_pop_right:cN 5348 \prg_return_true: \seq_break:
 \seq_gpop_right:NN 5349 \seq_break_point:n { }
 \seq_gpop_right:cN 5350 }
 5351 \prg_new_protected_conditional:Npnn \seq_gpop_left:NN #1#2 { T , F , TF }
 5352 {
 5353 \seq_if_empty_break_return_false:N #1
 5354 \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_gset:Nn #1#2
 5355 \prg_return_true: \seq_break:
 5356 \seq_break_point:n { }
 5357 }
 5358 \prg_new_protected_conditional:Npnn \seq_pop_right:NN #1#2 { T , F , TF }
 5359 {
 5360 \seq_if_empty_break_return_false:N #1
 5361 \seq_pop_right_aux_ii:NNN \tl_set:Nx #1 #2
 5362 \prg_return_true: \seq_break:
 5363 \seq_break_point:n { }
 5364 }

```

5365 \prg_new_protected_conditional:Npnn \seq_gpop_right:NN #1#2 { T , F , TF }
5366 {
5367   \seq_if_empty_break_return_false:N #1
5368   \seq_pop_right_aux_ii:NNN \tl_gset:Nx #1 #2
5369   \prg_return_true: \seq_break:
5370   \seq_break_point:n { }
5371 }
5372 \cs_generate_variant:Nn \seq_pop_left:NNT { c }
5373 \cs_generate_variant:Nn \seq_pop_left:NNF { c }
5374 \cs_generate_variant:Nn \seq_pop_left:NNTF { c }
5375 \cs_generate_variant:Nn \seq_gpop_left:NNT { c }
5376 \cs_generate_variant:Nn \seq_gpop_left:NNF { c }
5377 \cs_generate_variant:Nn \seq_gpop_left:NNTF { c }
5378 \cs_generate_variant:Nn \seq_pop_right:NNT { c }
5379 \cs_generate_variant:Nn \seq_pop_right:NNF { c }
5380 \cs_generate_variant:Nn \seq_pop_right:NNTF { c }
5381 \cs_generate_variant:Nn \seq_gpop_right:NNT { c }
5382 \cs_generate_variant:Nn \seq_gpop_right:NNF { c }
5383 \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`

```

5384 \cs_new:Npn \seq_length:N #1
5385 {
5386   \int_eval:n
5387   {
5388     0
5389     \seq_map_function:NN #1 \seq_length_aux:n
5390   }
5391 }
5392 \cs_new:Npn \seq_length_aux:n #1 { +1 }
5393 \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`

```

5394 \cs_new_nopar:Npn \seq_item:Nn #1#2
5395 {
5396   \exp_last_unbraced:Nfo \seq_item_aux:nnn
5397   {
5398     \int_eval:n
5399     {
5400       \int_compare:nNnT {#2} < \c_zero
5401       { \seq_length:N #1 + }

```

```

5402         #2
5403     }
5404 }
5405 #1
5406 { ? \seq_break: }
5407 { }
5408 \seq_break_point:n { }
5409 }
5410 \cs_new_nopar:Npn \seq_item_aux:nnn #1#2#3
5411 {
5412     \use_none:n #2
5413     \int_compare:nNnTF {#1} = \c_zero
5414     { \seq_break:n {#3} }
5415     { \exp_args:Nf \seq_item_aux:nnn { #1 - 1 } }
5416 }
5417 \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
5418 \cs_new_nopar:Npn \seq_use:N #1 { \seq_map_function:NN #1 \use:n }
5419 \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: } { } to the end of each on. This is most conveniently done in two steps using an auxiliary function. The mapping then throws away the first token of #2 and #5, which for items in the sequences will both be \seq_item:n. The function to be mapped will then be applied to the two entries. When the code hits the end of one of the sequences, the break material will stop the entire loop and tidy up. This avoids needing to find the length of the two sequences, or worrying about which is longer.

```

5420 \cs_new_nopar:Npn \seq_mapthread_function:NNN #1#2#3
5421 {
5422     \exp_after:wN \seq_mapthread_function_aux:NN
5423     \exp_after:wN #3
5424     \exp_after:wN #1
5425     #2
5426     { ? \seq_break: } { }
5427     \seq_break_point:n { }
5428 }
5429 \cs_new_nopar:Npn \seq_mapthread_function_aux:NN #1#2
5430 {
5431     \exp_after:wN \seq_mapthread_function_aux:Nnnwnn
5432     \exp_after:wN #1
5433     #2
5434     { ? \seq_break: } { }
5435     \q_stop
5436 }
5437 \cs_new:Npn \seq_mapthread_function_aux:Nnnwnn #1#2#3#4 \q_stop #5#6

```

```

5438 {
5439   \use_none:n #2
5440   \use_none:n #5
5441   #1 {#3} {#6}
5442   \seq_mapthread_function_aux:Nnnwnn #1 #4 \q_stop
5443 }
5444 \cs_generate_variant:Nn \seq_mapthread_function:NNN { Nc }
5445 \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 5446 \cs_new_protected:Npn \seq_set_from_clist:NN #1#2
\seq_set_from_clist:Nc 5447 {
\seq_set_from_clist:cc 5448   \tl_set:Nx #1
\seq_set_from_clist:Nn 5449   { \clist_map_function:NN #2 \seq_wrap_item:n }
\seq_set_from_clist:cn 5450 }
\seq_gset_from_clist:NN 5451 \cs_new_protected:Npn \seq_gset_from_clist:NN #1#2
\seq_gset_from_clist:cN 5452 {
\seq_gset_from_clist:Nc 5453   \tl_set:Nx #1
\seq_gset_from_clist:cc 5454   { \clist_map_function:nN {#2} \seq_wrap_item:n }
\seq_gset_from_clist:Nn 5455 }
\seq_gset_from_clist:cn 5456 \cs_new_protected:Npn \seq_gset_from_clist:NN #1#2
\seq_wrap_item:n 5457 {
5458   \tl_gset:Nx #1
5459   { \clist_map_function:NN #2 \seq_wrap_item:n }
5460 }
5461 \cs_new_protected:Npn \seq_gset_from_clist:Nn #1#2
5462 {
5463   \tl_gset:Nx #1
5464   { \clist_map_function:nN {#2} \seq_wrap_item:n }
5465 }
5466 \cs_new:Npn \seq_wrap_item:n #1 { \exp_not:n { \seq_item:n {#1} } }
5467 \cs_generate_variant:Nn \seq_set_from_clist:NN { Nc }
5468 \cs_generate_variant:Nn \seq_set_from_clist:NN { c , cc }
5469 \cs_generate_variant:Nn \seq_set_from_clist:Nn { c }
5470 \cs_generate_variant:Nn \seq_gset_from_clist:NN { Nc }
5471 \cs_generate_variant:Nn \seq_gset_from_clist:NN { c , cc }
5472 \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.

```

5473 \cs_new_protected_nopar:Npn \seq_tmp:w
5474 { \msg_expandable_error:n { There-is-a-bug-in-LaTeX3! } }
5475 \cs_new_protected_nopar:Npn \seq_set_reverse:N
5476 { \seq_reverse_aux:NN \tl_set:Nx }
5477 \cs_new_protected_nopar:Npn \seq_gset_reverse:N

```

```

5478 { \seq_reverse_aux:NN \tl_gset:Nx }
5479 \cs_new_protected_nopar:Npn \seq_reverse_aux:NN #1 #2
5480 {
5481   \cs_set_eq:NN \seq_tmp:w \seq_item:n
5482   \cs_set_eq:NN \seq_item:n \seq_reverse_aux_item:w
5483   #1 #2 { #2 \prg_do_nothing: }
5484   \cs_set_eq:NN \seq_item:n \seq_tmp:w
5485 }
5486 \cs_new:Npn \seq_reverse_aux_item:w #1 #2 \prg_do_nothing:
5487 {
5488   #2
5489   \prg_do_nothing:
5490   \exp_not:n { \seq_item:n {#1} }
5491 }

```

(End definition for `\seq_set_reverse:N` and `\seq_gset_reverse:N`. These functions are documented on page 103.)

`\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.

```

5492 \cs_new_protected_nopar:Npn \seq_set_split:Nnn
5493 { \seq_set_split_aux:NNnn \tl_set:Nx }
5494 \cs_new_protected_nopar:Npn \seq_gset_split:Nnn
5495 { \seq_set_split_aux:NNnn \tl_gset:Nx }
5496 \cs_new_protected_nopar:Npn \seq_set_split_aux:NNnn #1 #2 #3 #4
5497 {
5498   \tl_if_empty:nTF {#4}
5499   { #1 #2 { } }
5500   {
5501     \tl_set:Nn \l_seq_tmpa_tl
5502     {
5503       \seq_set_split_aux_i:w \prg_do_nothing:
5504       #4
5505       \seq_set_split_aux_end:
5506     }
5507     \tl_replace_all:Nnn \l_seq_tmpa_tl { #3 }
5508     {
5509       \seq_set_split_aux_end:
5510       \seq_set_split_aux_i:w \prg_do_nothing:
5511     }
5512     \tl_set:Nx \l_seq_tmpa_tl { \l_seq_tmpa_tl }
5513     #1 #2 { \l_seq_tmpa_tl }

```

```

5514     }
5515   }
5516   \cs_new:Npn \seq_set_split_aux_i:w #1 \seq_set_split_aux_end:
5517   {
5518     \exp_not:N \seq_set_split_aux_ii:w
5519     \exp_args:No \tl_trim_spaces:n {#1}
5520     \exp_not:N \seq_set_split_aux_end:
5521   }
5522   \cs_new:Npn \seq_set_split_aux_ii:w #1 \seq_set_split_aux_end:
5523   { \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 ??.)

185.10 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 5524 <*deprecated>
5525 \cs_new_eq:NN \seq_top:NN \seq_get_left:NN
5526 \cs_new_eq:NN \seq_top:cN \seq_get_left:cN
5527 </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 5528 <*deprecated>
5529 \cs_new_eq:NN \seq_display:N \seq_show:N
5530 \cs_new_eq:NN \seq_display:c \seq_show:c
5531 </deprecated>

```

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

```

5532 </initex | package>

```

186 l3clist implementation

The following test files are used for this code: m3clist002.

```

5533 <*initex | package>
5534 <*package>
5535 \ProvidesExplPackage
5536   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5537 \package_check_loaded_expl:
5538 </package>

```

\l_clist_tmpa_tl Scratch space for various internal uses.

```

5539 \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.

5540 \cs_new_protected:Npn \clist_tmp:w { }

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

186.1 Allocation and initialisation

\clist_new:N Internally, comma lists are just token lists.

\clist_new:c 5541 \cs_new_eq:NN \clist_new:N \tl_new:N

5542 \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 5543 \cs_new_eq:NN \clist_clear:N \tl_clear:N

\clist_gclear:N 5544 \cs_new_eq:NN \clist_clear:c \tl_clear:c

\clist_gclear:c 5545 \cs_new_eq:NN \clist_gclear:N \tl_gclear:N

5546 \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 5547 \cs_new_eq:NN \clist_clear_new:N \tl_clear_new:N

\clist_gclear_new:N 5548 \cs_new_eq:NN \clist_clear_new:c \tl_clear_new:c

\clist_gclear_new:c 5549 \cs_new_eq:NN \clist_gclear_new:N \tl_gclear_new:N

5550 \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 5551 \cs_new_eq:NN \clist_set_eq:NN \tl_set_eq:NN

\clist_set_eq:Nc 5552 \cs_new_eq:NN \clist_set_eq:Nc \tl_set_eq:Nc

\clist_set_eq:cc 5553 \cs_new_eq:NN \clist_set_eq:cN \tl_set_eq:cN

\clist_gset_eq:NN 5554 \cs_new_eq:NN \clist_set_eq:cc \tl_set_eq:cc

\clist_gset_eq:cN 5555 \cs_new_eq:NN \clist_gset_eq:NN \tl_gset_eq:NN

\clist_gset_eq:Nc 5556 \cs_new_eq:NN \clist_gset_eq:Nc \tl_gset_eq:Nc

\clist_gset_eq:cN 5557 \cs_new_eq:NN \clist_gset_eq:cN \tl_gset_eq:cN

5558 \cs_new_eq:NN \clist_gset_eq:cc \tl_gset_eq:cc

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

\clist_concat:NNN Concatenating sequences is not quite as easy as it seems, as there needs to be the correct addition of a comma to the output. So a little work to do.

\clist_concat:ccc 5559 \cs_new_protected_nopar:Npn \clist_concat:NNN

\clist_gconcat:NNN 5560 { \clist_concat_aux:NNNN \tl_set:Nx }

\clist_gconcat:ccc 5561 \cs_new_protected_nopar:Npn \clist_gconcat:NNN

5562 { \clist_concat_aux:NNNN \tl_gset:Nx }

5563 \cs_new_protected_nopar:Npn \clist_concat_aux:NNNN #1#2#3#4


```

5564 {
5565   #1 #2
5566   {
5567     \exp_not:o #3
5568     \clist_if_empty:NF #3 { \clist_if_empty:NF #4 { , } }
5569     \exp_not:o #4
5570   }
5571 }
5572 \cs_generate_variant:Nn \clist_concat:NNN { ccc }
5573 \cs_generate_variant:Nn \clist_gconcat:NNN { ccc }

```

(End definition for `\clist_concat:NNN` and `\clist_concat:ccc`. These functions are documented on page ??.)

186.2 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`.

```

5574 \cs_set:Npn \clist_tmp:w #1
5575 {
5576   \cs_new:Npn \clist_trim_spaces_generic:nw ##1 ##2 ,
5577   {
5578     \tl_trim_spaces_aux_i:w
5579     ##2
5580     \q_nil
5581     \q_mark #1 { }
5582     \q_mark \tl_trim_spaces_aux_ii:w
5583     \tl_trim_spaces_aux_iii:w
5584     #1 \q_nil
5585     \clist_trim_spaces_generic_aux:w
5586     \q_stop
5587     {##1}
5588   }
5589 }
5590 \clist_tmp:w {~}
5591 \cs_new:Npn \clist_trim_spaces_generic_aux:w #1 \q_nil #2 \q_stop
5592 { \exp_args:No \clist_trim_spaces_generic_aux_ii:nn { \use_none:n #1 } }
5593 \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.

```

5594 \cs_new:Npn \clist_trim_spaces:n #1
5595 {
5596   \clist_trim_spaces_aux:n { } \q_mark #1 ,
5597   \q_recursion_tail, \q_recursion_stop
5598 }
5599 \cs_new:Npn \clist_trim_spaces_aux:n #1
5600 {
5601   \clist_trim_spaces_generic:nw
5602   { \clist_trim_spaces_aux_ii:nn {#1} }
5603 }
5604 \cs_new:Npn \clist_trim_spaces_aux_ii:nn #1 #2
5605 {
5606   \quark_if_recursion_tail_stop:n {#2}
5607   \tl_if_empty:nTF {#2}
5608   {
5609     \clist_trim_spaces_aux:n {#1} \q_mark
5610   }
5611   {
5612     #1 \exp_not:n {#2}
5613     \clist_trim_spaces_aux:n { , } \q_mark
5614   }
5615 }

```

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

186.3 Adding data to comma lists

```

\clist_set:Nn
\clist_set:NV 5616 \cs_new_protected:Npn \clist_set:Nn #1#2
\clist_set:No 5617 { \tl_set:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:Nx 5618 \cs_new_protected:Npn \clist_gset:Nn #1#2
\clist_set:cn 5619 { \tl_gset:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:cV 5620 \cs_generate_variant:Nn \clist_set:Nn { NV , No , Nx , c , cV , co , cx }
\clist_set:co 5621 \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

```

```

5622 \cs_new_protected_nopar:Npn \clist_put_left:Nn
5623 { \clist_put_aux:NnnNn \tl_set_eq:NN \tl_put_left:Nx { } , }
5624 \cs_new_protected_nopar:Npn \clist_gput_left:Nn
5625 { \clist_put_aux:NnnNn \tl_gset_eq:NN \tl_gput_left:Nx { } , }
5626 \cs_new_protected:Npn \clist_put_aux:NnnNn #1#2#3#4#5#6
5627 {
5628   \tl_set:Nx \l_clist_tmpa_tl { \clist_trim_spaces:n {#6} }
5629   \clist_if_empty:NTF #5
5630   { #1 #5 \l_clist_tmpa_tl }
5631   {
5632     \tl_if_empty:NF \l_clist_tmpa_tl
5633     { #2 #5 { #3 \exp_not:o \l_clist_tmpa_tl #4 } }

```

```

5634     }
5635 }
5636 \cs_generate_variant:Nn \clist_put_left:Nn { NV , No , Nx }
5637 \cs_generate_variant:Nn \clist_put_left:Nn { c , cV , co , cx }
5638 \cs_generate_variant:Nn \clist_gput_left:Nn { NV , No , Nx }
5639 \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 ??.)

```

\clist_put_right:Nn
\clist_put_right:NV
\clist_put_right:No
\clist_put_right:Nx
\clist_put_right:cn
\clist_put_right:cV
\clist_put_right:co
\clist_put_right:cx
\clist_gput_right:Nn
\clist_gput_right:NV
\clist_gput_right:No
\clist_gput_right:Nx
\clist_gput_right:cn
\clist_gput_right:cV
\clist_gput_right:co
\clist_gput_right:cx
\clist_get:Nn
\clist_get:NV
\clist_get:No
\clist_get:Nx
\clist_get:cn
\clist_get:cV
\clist_get:co
\clist_get:cx
\clist_get_aux:Nn
\clist_get_aux:NV
\clist_get_aux:No
\clist_get_aux:Nx
\clist_get_aux:cn
\clist_get_aux:cV
\clist_get_aux:co
\clist_get_aux:cx

```

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

186.4 Comma lists as stacks

Getting an item from the left of a comma list is pretty easy: just trim off the first item using the comma.

```

5648 \cs_new_protected_nopar:Npn \clist_get:NN #1#2
5649 { \exp_after:wN \clist_get_aux:wN #1 , \q_stop #2 }
5650 \cs_new_protected:Npn \clist_get_aux:wN #1 , #2 \q_stop #3
5651 { \tl_set:Nn #3 {#1} }
5652 \cs_generate_variant:Nn \clist_get:NN { c }

```

(End definition for \clist_get:NN and \clist_get:cN. These functions are documented on page ??.)

```

\clist_pop:NN
\clist_pop:cN
\clist_gpop:NN
\clist_gpop:cN
\clist_pop_aux:NNN
\clist_pop_aux:NwNNN
\clist_pop_aux:wNN

```

The aim here is to get the popped item as #1 in the auxiliary, with #2 containing either the remainder of the list or \q_nil if there were insufficient items. That keeps the number of auxiliary functions down.

```

5653 \cs_new_protected_nopar:Npn \clist_pop:NN
5654 { \clist_pop_aux:NNN \tl_set:Nf }
5655 \cs_new_protected_nopar:Npn \clist_gpop:NN
5656 { \clist_pop_aux:NNN \tl_gset:Nf }
5657 \cs_new_protected_nopar:Npn \clist_pop_aux:NNN #1#2#3
5658 {
5659   \exp_after:wN \clist_pop_aux:wNNN #2 , \q_nil \q_stop #1#2#3
5660 }
5661 \cs_new_protected:Npn \clist_pop_aux:wNNN #1 , #2 \q_stop #3#4#5
5662 {
5663   \tl_set:Nn #5 {#1}
5664   \quark_if_nil:nTF {#2}
5665   { #3 #4 { } }
5666   { #3 #4 { \clist_pop_aux:w \exp_stop_f: #2 } }

```

```

5667 }
5668 \cs_new_protected:Npn \clist_pop_aux:w #1 , \q_nil {#1}
5669 \cs_generate_variant:Nn \clist_pop:NN { c }
5670 \cs_generate_variant:Nn \clist_gpop:NN { c }

```

(End definition for \clist_pop:NN and \clist_pop:cN. These functions are documented on page ??.)

\clist_push:Nn Pushing to a sequence is the same as adding on the left.

```

\clist_push:NV 5671 \cs_new_eq:NN \clist_push:Nn \clist_put_left:Nn
\clist_push:No 5672 \cs_new_eq:NN \clist_push:NV \clist_put_left:NV
\clist_push:Nx 5673 \cs_new_eq:NN \clist_push:No \clist_put_left:No
\clist_push:cn 5674 \cs_new_eq:NN \clist_push:Nx \clist_put_left:Nx
\clist_push:cV 5675 \cs_new_eq:NN \clist_push:cn \clist_put_left:cn
\clist_push:co 5676 \cs_new_eq:NN \clist_push:cV \clist_put_left:cV
\clist_push:cx 5677 \cs_new_eq:NN \clist_push:co \clist_put_left:co
\clist_gpush:Nn 5678 \cs_new_eq:NN \clist_gpush:cx \clist_put_left:cx
\clist_gpush:NV 5679 \cs_new_eq:NN \clist_gpush:Nn \clist_gput_left:Nn
\clist_gpush:No 5680 \cs_new_eq:NN \clist_gpush:NV \clist_gput_left:NV
\clist_gpush:Nx 5681 \cs_new_eq:NN \clist_gpush:No \clist_gput_left:No
\clist_gpush:cn 5682 \cs_new_eq:NN \clist_gpush:Nx \clist_gput_left:Nx
\clist_gpush:cV 5683 \cs_new_eq:NN \clist_gpush:cn \clist_gput_left:cn
\clist_gpush:co 5684 \cs_new_eq:NN \clist_gpush:cV \clist_gput_left:cV
\clist_gpush:cx 5685 \cs_new_eq:NN \clist_gpush:co \clist_gput_left:co
5686 \cs_new_eq:NN \clist_gpush:cx \clist_gput_left:cx

```

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

186.5 Using comma lists

\clist_use:N The approach is the same as for \tl_use:N.
\clist_use:c

```

5687 \cs_new_eq:NN \clist_use:N \tl_use:N
5688 \cs_new_eq:NN \clist_use:c \tl_use:c

```

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

186.6 Modifying comma lists

\l_clist_remove_clist An internal comma list for the removal routines.

```

5689 \clist_new:N \l_clist_remove_clist

```

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

\clist_remove_duplicates:N Removing duplicates means making a new list then copying it.
\clist_remove_duplicates:c
\clist_gremove_duplicates:N
\clist_gremove_duplicates:c

```

5690 \cs_new_protected:Npn \clist_remove_duplicates:N
5691 { \clist_remove_duplicates_aux:NN \clist_set_eq:NN }
5692 \cs_new_protected:Npn \clist_gremove_duplicates:N
5693 { \clist_remove_duplicates_aux:NN \clist_gset_eq:NN }
5694 \cs_new_protected:Npn \clist_remove_duplicates_aux:NN #1#2
5695 {
5696   \clist_clear:N \l_clist_remove_clist

```

```

5697 \clist_map_inline:Nn #2
5698 {
5699     \clist_if_in:NnF \l_clist_remove_clist {##1}
5700     { \clist_put_right:Nn \l_clist_remove_clist {##1} }
5701 }
5702 #1 #2 \l_clist_remove_clist
5703 }
5704 \cs_generate_variant:Nn \clist_remove_duplicates:N { c }
5705 \cs_generate_variant:Nn \clist_gremove_duplicates:N { c }

```

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

`\clist_remove_all:Nn` The method used here is very similar to `\tl_replace_all:Nnn`. Build a function delimited by the *<item>* that should be removed, surrounded with commas, and call that function followed by the expanded comma list, and another copy of the *<item>*. The loop is controlled by the argument grabbed by `\clist_remove_all_aux:w`: when the item was found, the `\q_mark` delimiter used is the one inserted by `\clist_tmp:w`, and `\use_none_delimit_by_q_stop:w` is deleted. At the end, the final *<item>* is grabbed, and the argument of `\clist_tmp:w` contains `\q_mark`: in that case, `\clist_remove_all_aux:w` removes the second `\q_mark` (inserted by `\clist_tmp:w`), and lets `\use_none_delimit_by_q_stop:w` act.

No brace is lost because items are always grabbed with a leading comma. The result of the first assignment has an extra leading comma, which we remove in a second assignment. Two exceptions: if the clist lost all of its elements, the result is empty, and we shouldn't remove anything; if the clist started up empty, the first step happens to turn it into a single comma, and the second step removes it.

```

5706 \cs_new_protected:Npn \clist_remove_all:Nn
5707 { \clist_remove_all_aux:NNn \tl_set:Nx }
5708 \cs_new_protected:Npn \clist_gremove_all:Nn
5709 { \clist_remove_all_aux:NNn \tl_gset:Nx }
5710 \cs_new_protected:Npn \clist_remove_all_aux:NNn #1#2#3
5711 {
5712     \cs_set:Npn \clist_tmp:w ##1 , #3 ,
5713     {
5714         ##1
5715         , \q_mark , \use_none_delimit_by_q_stop:w ,
5716         \clist_remove_all_aux:
5717     }
5718     #1 #2
5719     {
5720         \exp_after:wN \clist_remove_all_aux:
5721         #2 , \q_mark , #3 , \q_stop
5722     }
5723     \clist_if_empty:NF #2
5724     {
5725         #1 #2
5726         {
5727             \exp_args:No \exp_not:o

```

```

5728             { \exp_after:wN \use_none:n #2 }
5729         }
5730     }
5731 }
5732 \cs_new:Npn \clist_remove_all_aux:
5733 { \exp_after:wN \clist_remove_all_aux:w \clist_tmp:w , }
5734 \cs_new:Npn \clist_remove_all_aux:w #1 , \q_mark , #2 , { \exp_not:n {#1} }
5735 \cs_generate_variant:Nn \clist_remove_all:Nn { c }
5736 \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 ??.)

```

186.7 Comma list conditionals

`\l_clist_if_in_clist` An internal comma list for `\clist_if_in:nn` conditionals.

```

5737 \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
5738 \prg_new_eq_conditional:NNn \clist_if_empty:N \tl_if_empty:N { p , T , F , TF }
5739 \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
5740 \prg_new_eq_conditional:NNn \clist_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\clist_if_eq:cN
5741 \prg_new_eq_conditional:NNn \clist_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
\clist_if_eq:cc
5742 \prg_new_eq_conditional:NNn \clist_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
5743 \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
5744 \prg_new_protected_conditional:Npnn \clist_if_in:Nn #1#2 { T , F , TF }
\clist_if_in:No
5745 {
\clist_if_in:cV
5746     \exp_args:No \clist_if_in_return:nn #1 {#2}
\clist_if_in:co
5747 }
\clist_if_in:nn
5748 \prg_new_protected_conditional:Npnn \clist_if_in:nn #1#2 { T , F , TF }
\clist_if_in:nV
5749 {
\clist_if_in:no
5750     \clist_set:Nn \l_clist_if_in_clist {#1}
5751     \exp_args:No \clist_if_in_return:nn \l_clist_if_in_clist {#2}
\clist_if_in_return:nn
5752 }
5753 \cs_new_protected:Npn \clist_if_in_return:nn #1#2
5754 {
5755     \cs_set:Npn \clist_tmp:w ##1 ,#2, { }
5756     \tl_if_empty:oTF
5757     { \clist_tmp:w ,#1, {} {} } ,#2, { }
5758     { \prg_return_false: } { \prg_return_true: }

```

```

5759 }
5760 \cs_generate_variant:Nn \clist_if_in:NnT { c , cV , co }
5761 \cs_generate_variant:Nn \clist_if_in:NnT { c , cV , co }
5762 \cs_generate_variant:Nn \clist_if_in:NnF { c , cV , co }
5763 \cs_generate_variant:Nn \clist_if_in:NnF { c , cV , co }
5764 \cs_generate_variant:Nn \clist_if_in:NnTF { c , cV , co }
5765 \cs_generate_variant:Nn \clist_if_in:NnTF { c , cV , co }
5766 \cs_generate_variant:Nn \clist_if_in:nnT { c , cV , co }
5767 \cs_generate_variant:Nn \clist_if_in:nnF { c , cV , co }
5768 \cs_generate_variant:Nn \clist_if_in:nnTF { c , cV , co }

```

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

186.8 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

```

5769 \cs_new_nopar:Npn \clist_map_function:NN #1#2
5770 {
5771   \clist_if_empty:NF #1
5772   {
5773     \exp_last_unbraced:NNo \clist_map_function_aux:Nw #2 #1
5774     , \q_recursion_tail , \q_recursion_stop
5775   }
5776 }
5777 \cs_new:Npn \clist_map_function_aux:Nw #1#2 ,
5778 {
5779   \quark_if_recursion_tail_stop:n {#2}
5780   #1 {#2}
5781   \clist_map_function_aux:Nw #1
5782 }
5783 \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.

```

5784 \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 TeX’s groups.
\clist_map_inline:nn

```

5785 \cs_new_protected:Npn \clist_map_inline:Nn #1#2
5786 {
5787   \int_gincr:N \g_clist_map_inline_int
5788   \cs_gset:cpn { clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5789   ##1
5790   {#2}
5791   \exp_args:NNc \clist_map_function:NN #1
5792   { clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }

```

```

5793     \int_gdecr:N \g_clist_map_inline_int
5794   }
5795   \cs_new_protected:Npn \clist_map_inline:nn #1#2
5796   {
5797     \int_gincr:N \g_clist_map_inline_int
5798     \cs_gset:cpn { \clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5799     ##1
5800     {#2}
5801     \exp_args:Nnc \clist_map_function:nN {#1}
5802     { \clist_map_inline_ \int_use:N \g_clist_map_inline_int :n }
5803     \int_gdecr:N \g_clist_map_inline_int
5804   }
5805   \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

```

5806   \cs_new_protected:Npn \clist_map_variable:NNn #1#2#3
5807   {
5808     \clist_if_empty:NF #1
5809     {
5810       \exp_args:Nno \use:nn
5811       { \clist_map_variable_aux:Nnw #2 {#3} }
5812       #1
5813       , \q_recursion_tail , \q_recursion_stop
5814     }
5815   }
5816   \cs_new_protected:Npn \clist_map_variable_aux:Nnw #1#2#3 ,
5817   {
5818     \quark_if_recursion_tail_stop:n {#3}
5819     \tl_set:Nn #1 {#3}
5820     #2
5821     \clist_map_variable_aux:Nnw #1 {#2}
5822   }
5823   \cs_new_protected:Npn \clist_map_variable:nNn #1
5824   {
5825     \tl_set:Nx \l_clist_tmpa_tl { \clist_trim_spaces:n {#1} }
5826     \clist_map_variable:NNn \l_clist_tmpa_tl
5827   }
5828   \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.

```

5829   \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`.

```

5830 \cs_new:Npn \clist_map_function:nN #1#2
5831 {
5832   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #2 }
5833   \q_mark #1, \q_recursion_tail, \q_recursion_stop
5834 }
5835 \cs_new:Npn \clist_map_function_n_aux:Nn #1 #2
5836 {
5837   \quark_if_recursion_tail_stop:n {#2}
5838   \tl_if_empty:nF {#2} { \clist_map_aux_unbrace:Nw #1 #2, }
5839   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #1 }
5840   \q_mark
5841 }

```

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

`\clist_map_break:` Both are simple renaming.

```

\clist_map_break:n
5842 \cs_new_eq:NN \clist_map_break: \use_none_delimit_by_q_recursion_stop:w
5843 \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 111.)

187 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”.

```

\clist_show:c
\clist_show_aux:n
\clist_show_aux:w
5844 \cs_new_protected_nopar:Npn \clist_show:N #1
5845 {
5846   \clist_if_empty:NTF #1
5847   {
5848     \iow_term:x { Comma-list~\token_to_str:N #1-is-empty }
5849     \tl_show:n { }
5850   }
5851   {
5852     \iow_term:x
5853     {
5854       Comma-list~\token_to_str:N #1~
5855       contains~the~items~(without~outer~braces):
5856     }
5857     \tl_set:Nx \l_clist_show_tl
5858     { \clist_map_function:NN #1 \clist_show_aux:n }
5859     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
5860     { \exp_after:wN \clist_show_aux:w \l_clist_show_tl }
5861   }

```

```

5862 }
5863 \cs_new:Npn \clist_show_aux:n #1
5864 {
5865   \iow_newline: > \c_space_tl \c_space_tl
5866   \iow_char:N \{ \exp_not:n {#1} \iow_char:N \}
5867 }
5868 \cs_new:Npn \clist_show_aux:w #1 > ~ { }
5869 \cs_generate_variant:Nn \clist_show:N { c }

```

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

187.1 Scratch comma lists

`\l_tmpa_clist` Temporary comma list variables.

```

\l_tmpb_clist 5870 \clist_new:N \l_tmpa_clist
\g_tmpa_tl    5871 \clist_new:N \l_tmpb_clist
\g_tmpb_tl    5872 \clist_new:N \g_tmpa_clist
               5873 \clist_new:N \g_tmpb_clist

```

(End definition for `\l_tmpa_clist` and `\l_tmpb_clist`. These functions are documented on page 93.)

187.2 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).

```

5874 \cs_new:Npn \clist_length:N #1
5875 {
5876   \int_eval:n
5877   {
5878     0
5879     \clist_map_function:NN #1 \clist_length_aux:n
5880   }
5881 }
5882 \cs_new:Npn \clist_length_aux:n #1 { +1 }
5883 \cs_new:Npn \clist_length:n #1
5884 {
5885   \int_eval:n
5886   {
5887     0
5888     \clist_map_function_aux:Nw \clist_length_n_aux:n #1
5889     , \q_recursion_tail , \q_recursion_stop
5890   }
5891 }
5892 \cs_new:Npn \clist_length_n_aux:n #1 { \tl_if_blank:nF {#1} {+1} }
5893 \cs_generate_variant:Nn \clist_length:N { c }

```

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

`\clist_item:Nn` To avoid needing to test the end of the list at each step, we first compute the $\langle length \rangle$ of the list. If the item number is less than $-\langle length \rangle$ or more than $\langle length \rangle - 1$, the result is empty. If it is negative, but not less than $-\langle length \rangle$, add the $\langle length \rangle$ to the item number before performing the loop. The loop itself is very simple, return the item if the counter reached zero, otherwise, decrease the counter and repeat.

```

5894 \cs_new:Npn \clist_item:Nn #1#2
5895 {
5896   \exp_args:Nfo \clist_item_aux:nnNn
5897     { \clist_length:N #1 }
5898     #1
5899     \clist_item_N_loop:nw
5900     {#2}
5901 }
5902 \cs_new:Npn \clist_item_aux:nnNn #1#2#3#4
5903 {
5904   \int_compare:nNnTF {#4} < \c_zero
5905   {
5906     \int_compare:nNnTF {#4} < { - #1 }
5907     { \use_none_delimit_by_q_stop:w }
5908     { \exp_args:Nf #3 { \int_eval:n { #4 + #1 } } }
5909   }
5910   {
5911     \int_compare:nNnTF {#4} < {#1}
5912     { #3 {#4} }
5913     { \use_none_delimit_by_q_stop:w }
5914   }
5915   #2, \q_stop
5916 }
5917 \cs_new:Npn \clist_item_N_loop:nw #1 #2,
5918 {
5919   \int_compare:nNnTF {#1} = \c_zero
5920   { \use_i_delimit_by_q_stop:nw {#2} }
5921   { \exp_args:Nf \clist_item_N_loop:nw { \int_eval:n { #1 - 1 } } }
5922 }
5923 \cs_generate_variant:Nn \clist_item:Nn { c }

```

(End definition for `\clist_item:Nn` and `\clist_item:cn`. These functions are documented on page ??.)

`\clist_item:nn` This starts in the same way as `\clist_item:Nn` by checking the length of the comma list.
`\clist_item_n_aux:nw` The final item should be space-trimmed before being brace-stripped, hence we insert a
`\clist_item_n_loop:nw` couple of odd-looking `\prg_do_nothing:` to avoid losing braces. Blank items are ignored.

```

\clist_item_n_end:n
\clist_item_n_strip:w
5924 \cs_new:Npn \clist_item:nn #1#2
5925 {
5926   \exp_args:Nf \clist_item_aux:nnNn
5927     { \clist_length:n {#1} }
5928     {#1}

```

```

5929     \clist_item_n_aux:nw
5930     {#2}
5931   }
5932   \cs_new:Npn \clist_item_n_aux:nw #1
5933   { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
5934   \cs_new:Npn \clist_item_n_loop:nw #1 #2,
5935   {
5936     \exp_args:No \tl_if_blank:nTF {#2}
5937     { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
5938     {
5939       \int_compare:nNnTF {#1} = \c_zero
5940       { \exp_args:No \clist_item_n_end:n {#2} }
5941       {
5942         \exp_args:Nf \clist_item_n_loop:nw
5943         { \int_eval:n { #1 - 1 } }
5944         \prg_do_nothing:
5945       }
5946     }
5947   }
5948   \cs_new:Npn \clist_item_n_end:n #1 #2 \q_stop
5949   {
5950     \exp_after:wN \exp_after:wN \exp_after:wN \clist_item_n_strip:w
5951     \tl_trim_spaces:n {#1} ,
5952   }
5953   \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.

```

5954 \cs_new_protected:Npn \clist_set_from_seq:NN
5955 { \clist_set_from_seq_aux:NNNN \clist_clear:N \tl_set:Nx }
5956 \cs_new_protected:Npn \clist_gset_from_seq:NN
5957 { \clist_set_from_seq_aux:NNNN \clist_gclear:N \tl_gset:Nx }
5958 \cs_new_protected:Npn \clist_set_from_seq_aux:NNNN #1#2#3#4
5959 {
5960   \seq_if_empty:NTF #4
5961   { #1 #3 }
5962   {
5963     \seq_push_item_def:n
5964     {
5965       ,
5966       \tl_if_empty:oTF { \clist_set_from_seq_aux:w ##1 ~ , ##1 ~ }
5967       { \exp_not:n {##1} }
5968       { \exp_not:n { {##1} } }
5969     }
5970     #2 #3 { \exp_last_unbraced:Nf \use_none:n #4 }
5971     \seq_pop_item_def:

```

```

5972     }
5973   }
5974   \cs_new:Npn \clist_set_from_seq_aux:w #1 , #2 ~ { }
5975   \cs_generate_variant:Nn \clist_set_from_seq:NN { Nc }
5976   \cs_generate_variant:Nn \clist_set_from_seq:NN { c , cc }
5977   \cs_generate_variant:Nn \clist_gset_from_seq:NN { Nc }
5978   \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 ??.)

187.3 Deprecated interfaces

Deprecated on 2011-05-27, for removal by 2011-08-31.

\clist_top:NN These are old stack functions.

```

\clist_top:cN 5979 < *deprecated >
5980 \cs_new_eq:NN \clist_top:NN \clist_get:NN
5981 \cs_new_eq:NN \clist_top:cN \clist_get:cN
5982 < /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 5983 < *deprecated >
5984 \cs_new_eq:NN \clist_remove_element:Nn \clist_remove_all:Nn
5985 \cs_new_eq:NN \clist_gremove_element:Nn \clist_gremove_all:Nn
5986 < /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 5987 < *deprecated >
5988 \cs_new_eq:NN \clist_display:N \clist_show:N
5989 \cs_new_eq:NN \clist_display:c \clist_show:c
5990 < /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 5991 \cs_new_protected:Npn \clist_trim_spaces:N #1 { \clist_set:No #1 {#1} }
\clist_gtrim_spaces:N 5992 \cs_new_protected:Npn \clist_gtrim_spaces:N #1 { \clist_gset:No #1 {#1} }
\clist_gtrim_spaces:c 5993 \cs_generate_variant:Nn \clist_trim_spaces:N { c }
5994 \cs_generate_variant:Nn \clist_gtrim_spaces:N { c }

```

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

```

5995 < /initex | package >

```

188 l3prop implementation

The following test files are used for this code: *m3prop001*.

```

5996 <*initex | package>
5997 <*package>
5998 \ProvidesExplPackage
5999   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
6000 \package_check_loaded_expl:
6001 </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.

```

6002 \quark_new:N \q_prop
      (End definition for \q_prop. This function is documented on page 119.)

```

`\c_empty_prop` An empty prop contains exactly one `\q_prop`.

```

6003 \tl_const:Nn \c_empty_prop { \q_prop }
      (End definition for \c_empty_prop. This function is documented on page 119.)

```

188.1 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.

```

6004 \cs_new_protected:Npn \prop_new:N #1 { \cs_new_eq:NN #1 \c_empty_prop }
6005 \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 6006 \cs_new_protected:Npn \prop_clear:N #1 { \cs_set_eq:NN #1 \c_empty_prop }
\prop_gclear:N 6007 \cs_new_protected:Npn \prop_clear:c #1 { \cs_set_eq:cN {#1} \c_empty_prop }
\prop_gclear:c 6008 \cs_new_protected:Npn \prop_gclear:N #1 { \cs_gset_eq:NN #1 \c_empty_prop }
6009 \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 6010 \cs_new_protected:Npn \prop_clear_new:N #1
\prop_gclear_new:N 6011   { \cs_if_exist:NTF #1 { \prop_clear:N #1 } { \prop_new:N #1 } }
\prop_gclear_new:c 6012 \cs_generate_variant:Nn \prop_clear_new:N {c}
6013 \cs_new_protected:Npn \prop_gclear_new:N #1
6014   { \cs_if_exist:NTF #1 { \prop_gclear:N #1 } { \prop_new:N #1 } }
6015 \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` 6016 `\cs_new_eq:NN \prop_set_eq:NN \tl_set_eq:NN`
`\prop_set_eq:Nc` 6017 `\cs_new_eq:NN \prop_set_eq:Nc \tl_set_eq:Nc`
`\prop_set_eq:cc` 6018 `\cs_new_eq:NN \prop_set_eq:cN \tl_set_eq:cN`
`\prop_gset_eq:NN` 6019 `\cs_new_eq:NN \prop_set_eq:cc \tl_set_eq:cc`
`\prop_gset_eq:cN` 6020 `\cs_new_eq:NN \prop_gset_eq:NN \tl_gset_eq:NN`
`\prop_gset_eq:Nc` 6021 `\cs_new_eq:NN \prop_gset_eq:Nc \tl_gset_eq:Nc`
`\prop_gset_eq:cN` 6022 `\cs_new_eq:NN \prop_gset_eq:cN \tl_gset_eq:cN`
`\prop_gset_eq:cc` 6023 `\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 ??.)

188.2 Accessing data in property lists

`\prop_split:NnTF` This function is used by most of the module, and hence must be fast. The aim here is
`\prop_split_aux:NnTF` to split a property list at a given key into the part before the key–value pair, the value
`\prop_split_aux:nnnn` associated with the key and the part after the key–value pair. To do this, the key is first
`\prop_split_aux:w` detokenized (to avoid repeatedly doing this), then a delimited function is constructed
to match the key. It will match `\q_prop <detokenized key> \q_prop {<value>} <extra
argument>`, effectively separating an *<extract1>* before the key in the property list and an
<extract2> after the key.

If the key is present in the property list, then *<extra argument>* is simply `\q_prop`,
and `\prop_split_aux:nnnn` will gobble this and the false branch (#4), leaving the correct
code on the input stream. More precisely, it leaves the user code (true branch), followed
by three groups, `{<extract1>} {<value>} {<extract2>}`. In order for *<extract1>**<extract2>* to
be a well-formed property list, *<extract1>* has a leading and trailing `\q_prop`, retaining
exactly the structure of a property list, while *<extract2>* omits the leading `\q_prop`.

If the key is not there, then *<extra argument>* is `? \use_ii:nn { }`, and `\prop_split_aux:nnnn ? \u`
removes the three brace groups that just follow. Then `\use_ii:nn` removes the true
branch, leaving the false branch, with no trailing material.

```

6024 \cs_set_protected:Npn \prop_split:NnTF #1#2
6025 { \exp_args:NNo \prop_split_aux:NnTF #1 { \tl_to_str:n {#2} } }
6026 \cs_new_protected:Npn \prop_split_aux:NnTF #1#2
6027 {
6028   \cs_set_protected:Npn \prop_split_aux:w
6029     ##1 \q_prop #2 \q_prop ##2 ##3 ##4 \q_mark ##5 \q_stop
6030   { \prop_split_aux:nnnn ##3 { {##1 \q_prop } {##2} {##4} } }
6031   \exp_after:wN \prop_split_aux:w #1 \q_mark
6032     \q_prop #2 \q_prop { } { ? \use_ii:nn { } } \q_mark \q_stop
6033 }
6034 \cs_new:Npn \prop_split_aux:nnnn #1#2#3#4 { #3 #2 }
6035 \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, `{<extract1>} {<value>} {<extract2>}`. 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 $\backslash q_prop$.

```

6036 \cs_set_protected:Npn \prop_split:Nnn #1#2#3
6037 {
6038   \prop_split:NnTF #1 {#2}
6039   {#3}
6040   { \exp_args:Nno \use:n {#3} {#1} { \q_no_value } { } }
6041 }

```

(End definition for $\backslash prop_split:Nnn$. This function is documented on page 120.)

$\backslash prop_del:Nn$	Deleting from a property starts by splitting the list. If the key is present in the property
$\backslash prop_del:NV$	list, the returned value is ignored. If the key is missing, nothing happens.
$\backslash prop_del:cn$	
$\backslash prop_del:cV$	6042 \cs_new_protected:Npn \prop_del:Nn #1#2
$\backslash prop_gdel:Nn$	6043 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_set:Nn #1 } { } }
$\backslash prop_gdel:NV$	6044 \cs_new_protected:Npn \prop_gdel:Nn #1#2
$\backslash prop_gdel:cn$	6045 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_gset:Nn #1 } { } }
$\backslash prop_gdel:cV$	6046 \cs_new_protected:Npn \prop_del_aux:NNnnn #1#2#3#4#5
$\backslash prop_del_aux:NNnnn$	6047 { #1 #2 { #3 #5 } }
	6048 \cs_generate_variant:Nn \prop_del:Nn { NV }
	6049 \cs_generate_variant:Nn \prop_del:Nn { c , cV }
	6050 \cs_generate_variant:Nn \prop_gdel:Nn { NV }
	6051 \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 $\backslash q_no_value$.
$\backslash prop_get:NoN$	
$\backslash prop_get:cnN$	6052 \cs_new_protected:Npn \prop_get:NnN #1#2#3
$\backslash prop_get:cVN$	6053 {
$\backslash prop_get:NoN$	6054 \prop_split:NnTF #1 {#2}
$\backslash prop_get_aux:NNnn$	6055 { \prop_get_aux:Nnnn #3 }
	6056 { \tl_set:Nn #3 { \q_no_value } }
	6057 }
	6058 \cs_new_protected:Npn \prop_get_aux:NNnn #1#2#3#4
	6059 { \tl_set:Nn #1 {#3} }
	6060 \cs_generate_variant:Nn \prop_get:NnN { NV , No }
	6061 \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.
$\backslash prop_pop:coN$	
$\backslash prop_gpop:NnN$	6062 \cs_new_protected:Npn \prop_pop:NnN #1#2#3
$\backslash prop_gpop:NoN$	6063 {
$\backslash prop_gpop:cnN$	6064 \prop_split:NnTF #1 {#2}
$\backslash prop_gpop:coN$	6065 { \prop_pop_aux:NNnnnn \tl_set:Nn #1 #3 }
	6066 { \tl_set:Nn #3 { \q_no_value } }
$\backslash prop_pop_aux:NNnnnn$	6067 }


```

6068 \cs_new_protected:Npn \prop_gpop:NnN #1#2#3
6069 {
6070   \prop_split:NnTF #1 {#2}
6071   { \prop_pop_aux:NNNnnn \tl_gset:Nn #1 #3 }
6072   { \tl_set:Nn #3 { \q_no_value } }
6073 }
6074 \cs_new_protected:Npn \prop_pop_aux:NNNnnn #1#2#3#4#5#6
6075 {
6076   \tl_set:Nn #3 {#5}
6077   #1 #2 { #4 #6 }
6078 }
6079 \cs_generate_variant:Nn \prop_pop:NnN { No }
6080 \cs_generate_variant:Nn \prop_pop:NnN { c , co }
6081 \cs_generate_variant:Nn \prop_gpop:NnN { No }
6082 \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.

```

6083 \cs_new_protected:Npn \prop_put:Nnn { \prop_put_aux:NNnn \tl_set:Nx }
6084 \cs_new_protected:Npn \prop_gput:Nnn { \prop_put_aux:NNnn \tl_gset:Nx }
6085 \cs_new_protected:Npn \prop_put_aux:NNnn #1#2#3#4
6086 {
6087   \prop_split:Nnn #2 {#3} { \prop_put_aux:NNnnnnn #1 #2 {#3} {#4} }
6088 }
6089 \cs_new_protected:Npn \prop_put_aux:NNnnnnn #1#2#3#4#5#6#7
6090 {
6091   #1 #2
6092   {
6093     \exp_not:n { #5 #7 }
6094     \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop }
6095   }
6096 }
6097 \cs_generate_variant:Nn \prop_put:Nnn
6098 { NnV , Nno , Nnx , NV , NVV , No , Noo }
6099 \cs_generate_variant:Nn \prop_put:Nnn
6100 { c , cnV , cno , cnx , cV , cVV , co , coo }
6101 \cs_generate_variant:Nn \prop_gput:Nnn
6102 { NnV , Nno , Nnx , NV , NVV , No , Noo }
6103 \cs_generate_variant:Nn \prop_gput:Nnn
6104 { 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.

```

6105 \cs_new_protected_nopar:Npn \prop_put_if_new:Nnn
6106 { \prop_put_if_new_aux:NNnn \tl_put_right:Nx }

```

\prop_put_if_new:Nnn

```

6107 \cs_new_protected_nopar:Npn \prop_gput_if_new:Nnn
6108   { \prop_put_if_new_aux:NNnn \tl_gput_right:Nx }
6109 \cs_new_protected:Npn \prop_put_if_new_aux:NNnn #1#2#3#4
6110   {
6111     \prop_split:NnTF #2 {#3}
6112     { \use_none:nnn }
6113     {
6114       #1 #2
6115       { \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop } }
6116     }
6117   }
6118 \cs_generate_variant:Nn \prop_put_if_new:Nnn { c }
6119 \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 ??.)

188.3 Property list conditionals

\prop_if_empty:N The test here uses \c_empty_prop as it is not really empty!

```

\prop_if_empty:c
6120 \prg_new_conditional:Npnn \prop_if_empty:N #1 { p , T , F , TF }
6121   {
6122     \if_meaning:w #1 \c_empty_prop
6123     \prg_return_true:
6124     \else:
6125       \prg_return_false:
6126     \fi:
6127   }
6128 \cs_generate_variant:Nn \prop_if_empty_p:N {c}
6129 \cs_generate_variant:Nn \prop_if_empty:NnTF {c}
6130 \cs_generate_variant:Nn \prop_if_empty:NT {c}
6131 \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.

```

6132 \prg_new_conditional:Npnn \prop_if_in:Nn #1#2 { p , T , F , TF }
6133 {
6134   \exp_last_unbraced:Noo \prop_if_in_aux:nwn
6135   { \tl_to_str:n {#2} } #1
6136   \tl_to_str:n {#2} \q_prop { }
6137   \q_recursion_tail \q_recursion_stop
6138 }
6139 \cs_new:Npn \prop_if_in_aux:nwn #1 \q_prop #2 \q_prop #3
6140 {
6141   \str_if_eq:xxTF {#1} {#2}
6142   { \prop_if_in_aux:Nw }
6143   { \prop_if_in_aux:nwn {#1} }
6144 }
6145 \cs_new:Npn \prop_if_in_aux:Nw #1 #2 \q_recursion_stop
6146 {
6147   \if_meaning:w \q_prop #1
6148   \prg_return_true:
6149   \else:
6150   \prg_return_false:
6151   \fi:
6152 }
6153 \cs_generate_variant:Nn \prop_if_in_p:Nn { NV , No }
6154 \cs_generate_variant:Nn \prop_if_in_p:Nn { c , cV , co }
6155 \cs_generate_variant:Nn \prop_if_in:NnT { NV , No }
6156 \cs_generate_variant:Nn \prop_if_in:NnT { c , cV , co }
6157 \cs_generate_variant:Nn \prop_if_in:NnF { NV , No }
6158 \cs_generate_variant:Nn \prop_if_in:NnF { c , cV , co }
6159 \cs_generate_variant:Nn \prop_if_in:NnTF { NV , No }
6160 \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 ??.)

188.4 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>	6161 \prg_new_protected_conditional:Npnn \prop_get:NnN #1#2#3 { T , F , TF }
<code>\prop_get:cVN</code>	6162 {
<code>\prop_get:coN</code>	6163 \prop_split:NnTF #1 {#2}
<code>\prop_get_aux_true:Nnnn</code>	6164 { \prop_get_aux_true:Nnnn #3 }

```

6165     { \prg_return_false: }
6166   }
6167   \cs_new_protected:Npn \prop_get_aux_true:Nnnn #1#2#3#4
6168   {
6169     \tl_set:Nn #1 {#3}
6170     \prg_return_true:
6171   }
6172   \cs_generate_variant:Nn \prop_get:NnNT { NV , No }
6173   \cs_generate_variant:Nn \prop_get:NnNF { NV , No }
6174   \cs_generate_variant:Nn \prop_get:NnNTF { NV , No }
6175   \cs_generate_variant:Nn \prop_get:NnNT { c , cV , co }
6176   \cs_generate_variant:Nn \prop_get:NnNF { c , cV , co }
6177   \cs_generate_variant:Nn \prop_get:NnNTF { c , cV , co }

```

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

188.5 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

```

6178   \cs_new_nopar:Npn \prop_map_function:NN #1#2
6179   {
6180     \exp_last_unbraced:NNo \prop_map_function_aux:Nwn #2
6181     #1 \q_recursion_tail \q_prop { } \q_recursion_stop
6182   }
6183   \cs_new:Npn \prop_map_function_aux:Nwn #1 \q_prop #2 \q_prop #3
6184   {
6185     \if_meaning:w \q_recursion_tail #2
6186     \exp_after:wN \prop_map_break:
6187     \fi:
6188     #1 {#2} {#3}
6189     \prop_map_function_aux:Nwn #1
6190   }
6191   \cs_generate_variant:Nn \prop_map_function:NN { Nc }
6192   \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.

```

6193   \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

```

6194   \cs_new_protected:Npn \prop_map_inline:Nn #1#2
6195   {
6196     \int_gincr:N \g_prop_map_inline_int
6197     \cs_gset:cpn { prop_map_inline_ \int_use:N \g_prop_map_inline_int :nn }
6198     ##1##2 {#2}
6199     \prop_map_function:Nc #1
6200     { prop_map_inline_ \int_use:N \g_prop_map_inline_int :nn }

```

```

6201     \int_gdecr:N \g_prop_map_inline_int
6202   }
6203   \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.

```

6204   \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.

```

6205   \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 118.)

```

188.6 Viewing property lists

`\l_prop_show_tl` Used to store the material for display.

```

6206   \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 T_EX error if there is an empty property list, a simple test is used to keep the output “clean”.

```

\prop_show_aux:n
\prop_show_aux:w
6207   \cs_new_protected_nopar:Npn \prop_show:N #1
6208   {
6209     \prop_if_empty:NTF #1
6210     {
6211       \iow_term:x { Property-list~\token_to_str:N #1-is~empty }
6212       \tl_show:n { }
6213     }
6214     {
6215       \iow_term:x
6216       {
6217         Property-list~\token_to_str:N #1~
6218         contains~the~pairs~(without~outer~braces):
6219       }
6220       \tl_set:Nx \l_prop_show_tl
6221       { \prop_map_function:NN #1 \prop_show_aux:nn }
6222       \tl_show:n \exp_after:wN \exp_after:wN \exp_after:wN
6223       { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
6224     }
6225   }
6226   \cs_new:Npn \prop_show_aux:nn #1#2
6227   {
6228     \iow_newline: > \c_space_tl \c_space_tl
6229     \iow_char:N \{ #1 \iow_char:N \}
6230     \c_space_tl \c_space_tl => \c_space_tl \c_space_tl

```

```

6231     \iow_char:N \{ \exp_not:n {#2} \iow_char:N \}
6232   }
6233   \cs_new:Npn \prop_show_aux:w #1 > ~ { }
6234   \cs_generate_variant:Nn \prop_show:N { c }

```

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

188.7 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
6235   \prg_new_protected_conditional:Npnn \prop_pop:NnN #1#2#3 { T , F , TF }
6236   {
6237     \prop_split:NnTF #1 {#2}
6238     { \prop_pop_aux_true:NNNnnn \tl_set:Nn #1 #3 }
6239     { \prg_return_false: }
6240   }
6241   \prg_new_protected_conditional:Npnn \prop_gpop:NnN #1#2#3 { T , F , TF }
6242   {
6243     \prop_split:NnTF #1 {#2}
6244     { \prop_pop_aux_true:NNNnnn \tl_gset:Nn #1 #3 }
6245     { \prg_return_false: }
6246   }
6247   \cs_new_protected:Npn \prop_pop_aux_true:NNNnnn #1#2#3#4#5#6
6248   {
6249     \tl_set:Nn #3 {#5}
6250     #1 #2 { #4 #6 }
6251     \prg_return_true:
6252   }
6253   \cs_generate_variant:Nn \prop_pop:NnNT { c }
6254   \cs_generate_variant:Nn \prop_pop:NnNF { c }
6255   \cs_generate_variant:Nn \prop_pop:NnNTF { c }
6256   \cs_generate_variant:Nn \prop_gpop:NnNT { c }
6257   \cs_generate_variant:Nn \prop_gpop:NnNF { c }
6258   \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

```

6259   \cs_new:Npn \prop_map_tokens:Nn #1#2
6260   {
6261     \exp_last_unbraced:Nno \prop_map_tokens_aux:nwn {#2} #1
6262     \q_recursion_tail \q_prop { } \q_recursion_stop
6263   }
6264   \cs_new:Npn \prop_map_tokens_aux:nwn #1 \q_prop #2 \q_prop #3
6265   {
6266     \if_meaning:w \q_recursion_tail #2

```

```
6267 \exp_after:wN \prop_map_break:
```

```
6268 \fi:
```

```
6269 \use:n {#1} {#2} {#3}
```

```
6270 \prop_map_tokens_aux:nwn {#1}
```

```
6271 }
```

```
6272 \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.

```
6273 \cs_new:Npn \prop_get:Nn #1#2
```

```
6274 {
```

```
6275 \exp_last_unbraced:Noo \prop_get_Nn_aux:nwn
```

```
6276 { \tl_to_str:n {#2} } #1
```

```
6277 \tl_to_str:n {#2} \q_prop { }
```

```
6278 \q_recursion_stop
```

```
6279 }
```

```
6280 \cs_new:Npn \prop_get_Nn_aux:nwn #1 \q_prop #2 \q_prop #3
```

```
6281 {
```

```
6282 \str_if_eq:xxTF {#1} {#2}
```

```
6283 { \use_i_delimit_by_q_recursion_stop:nw {#3} }
```

```
6284 { \prop_get_Nn_aux:nwn {#1} }
```

```
6285 }
```

```
6286 \cs_generate_variant:Nn \prop_get:Nn { c }
```

(End definition for `\prop_get:Nn` and `\prop_get:cn`. These functions are documented on page ??.)

188.8 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`.

```
6287 \*deprecated
```

```
6288 \cs_new_eq:NN \prop_display:N \prop_show:N
```

```
6289 \cs_new_eq:NN \prop_display:c \prop_show:c
```

```
6290 \*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.

```
6291 \*deprecated
```

```
6292 \cs_new_protected:Npn \prop_gget:NnN #1#2#3
```

```
6293 { \prop_split:Nnn #1 {#2} { \prop_gget_aux:Nnnn #3 } }
```

```
6294 \cs_new_protected:Npn \prop_gget_aux:Nnnn #1#2#3#4
```

```

6295 { \tl_gset:Nn #1 {#3} }
6296 \cs_generate_variant:Nn \prop_gget:NnN { NV }
6297 \cs_generate_variant:Nn \prop_gget:NnN { c , cV }
6298 \</deprecat
(End definition for \prop_gget:NnN and others. These functions are documented on page ??.)

```

\prop_get_gdel:NnN This name seems very odd.

```

6299 \*deprecat
6300 \cs_new_eq:NN \prop_get_gdel:NnN \prop_gpop:NnN
6301 \</deprecat
(End definition for \prop_get_gdel:NnN. This function is documented on page ??.)

```

\prop_if_in:cc A hang-over from an ancient implementation

```

6302 \*deprecat
6303 \cs_generate_variant:Nn \prop_if_in:NnT { cc }
6304 \cs_generate_variant:Nn \prop_if_in:NnF { cc }
6305 \cs_generate_variant:Nn \prop_if_in:NnTF { cc }
6306 \</deprecat
(End definition for \prop_if_in:cc. This function is documented on page ??.)

```

\prop_gput:ccx Another one.

```

6307 \*deprecat
6308 \cs_generate_variant:Nn \prop_gput:Nnn { ccx }
6309 \</deprecat
(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
\prop_if_eq:cN
\prop_if_eq:cc
6310 \*deprecat
6311 \prg_new_eq_conditional:NNn \prop_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
6312 \prg_new_eq_conditional:NNn \prop_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
6313 \prg_new_eq_conditional:NNn \prop_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
6314 \prg_new_eq_conditional:NNn \prop_if_eq:cc \tl_if_eq:cc { p , T , F , TF }
6315 \</deprecat
(End definition for \prop_if_eq:NN and others. These functions are documented on page ??.)
6316 \</initex | package)

```

189 l3box implementation

```

6317 \*initex | package)
6318 \*package)
6319 \ProvidesExplPackage
6320 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
6321 \package_check_loaded_expl:
6322 \</package)

```


The code in this module is very straight forward so I'm not going to comment it very extensively.

189.1 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> 6323 <*package> 6324 \cs_new_protected:Npn \box_new:N #1 6325 { 6326 \chk_if_free_cs:N #1 6327 \newbox #1 6328 } 6329 </package> 6330 \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> 6331 \cs_new_protected_nopar:Npn \box_clear:N #1 6332 { \box_set_eq:NN #1 \c_empty_box } 6333 \cs_new_protected_nopar:Npn \box_gclear:N #1 6334 { \box_gset_eq:NN #1 \c_empty_box } 6335 \cs_generate_variant:Nn \box_clear:N { c } 6336 \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> 6337 \cs_new_protected_nopar:Npn \box_clear_new:N #1 6338 { 6339 \cs_if_exist:NTF #1 6340 { \box_set_eq:NN #1 \c_empty_box } 6341 { \box_new:N #1 } 6342 } 6343 \cs_new_protected_nopar:Npn \box_gclear_new:N #1 6344 { 6345 \cs_if_exist:NTF #1 6346 { \box_gset_eq:NN #1 \c_empty_box } 6347 { \box_new:N #1 } 6348 } 6349 \cs_generate_variant:Nn \box_clear_new:N { c } 6350 \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> 6351 \cs_new_protected_nopar:Npn \box_set_eq:NN #1#2 6352 { \tex_setbox:D #1 \tex_copy:D #2 } 6353 \cs_new_protected_nopar:Npn \box_gset_eq:NN 6354 { \tex_global:D \box_set_eq:NN } 6355 \cs_generate_variant:Nn \box_set_eq:NN { cN , Nc , cc } 6356 \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>	6357 <code>\cs_new_protected_nopar:Npn \box_set_eq_clear:NN #1#2</code>
<code>\box_set_eq_clear:cc</code>	6358 <code>{ \tex_setbox:D #1 \tex_box:D #2 }</code>
<code>\box_gset_eq_clear:NN</code>	6359 <code>\cs_new_protected_nopar:Npn \box_gset_eq_clear:NN</code>
<code>\box_gset_eq_clear:cN</code>	6360 <code>{ \tex_global:D \box_set_eq_clear:NN }</code>
<code>\box_gset_eq_clear:Nc</code>	6361 <code>\cs_generate_variant:Nn \box_set_eq_clear:NN { cN , Nc , cc }</code>
<code>\box_gset_eq_clear:cc</code>	6362 <code>\cs_generate_variant:Nn \box_gset_eq_clear:NN { cN , Nc , cc }</code>

189.2 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>	6363 <code>\cs_new_eq:NN \box_ht:N \tex_ht:D</code>
<code>\box_dp:N</code>	6364 <code>\cs_new_eq:NN \box_dp:N \tex_dp:D</code>
<code>\box_dp:c</code>	6365 <code>\cs_new_eq:NN \box_wd:N \tex_wd:D</code>
<code>\box_wd:N</code>	6366 <code>\cs_generate_variant:Nn \box_ht:N { c }</code>
<code>\box_wd:c</code>	6367 <code>\cs_generate_variant:Nn \box_dp:N { c }</code>
	6368 <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>	6369 <code>\cs_new_protected_nopar:Npn \box_set_dp:Nn #1#2</code>
<code>\box_set_dp:cn</code>	6370 <code>{ \box_dp:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
<code>\box_set_wd:Nn</code>	6371 <code>\cs_new_protected_nopar:Npn \box_set_ht:Nn #1#2</code>
<code>\box_set_wd:cn</code>	6372 <code>{ \box_ht:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6373 <code>\cs_new_protected_nopar:Npn \box_set_wd:Nn #1#2</code>
	6374 <code>{ \box_wd:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6375 <code>\cs_generate_variant:Nn \box_set_ht:Nn { c }</code>
	6376 <code>\cs_generate_variant:Nn \box_set_dp:Nn { c }</code>
	6377 <code>\cs_generate_variant:Nn \box_set_wd:Nn { c }</code>

189.3 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>	6378 <code>\cs_new_eq:NN \box_use_clear:N \tex_box:D</code>
<code>\box_use:N</code>	6379 <code>\cs_new_eq:NN \box_use:N \tex_copy:D</code>
<code>\box_use:c</code>	6380 <code>\cs_generate_variant:Nn \box_use_clear:N { c }</code>
	6381 <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>	6382 <code>\cs_new_protected:Npn \box_move_left:nn #1#2</code>
<code>\box_move_up:nn</code>	6383 <code>{ \tex_moveleft:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
<code>\box_move_down:nn</code>	6384 <code>\cs_new_protected:Npn \box_move_right:nn #1#2</code>
	6385 <code>{ \tex_moveright:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6386 <code>\cs_new_protected:Npn \box_move_up:nn #1#2</code>
	6387 <code>{ \tex_raise:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6388 <code>\cs_new_protected:Npn \box_move_down:nn #1#2</code>
	6389 <code>{ \tex_lower:D \dim_eval:w #1 \dim_eval_end: #2 }</code>

189.4 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` 6390 `\cs_new_eq:NN \if_hbox:N \tex_ifhbox:D`

`\if_box_empty:N` 6391 `\cs_new_eq:NN \if_vbox:N \tex_ifvbox:D`

6392 `\cs_new_eq:NN \if_box_empty:N \tex_ifvoid:D`

`\box_if_horizontal:N`

`\box_if_horizontal:c` 6393 `\prg_new_conditional:Npnn \box_if_horizontal:N #1 { p , T , F , TF }`

`\box_if_vertical:N` 6394 `{ \if_hbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

`\box_if_vertical:c` 6395 `\prg_new_conditional:Npnn \box_if_vertical:N #1 { p , T , F , TF }`

6396 `{ \if_vbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

6397 `\cs_generate_variant:Nn \box_if_horizontal_p:N { c }`

6398 `\cs_generate_variant:Nn \box_if_horizontal:NT { c }`

6399 `\cs_generate_variant:Nn \box_if_horizontal:NF { c }`

6400 `\cs_generate_variant:Nn \box_if_horizontal:NTF { c }`

6401 `\cs_generate_variant:Nn \box_if_vertical_p:N { c }`

6402 `\cs_generate_variant:Nn \box_if_vertical:NT { c }`

6403 `\cs_generate_variant:Nn \box_if_vertical:NF { c }`

6404 `\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` 6405 `\prg_new_conditional:Npnn \box_if_empty:N #1 { p , T , F , TF }`

6406 `{ \if_box_empty:N #1 \prg_return_true: \else: \prg_return_false: \fi: }`

6407 `\cs_generate_variant:Nn \box_if_empty_p:N { c }`

6408 `\cs_generate_variant:Nn \box_if_empty:NT { c }`

6409 `\cs_generate_variant:Nn \box_if_empty:NF { c }`

6410 `\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 ??.)

189.5 The last box inserted

`\l_last_box` A different name for this read-only primitive.

6411 `\cs_new_eq:NN \l_last_box \tex_lastbox:D`

(End definition for `\l_last_box`. This function is documented on page 125.)

`\box_set_to_last:N` Set a box to the previous box.

`\box_set_to_last:c` 6412 `\cs_new_protected_nopar:Npn \box_set_to_last:N #1`

`\box_gset_to_last:N` 6413 `{ \tex_setbox:D #1 \l_last_box }`

`\box_gset_to_last:c` 6414 `\cs_new_protected_nopar:Npn \box_gset_to_last:N`

6415 `{ \tex_global:D \box_set_to_last:N }`

6416 `\cs_generate_variant:Nn \box_set_to_last:N { c }`

6417 `\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 ??.)

189.6 Constant boxes

`\c_empty_box`

```
6418 <*package>
6419 \cs_new_eq:NN \c_empty_box \voidb@x
6420 </package>
6421 <*initex>
6422 \box_new:N \c_empty_box
6423 </initex>
(End definition for \c_empty_box. This function is documented on page 125.)
```

189.7 Scratch boxes

`\l_tmpa_box`

`\l_tmpb_box`

```
6424 <*package>
6425 \cs_new_eq:NN \l_tmpa_box \@tempboxa
6426 </package>
6427 <*initex>
6428 \box_new:N \l_tmpa_box
6429 </initex>
6430 \box_new:N \l_tmpb_box
(End definition for \l_tmpa_box and \l_tmpb_box. These functions are documented on page 125.)
```

189.8 Viewing box contents

`\box_show:N` Show the contents of a box and write it into the log file.

`\box_show:c`

```
6431 \cs_new_eq:NN \box_show:N \tex_showbox:D
6432 \cs_generate_variant:Nn \box_show:N { c }
(End definition for \box_show:N and \box_show:c. These functions are documented on page ??.)
```

189.9 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.

```
6433 \cs_new_protected_nopar:Npn \hbox:n { \tex_hbox:D \scan_stop: }
(End definition for \hbox:n. This function is documented on page 126.)
```

`\hbox_set:Nn`

`\hbox_set:cn`

`\hbox_gset:Nn`

`\hbox_gset:cn`

```
6434 \cs_new_protected:Npn \hbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_hbox:D {#2} }
6435 \cs_new_protected_nopar:Npn \hbox_gset:Nn { \tex_global:D \hbox_set:Nn }
6436 \cs_generate_variant:Nn \hbox_set:Nn { c }
6437 \cs_generate_variant:Nn \hbox_gset:Nn { c }
```

(End definition for `\hbox_set:Nn` and `\hbox_set:cn`. These functions are documented on page ??.)

<code>\hbox_set_to_wd:Nnn</code>	Storing material in a horizontal box with a specified width.
<code>\hbox_set_to_wd:cnn</code>	6438 <code>\cs_new_protected:Npn \hbox_set_to_wd:Nnn #1#2#3</code>
<code>\hbox_gset_to_wd:Nnn</code>	6439 <code>{ \tex_setbox:D #1 \tex_hbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }</code>
<code>\hbox_gset_to_wd:cnn</code>	6440 <code>\cs_new_protected_nopar:Npn \hbox_gset_to_wd:Nnn</code>
	6441 <code>{ \tex_global:D \hbox_set_to_wd:Nnn }</code>
	6442 <code>\cs_generate_variant:Nn \hbox_set_to_wd:Nnn { c }</code>
	6443 <code>\cs_generate_variant:Nn \hbox_gset_to_wd:Nnn { c }</code>
	<i>(End definition for \hbox_set_to_wd:Nnn and \hbox_set_to_wd:cnn. These functions are documented on page ??.)</i>
 <code>\hbox_set:Nw</code>	Storing material in a horizontal box. This type is useful in environment definitions.
<code>\hbox_set:cw</code>	6444 <code>\cs_new_protected_nopar:Npn \hbox_set:Nw #1</code>
<code>\hbox_gset:Nw</code>	6445 <code>{ \tex_setbox:D #1 \tex_hbox:D \c_group_begin_token }</code>
<code>\hbox_gset:cw</code>	6446 <code>\cs_new_protected_nopar:Npn \hbox_gset:Nw</code>
<code>\hbox_set_end:</code>	6447 <code>{ \tex_global:D \hbox_set:Nw }</code>
<code>\hbox_gset_end:</code>	6448 <code>\cs_generate_variant:Nn \hbox_set:Nw { c }</code>
	6449 <code>\cs_generate_variant:Nn \hbox_gset:Nw { c }</code>
	6450 <code>\cs_new_eq:NN \hbox_set_end: \c_group_end_token</code>
	6451 <code>\cs_new_eq:NN \hbox_gset_end: \c_group_end_token</code>
	<i>(End definition for \hbox_set:Nw and \hbox_set:cw. These functions are documented on page ??.)</i>
 <code>\hbox_set_inline_begin:N</code>	Renamed September 2011.
<code>\hbox_set_inline_begin:c</code>	6452 <code>\cs_new_eq:NN \hbox_set_inline_begin:N \hbox_set:Nw</code>
<code>\hbox_gset_inline_begin:N</code>	6453 <code>\cs_new_eq:NN \hbox_set_inline_begin:c \hbox_set:cw</code>
<code>\hbox_gset_inline_begin:c</code>	6454 <code>\cs_new_eq:NN \hbox_set_inline_end: \hbox_set_end:</code>
<code>\hbox_set_inline_end:</code>	6455 <code>\cs_new_eq:NN \hbox_gset_inline_begin:N \hbox_gset:Nw</code>
<code>\hbox_gset_inline_end:</code>	6456 <code>\cs_new_eq:NN \hbox_gset_inline_begin:c \hbox_gset:cw</code>
	6457 <code>\cs_new_eq:NN \hbox_gset_inline_end: \hbox_gset_end:</code>
	<i>(End definition for \hbox_set_inline_begin:N and \hbox_set_inline_begin:c. These functions are documented on page ??.)</i>
 <code>\hbox_to_wd:nn</code>	Put a horizontal box directly into the input stream.
<code>\hbox_to_zero:n</code>	6458 <code>\cs_new_protected:Npn \hbox_to_wd:nn #1#2</code>
	6459 <code>{ \tex_hbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }</code>
	6460 <code>\cs_new_protected:Npn \hbox_to_zero:n #1 { \tex_hbox:D to \c_zero_skip {#1} }</code>
	<i>(End definition for \hbox_to_wd:nn. This function is documented on page 126.)</i>
 <code>\hbox_overlap_left:n</code>	Put a zero-sized box with the contents pushed against one side (which makes it stick out
<code>\hbox_overlap_right:n</code>	on the other) directly into the input stream.
	6461 <code>\cs_new_protected:Npn \hbox_overlap_left:n #1</code>
	6462 <code>{ \hbox_to_zero:n { \tex_hss:D #1 } }</code>
	6463 <code>\cs_new_protected:Npn \hbox_overlap_right:n #1</code>
	6464 <code>{ \hbox_to_zero:n { #1 \tex_hss:D } }</code>
	<i>(End definition for \hbox_overlap_left:n. This function is documented on page 127.)</i>

`\hbox_unpack:N` Unpacking a box and if requested also clear it.

`\hbox_unpack:c` 6465 \cs_new_eq:NN \hbox_unpack:N \tex_unhcopy:D

`\hbox_unpack_clear:N` 6466 \cs_new_eq:NN \hbox_unpack_clear:N \tex_unhbox:D

`\hbox_unpack_clear:c` 6467 \cs_generate_variant:Nn \hbox_unpack:N { c }

6468 \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 ??.)

189.10 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.

6469 \cs_new_protected_nopar:Npn \vbox:n { \tex_vbox:D \scan_stop: }

6470 \cs_new_protected_nopar:Npn \vbox_top:n { \tex_vtop:D \scan_stop: }

(End definition for \vbox:n. This function is documented on page 128.)

`\vbox_to_ht:nn` Put a vertical box directly into the input stream.

`\vbox_to_zero:n` 6471 \cs_new_protected:Npn \vbox_to_ht:nn #1#2

`\vbox_to_ht:nn` 6472 { \tex_vbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }

`\vbox_to_zero:n` 6473 \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 128.)

`\vbox_set:Nn` Storing material in a vertical box with a natural height.

`\vbox_set:cn` 6474 \cs_new_protected:Npn \vbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_vbox:D {#2} }

`\vbox_gset:Nn` 6475 \cs_new_protected_nopar:Npn \vbox_gset:Nn { \tex_global:D \vbox_set:Nn }

`\vbox_gset:cn` 6476 \cs_generate_variant:Nn \vbox_set:Nn { c }

6477 \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` 6478 \cs_new_protected:Npn \vbox_set_top:Nn #1#2

`\vbox_gset_top:cn` 6479 { \tex_setbox:D #1 \tex_vtop:D {#2} }

6480 \cs_new_protected_nopar:Npn \vbox_gset_top:Nn

6481 { \tex_global:D \vbox_set_top:Nn }

6482 \cs_generate_variant:Nn \vbox_set_top:Nn { c }

6483 \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` 6484 `\cs_new_protected:Npn \vbox_set_to_ht:Nnn #1#2#3`

`\vbox_gset_to_ht:Nnn` 6485 `{ \tex_setbox:D #1 \tex_vbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }`

`\vbox_gset_to_ht:cnn` 6486 `\cs_new_protected_nopar:Npn \vbox_gset_to_ht:Nnn`
6487 `{ \tex_global:D \vbox_set_to_ht:Nnn }`
6488 `\cs_generate_variant:Nn \vbox_set_to_ht:Nnn { c }`
6489 `\cs_generate_variant:Nn \vbox_gset_to_ht:Nnn { c }`

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

`\vbox_set:Nw` Storing material in a vertical box. This type is useful in environment definitions.

`\vbox_set:cw` 6490 `\cs_new_nopar:Npn \vbox_set:Nw #1`

`\vbox_gset:Nw` 6491 `{ \tex_setbox:D #1 \tex_vbox:D \c_group_begin_token }`

`\vbox_gset:cw` 6492 `\cs_new_protected_nopar:Npn \vbox_gset:Nw`

`\vbox_set_end:` 6493 `{ \tex_global:D \vbox_set:Nw }`

`\vbox_gset_end:` 6494 `\cs_generate_variant:Nn \vbox_set:Nw { c }`
6495 `\cs_generate_variant:Nn \vbox_gset:Nw { c }`
6496 `\cs_new_eq:NN \vbox_set_end: \c_group_end_token`
6497 `\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` 6498 `\cs_new_eq:NN \vbox_set_inline_begin:N \vbox_set:Nw`

`\vbox_gset_inline_begin:N` 6499 `\cs_new_eq:NN \vbox_set_inline_begin:c \vbox_set:cw`

`\vbox_gset_inline_begin:c` 6500 `\cs_new_eq:NN \vbox_set_inline_end: \vbox_set_end:`

`\vbox_set_inline_end:` 6501 `\cs_new_eq:NN \vbox_gset_inline_begin:N \vbox_gset:Nw`

`\vbox_gset_inline_end:` 6502 `\cs_new_eq:NN \vbox_gset_inline_begin:c \vbox_gset:cw`
6503 `\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` 6504 `\cs_new_eq:NN \vbox_unpack:N \tex_unvcopy:D`

`\vbox_unpack_clear:N` 6505 `\cs_new_eq:NN \vbox_unpack_clear:N \tex_unvbox:D`

`\vbox_unpack_clear:c` 6506 `\cs_generate_variant:Nn \vbox_unpack:N { c }`
6507 `\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.

6508 `\cs_new_protected_nopar:Npn \vbox_set_split_to_ht:NNn #1#2#3`

6509 `{ \tex_setbox:D #1 \tex_vsplit:D #2 to \dim_eval:w #3 \dim_eval_end: }`

(End definition for `\vbox_set_split_to_ht:NNn`. This function is documented on page 129.)

189.11 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.

```
6510 \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 6511 \fp_new:N \l_box_cos_fp
6512 \fp_new:N \l_box_sin_fp
      (End definition for \l_box_cos_fp and \l_box_sin_fp. These functions are documented on page ??.)
```

`\l_box_top_dim` These are the positions of the four edges of a box before manipulation.

```
\l_box_bottom_dim 6513 \dim_new:N \l_box_top_dim
\l_box_left_dim 6514 \dim_new:N \l_box_bottom_dim
\l_box_right_dim 6515 \dim_new:N \l_box_left_dim
6516 \dim_new:N \l_box_right_dim
      (End definition for \l_box_top_dim and others. These functions are documented on page ??.)
```

`\l_box_top_new_dim` These are the positions of the four edges of a box after manipulation.

```
\l_box_bottom_new_dim 6517 \dim_new:N \l_box_top_new_dim
\l_box_left_new_dim 6518 \dim_new:N \l_box_bottom_new_dim
\l_box_right_new_dim 6519 \dim_new:N \l_box_left_new_dim
6520 \dim_new:N \l_box_right_new_dim
      (End definition for \l_box_top_new_dim and others. These functions are documented on page ??.)
```

`\l_box_tmp_box` Scratch space.

```
\l_box_tmp_fp 6521 \box_new:N \l_box_tmp_box
6522 \fp_new:N \l_box_tmp_fp
      (End definition for \l_box_tmp_box and \l_box_tmp_fp. These functions are documented on page ??.)
```

`\l_box_x_fp` Used as the input and output values for a point when manipulation the location.

```
\l_box_y_fp 6523 \fp_new:N \l_box_x_fp
\l_box_x_new_fp 6524 \fp_new:N \l_box_y_fp
\l_box_y_new_fp 6525 \fp_new:N \l_box_x_new_fp
6526 \fp_new:N \l_box_y_new_fp
      (End definition for \l_box_x_fp and others. These functions are documented on page ??.)
```

`\box_rotate:Nn` Rotation of a box starts with working out the relevant sine and cosine. There is then a check to avoid doing any real work for the trivial rotation.

```
\box_rotate_aux:N 6527 \cs_new_protected_nopar:Npn \box_rotate:Nn #1#2
\box_rotate_set_sin_cos: 6528 {
\box_rotate_x:nnN 6529 \hbox_set:Nn #1
\box_rotate_y:nnN 6530 {
\box_rotate_quadrant_one: 6531 \group_begin:
\box_rotate_quadrant_two: 6532 \fp_set:Nn \l_box_angle_fp {#2}
\box_rotate_quadrant_three:
\box_rotate_quadrant_four:
```

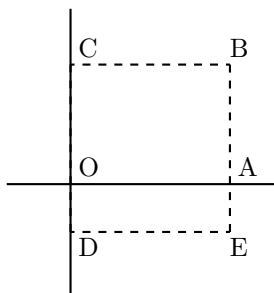



Figure 1: Co-ordinates of a box prior to rotation.

```

6533     \box_rotate_set_sin_cos:
6534     \fp_compare:NNTF \l_box_sin_fp = \c_zero_fp
6535     {
6536         \fp_compare:NNTF \l_box_cos_fp = \c_one_fp
6537         { \box_use:N #1 }
6538         { \box_rotate_aux:N #1 }
6539     }
6540     { \box_rotate_aux:N #1 }
6541 \group_end:
6542 }
6543 }

```

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.

```

6544 \cs_new_protected_nopar:Npn \box_rotate_aux:N #1
6545 {
6546     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6547     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6548     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6549     \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.)

```

6550 \fp_compare:NNTF \l_box_sin_fp > \c_zero_fp
6551 {
6552   \fp_compare:NNTF \l_box_cos_fp > \c_zero_fp
6553   { \box_rotate_quadrant_one: }
6554   { \box_rotate_quadrant_two: }
6555 }
6556 {
6557   \fp_compare:NNTF \l_box_cos_fp < \c_zero_fp
6558   { \box_rotate_quadrant_three: }
6559   { \box_rotate_quadrant_four: }
6560 }

```

The position of the box edges are now known, but the box at this stage be misplaced relative to the current \TeX reference point. So the content of the box is moved such that the reference point of the rotated box will be in the same place as the original.

```

6561 \hbox_set:Nn \l_box_tmp_box { \box_use:N #1 }
6562 \hbox_set:Nn \l_box_tmp_box
6563 {
6564   \tex_kern:D -\l_box_left_new_dim
6565   \hbox:n
6566   {
6567     \driver_box_rotate_begin:
6568     \box_use:N \l_box_tmp_box
6569     \driver_box_rotate_end:
6570   }
6571 }

```

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.

```

6572 \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6573 \box_set_dp:Nn \l_box_tmp_box { -\l_box_bottom_new_dim }
6574 \box_set_wd:Nn \l_box_tmp_box
6575 { \l_box_right_new_dim - \l_box_left_new_dim }
6576 \box_use:N \l_box_tmp_box
6577 }

```

A simple conversion from degrees to radians followed by calculation of the sine and cosine.

```

6578 \cs_new_protected_nopar:Npn \box_rotate_set_sin_cos:
6579 {
6580   \fp_set_eq:NN \l_box_tmp_fp \l_box_angle_fp
6581   \fp_div:Nn \l_box_tmp_fp { 180 }
6582   \fp_mul:Nn \l_box_tmp_fp { \c_pi_fp }
6583   \fp_sin:Nn \l_box_sin_fp { \l_box_tmp_fp }
6584   \fp_cos:Nn \l_box_cos_fp { \l_box_tmp_fp }
6585 }

```

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.

```

6586 \cs_new_protected_nopar:Npn \box_rotate_x:nnN #1#2#3
6587 {
6588   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6589   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6590   \fp_set_eq:NN \l_box_x_new_fp \l_box_x_fp
6591   \fp_set_eq:NN \l_box_tmp_fp \l_box_y_fp
6592   \fp_mul:Nn \l_box_x_new_fp { \l_box_cos_fp }
6593   \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }
6594   \fp_sub:Nn \l_box_x_new_fp { \l_box_tmp_fp }
6595   \dim_set:Nn #3 { \fp_to_dim:N \l_box_x_new_fp }
6596 }
6597 \cs_new_protected_nopar:Npn \box_rotate_y:nnN #1#2#3
6598 {
6599   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6600   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6601   \fp_set_eq:NN \l_box_y_new_fp \l_box_y_fp
6602   \fp_set_eq:NN \l_box_tmp_fp \l_box_x_fp
6603   \fp_mul:Nn \l_box_y_new_fp { \l_box_cos_fp }
6604   \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }
6605   \fp_add:Nn \l_box_y_new_fp { \l_box_tmp_fp }
6606   \dim_set:Nn #3 { \fp_to_dim:N \l_box_y_new_fp }
6607 }

```

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.

```

6608 \cs_new_protected_nopar:Npn \box_rotate_quadrant_one:
6609 {
6610   \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6611   \l_box_top_new_dim
6612   \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6613   \l_box_bottom_new_dim
6614   \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6615   \l_box_left_new_dim
6616   \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6617   \l_box_right_new_dim
6618 }
6619 \cs_new_protected_nopar:Npn \box_rotate_quadrant_two:
6620 {
6621   \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6622   \l_box_top_new_dim
6623   \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6624   \l_box_bottom_new_dim
6625   \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim

```

```

6626     \l_box_left_new_dim
6627     \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6628     \l_box_right_new_dim
6629 }
6630 \cs_new_protected_nopar:Npn \box_rotate_quadrant_three:
6631 {
6632     \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6633     \l_box_top_new_dim
6634     \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6635     \l_box_bottom_new_dim
6636     \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6637     \l_box_left_new_dim
6638     \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6639     \l_box_right_new_dim
6640 }
6641 \cs_new_protected_nopar:Npn \box_rotate_quadrant_four:
6642 {
6643     \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6644     \l_box_top_new_dim
6645     \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6646     \l_box_bottom_new_dim
6647     \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6648     \l_box_left_new_dim
6649     \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6650     \l_box_right_new_dim
6651 }

```

(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 6652 \fp_new:N \l_box_scale_x_fp
6653 \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 6654 \cs_new_protected:Npn \box_resize:Nnn #1#2#3
\box_resize_aux:Nnn 6655 {
6656     \hbox_set:Nn #1
6657     {
6658         \group_begin:
6659         \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6660         \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6661         \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6662         \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.

```

6663         \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6664         \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6665         \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.

```

6666         \fp_set_from_dim:Nn \l_box_scale_y_fp {#3}
6667         \fp_set_from_dim:Nn \l_box_tmp_fp
6668             { \l_box_top_dim - \l_box_bottom_dim }
6669         \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.

```

6670         \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6671         {
6672             \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6673             { \box_use:N #1 }
6674             { \box_resize_aux:Nnn #1 {#2} {#3} }
6675         }
6676         { \box_resize_aux:Nnn #1 {#2} {#3} }
6677     \group_end:
6678 }
6679 }
6680 \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.

```

6681 \cs_new_protected:Npn \box_resize_aux:Nnn #1#2#3
6682 {
6683     \dim_compare:nNnTF {#2} > \c_zero_dim
6684     { \dim_set:Nn \l_box_right_new_dim {#2} }
6685     { \dim_set:Nn \l_box_right_new_dim { \c_zero_dim - ( #2 ) } }
6686     \dim_compare:nNnTF {#3} > \c_zero_dim
6687     {
6688         \dim_set:Nn \l_box_top_new_dim
6689         { \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6690         \dim_set:Nn \l_box_bottom_new_dim
6691         { \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6692     }
6693     {
6694         \dim_set:Nn \l_box_top_new_dim
6695         { - \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6696         \dim_set:Nn \l_box_bottom_new_dim
6697         { - \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6698     }
6699     \box_resize_common:N #1
6700 }

```

(End definition for \box_resize:Nnn and \box_resize:cnn. These functions are documented on page ??.)

```

\box_resize_to_ht_plus_dp:Nn
\box_resize_to_ht_plus_dp:cn
    \box_resize_to_wd:Nn
    \box_resize_to_wd:cn

```

Scaling to a total height or to a width is a simplified version of the main resizing operation, with the scale simply copied between the two parts. The internal auxiliary is called using

the scaling value twice, as the sign for both parts is needed (as this allows the same internal code to be used as for the general case).

```

6701 \cs_new_protected_nopar:Npn \box_resize_to_ht_plus_dp:Nn #1#2
6702 {
6703   \hbox_set:Nn #1
6704   {
6705     \group_begin:
6706       \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6707       \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6708       \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6709       \dim_zero:N \l_box_left_dim
6710       \fp_set_from_dim:Nn \l_box_scale_y_fp {#2}
6711       \fp_set_from_dim:Nn \l_box_tmp_fp
6712         { \l_box_top_dim - \l_box_bottom_dim }
6713       \fp_div:Nn \l_box_scale_y_fp { \l_box_tmp_fp }
6714       \fp_set_eq:NN \l_box_scale_x_fp \l_box_scale_y_fp
6715       \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6716         { \box_use:N #1 }
6717         { \box_resize_aux:Nnn #1 {#2} {#2} }
6718     \group_end:
6719   }
6720 }
6721 \cs_generate_variant:Nn \box_resize_to_ht_plus_dp:Nn { c }
6722 \cs_new_protected_nopar:Npn \box_resize_to_wd:Nn #1#2
6723 {
6724   \hbox_set:Nn #1
6725   {
6726     \group_begin:
6727       \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6728       \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6729       \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6730       \dim_zero:N \l_box_left_dim
6731       \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6732       \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6733       \fp_div:Nn \l_box_scale_x_fp { \l_box_tmp_fp }
6734       \fp_set_eq:NN \l_box_scale_y_fp \l_box_scale_x_fp
6735       \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6736         { \box_use:N #1 }
6737         { \box_resize_aux:Nnn #1 {#2} {#2} }
6738     \group_end:
6739   }
6740 }
6741 \cs_generate_variant:Nn \box_resize_to_wd:Nn { c }

```

(End definition for \box_resize_to_ht_plus_dp:Nn and \box_resize_to_ht_plus_dp:cn. These functions are documented on page ??.)

\box_scale:Nnn	When scaling a box, setting the scaling itself is easy enough. The new dimensions are
\box_scale:cnn	also relatively easy to find, allowing only for the need to keep them positive in all cases.
\box_scale_aux:Nnn	Once that is done then after a check for the trivial scaling a hand-off can be made

to the common code. The dimension scaling operations are carried out using the \TeX mechanism as it avoids needing to use fp operations.

```

6742 \cs_new_protected_nopar:Npn \box_scale:Nnn #1#2#3
6743 {
6744   \hbox_set:Nn #1
6745   {
6746     \group_begin:
6747       \fp_set:Nn \l_box_scale_x_fp {#2}
6748       \fp_set:Nn \l_box_scale_y_fp {#3}
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_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6754       {
6755         \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6756         { \box_use:N #1 }
6757         { \box_scale_aux:Nnn #1 {#2} {#3} }
6758       }
6759       { \box_scale_aux:Nnn #1 {#2} {#3} }
6760     \group_end:
6761   }
6762 }
6763 \cs_generate_variant:Nn \box_scale:Nnn { c }
6764 \cs_new_protected_nopar:Npn \box_scale_aux:Nnn #1#2#3
6765 {
6766   \fp_compare:NNTF \l_box_scale_y_fp > \c_zero_fp
6767   {
6768     \dim_set:Nn \l_box_top_new_dim { #3 \l_box_top_dim }
6769     \dim_set:Nn \l_box_bottom_new_dim { #3 \l_box_bottom_dim }
6770   }
6771   {
6772     \dim_set:Nn \l_box_top_new_dim { -#3 \l_box_bottom_dim }
6773     \dim_set:Nn \l_box_bottom_new_dim { -#3 \l_box_top_dim }
6774   }
6775   \fp_compare:NNTF \l_box_scale_x_fp > \c_zero_fp
6776   { \l_box_right_new_dim #2 \l_box_right_dim }
6777   { \l_box_right_new_dim -#2 \l_box_right_dim }
6778   \box_resize_common:N #1
6779 }

```

(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.

```

6780 \cs_new_protected_nopar:Npn \box_resize_common:N #1
6781 {
6782   \hbox_set:Nn \l_box_tmp_box
6783   {

```

```

6784         \driver_box_scale_begin:
6785         \hbox_overlap_right:n { \box_use:N #1 }
6786         \driver_box_scale_end:
6787     }

```

The new height and depth can be applied directly.

```

6788     \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6789     \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.

```

6790     \fp_compare:NNNTF \l_box_scale_x_fp < \c_zero_fp
6791     {
6792         \hbox_to_wd:nn { \l_box_right_new_dim }
6793         {
6794             \tex_kern:D \l_box_right_new_dim
6795             \box_use:N \l_box_tmp_box
6796             \tex_hss:D
6797         }
6798     }
6799     {
6800         \box_set_wd:Nn \l_box_tmp_box { \l_box_right_new_dim }
6801         \box_use:N \l_box_tmp_box
6802     }
6803 }

```

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

```

6804 </initex | package>

```

190 l3coffins Implementation

```

6805 <*initex | package>
6806 <*package>
6807 \ProvidesExplPackage
6808   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
6809 \package_check_loaded_expl:
6810 </package>

```

190.1 Coffins: data structures and general variables

\l_coffin_tmp_box Scratch variables.

```

\l_coffin_tmp_dim 6811 \box_new:N \l_coffin_tmp_box
\l_coffin_tmp_fp   6812 \dim_new:N \l_coffin_tmp_dim
\l_coffin_tmp_tl   6813 \fp_new:N \l_coffin_tmp_fp
                   6814 \tl_new:N \l_coffin_tmp_tl

```

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

<code>\c_coffin_corners_prop</code>	<p>The “corners”; of a coffin define the real content, as opposed to the \TeX bounding box. They all start off in the same place, of course.</p> <pre> 6815 \prop_new:N \c_coffin_corners_prop 6816 \prop_put:Nnn \c_coffin_corners_prop { tl } { { 0 pt } { 0 pt } } 6817 \prop_put:Nnn \c_coffin_corners_prop { tr } { { 0 pt } { 0 pt } } 6818 \prop_put:Nnn \c_coffin_corners_prop { bl } { { 0 pt } { 0 pt } } 6819 \prop_put:Nnn \c_coffin_corners_prop { br } { { 0 pt } { 0 pt } } </pre> <p><i>(End definition for \c_coffin_corners_prop. This function is documented on page ??.)</i></p>
<code>\c_coffin_poles_prop</code>	<p>Pole positions are given for horizontal, vertical and reference-point based values.</p> <pre> 6820 \prop_new:N \c_coffin_poles_prop 6821 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } } 6822 \prop_put:Nno \c_coffin_poles_prop { l } { \l_coffin_tmp_tl } 6823 \prop_put:Nno \c_coffin_poles_prop { hc } { \l_coffin_tmp_tl } 6824 \prop_put:Nno \c_coffin_poles_prop { r } { \l_coffin_tmp_tl } 6825 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 1000 pt } { 0 pt } } 6826 \prop_put:Nno \c_coffin_poles_prop { b } { \l_coffin_tmp_tl } 6827 \prop_put:Nno \c_coffin_poles_prop { vc } { \l_coffin_tmp_tl } 6828 \prop_put:Nno \c_coffin_poles_prop { t } { \l_coffin_tmp_tl } 6829 \prop_put:Nno \c_coffin_poles_prop { B } { \l_coffin_tmp_tl } 6830 \prop_put:Nno \c_coffin_poles_prop { H } { \l_coffin_tmp_tl } 6831 \prop_put:Nno \c_coffin_poles_prop { T } { \l_coffin_tmp_tl } </pre> <p><i>(End definition for \c_coffin_poles_prop. This function is documented on page ??.)</i></p>
<code>\l_coffin_calc_a_fp</code> <code>\l_coffin_calc_b_fp</code> <code>\l_coffin_calc_c_fp</code> <code>\l_coffin_calc_d_fp</code> <code>\l_coffin_calc_result_fp</code>	<p>Used for calculations of intersections and in other internal places.</p> <pre> 6832 \fp_new:N \l_coffin_calc_a_fp 6833 \fp_new:N \l_coffin_calc_b_fp 6834 \fp_new:N \l_coffin_calc_c_fp 6835 \fp_new:N \l_coffin_calc_d_fp 6836 \fp_new:N \l_coffin_calc_result_fp </pre> <p><i>(End definition for \l_coffin_calc_a_fp. This function is documented on page ??.)</i></p>
<code>\l_coffin_error_bool</code>	<p>For propagating errors so that parts of the code can work around them.</p> <pre> 6837 \bool_new:N \l_coffin_error_bool </pre> <p><i>(End definition for \l_coffin_error_bool. This function is documented on page ??.)</i></p>
<code>\l_coffin_offset_x_dim</code> <code>\l_coffin_offset_y_dim</code>	<p>The offset between two sets of coffin handles when typesetting. These values are corrected from those requested in an alignment for the positions of the handles.</p> <pre> 6838 \dim_new:N \l_coffin_offset_x_dim 6839 \dim_new:N \l_coffin_offset_y_dim </pre> <p><i>(End definition for \l_coffin_offset_x_dim. This function is documented on page ??.)</i></p>
<code>\l_coffin_pole_a_tl</code> <code>\l_coffin_pole_b_tl</code>	<p>Needed for finding the intersection of two poles.</p> <pre> 6840 \tl_new:N \l_coffin_pole_a_tl 6841 \tl_new:N \l_coffin_pole_b_tl </pre> <p><i>(End definition for \l_coffin_pole_a_tl. This function is documented on page ??.)</i></p>

<code>\l_coffin_sin_fp</code> <code>\l_coffin_cos_fp</code>	<p>Used for rotations to get the sine and cosine values.</p> <pre> 6842 \fp_new:N \l_coffin_sin_fp 6843 \fp_new:N \l_coffin_cos_fp (End definition for \l_coffin_sin_fp. This function is documented on page ??.) </pre>
<code>\l_coffin_x_dim</code> <code>\l_coffin_y_dim</code> <code>\l_coffin_x_prime_dim</code> <code>\l_coffin_y_prime_dim</code>	<p>For calculating intersections and so forth.</p> <pre> 6844 \dim_new:N \l_coffin_x_dim 6845 \dim_new:N \l_coffin_y_dim 6846 \dim_new:N \l_coffin_x_prime_dim 6847 \dim_new:N \l_coffin_y_prime_dim (End definition for \l_coffin_x_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_x_fp</code> <code>\l_coffin_y_fp</code> <code>\l_coffin_x_prime_fp</code> <code>\l_coffin_y_prime_fp</code>	<p>Used for calculations where there are clear x- and y-components, for example during vector rotation.</p> <pre> 6848 \fp_new:N \l_coffin_x_fp 6849 \fp_new:N \l_coffin_y_fp 6850 \fp_new:N \l_coffin_x_prime_fp 6851 \fp_new:N \l_coffin_y_prime_fp (End definition for \l_coffin_x_fp. This function is documented on page ??.) </pre>
<code>\l_coffin_Depth_dim</code> <code>\l_coffin_Height_dim</code> <code>\l_coffin_TotalHeight_dim</code> <code>\l_coffin_Width_dim</code>	<p>Dimensions for the various parts of a coffin.</p> <pre> 6852 \dim_new:N \l_coffin_Depth_dim 6853 \dim_new:N \l_coffin_Height_dim 6854 \dim_new:N \l_coffin_TotalHeight_dim 6855 \dim_new:N \l_coffin_Width_dim (End definition for \l_coffin_Depth_dim. This function is documented on page ??.) </pre>
<code>\coffin_saved_Depth:</code> <code>\coffin_saved_Height:</code> <code>\coffin_saved_TotalHeight:</code> <code>\coffin_saved_Width:</code>	<p>Used to save the meaning of <code>\Depth</code>, <code>\Height</code>, <code>\TotalHeight</code> and <code>\Width</code>.</p> <pre> 6856 \cs_new_nopar:Npn \coffin_saved_Depth: { } 6857 \cs_new_nopar:Npn \coffin_saved_Height: { } 6858 \cs_new_nopar:Npn \coffin_saved_TotalHeight: { } 6859 \cs_new_nopar:Npn \coffin_saved_Width: { } (End definition for \coffin_saved_Depth:. This function is documented on page ??.) </pre>

190.2 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.

```

6860 \cs_new_protected:Npn \coffin_if_exist:NT #1#2
6861 {
6862   \cs_if_exist:NTF #1
6863   {
6864     \cs_if_exist:CTF { l_coffin_poles_ \int_value:w #1 _prop }
6865     { #2 }

```

```

6866         {
6867             \msg_kernel_error:nxx { coffins } { unknown-coffin }
6868             { \token_to_str:N #1 }
6869         }
6870     }
6871     {
6872         \msg_kernel_error:nxx { coffins } { unknown-coffin }
6873         { \token_to_str:N #1 }
6874     }
6875 }

```

(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
6876 \cs_new_protected_nopar:Npn \coffin_clear:N #1
6877 {
6878     \coffin_if_exist:NT #1
6879     {
6880         \box_clear:N #1
6881         \coffin_reset_structure:N #1
6882     }
6883 }
6884 \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.

```

6885 \cs_new_protected_nopar:Npn \coffin_new:N #1
6886 {
6887     \box_new:N #1
6888     \prop_clear_new:c { l_coffin_corners_ \int_value:w #1 _prop }
6889     \prop_clear_new:c { l_coffin_poles_ \int_value:w #1 _prop }
6890     \prop_gset_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
6891         \c_coffin_corners_prop
6892     \prop_gset_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
6893         \c_coffin_poles_prop
6894 }
6895 \cs_generate_variant:Nn \coffin_new:N { c }

```

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

\hcoffin_set:Nn Horizontal coffins are relatively easy: set the appropriate box, reset the structures then update the handle positions.

```

6896 \cs_new_protected:Npn \hcoffin_set:Nn #1#2
6897 {
6898     \coffin_if_exist:NT #1
6899     {

```

```

6900     \hbox_set:Nn #1
6901     {
6902         \color_group_begin:
6903         \color_ensure_current:
6904         #2
6905         \color_group_end:
6906     }
6907     \coffin_reset_structure:N #1
6908     \coffin_update_poles:N #1
6909     \coffin_update_corners:N #1
6910 }
6911 }
6912 \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.

```

6913 \cs_new_protected:Npn \vcoffin_set:Nnn #1#2#3
6914 {
6915     \coffin_if_exist:NT #1
6916     {
6917         \vbox_set:Nn #1
6918         {
6919             \dim_set:Nn \tex_hsize:D {#2}
6920             \color_group_begin:
6921             \color_ensure_current:
6922             #3
6923             \color_group_end:
6924         }
6925         \coffin_reset_structure:N #1
6926         \coffin_update_poles:N #1
6927         \coffin_update_corners:N #1
6928         \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
6929         \coffin_set_pole:Nnx #1 { T }
6930         {
6931             { 0 pt }
6932             { \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box } }
6933             { 1000 pt }
6934             { 0 pt }
6935         }
6936         \box_clear:N \l_coffin_tmp_box
6937     }
6938 }
6939 \cs_generate_variant:Nn \vcoffin_set:Nnn { c }

```

(End definition for \vcoffin_set:Nnn and \vcoffin_set:cn. These functions are documented on page ??.)

`\hcoffin_set:Nw` These are the “begin”/“end” versions of the above: watch the grouping!

```

\hcoffin_set:cnw 6940 \cs_new_protected_nopar:Npn \hcoffin_set:Nw #1
\hcoffin_set_end: 6941 {
                  6942   \coffin_if_exist:NT #1
                  6943   {
                  6944     \hbox_set:Nw #1 \color_group_begin: \color_ensure_current:
                  6945     \cs_set_protected_nopar:Npn \hcoffin_set_end:
                  6946     {
                  6947       \color_group_end:
                  6948       \hbox_set_end:
                  6949       \coffin_reset_structure:N #1
                  6950       \coffin_update_poles:N #1
                  6951       \coffin_update_corners:N #1
                  6952     }
                  6953   }
                  6954 }
6955 \cs_new_protected_nopar:Npn \hcoffin_set_end: { }
6956 \cs_generate_variant:Nn \hcoffin_set:Nw { c }

```

(End definition for \hcoffin_set:Nw and \hcoffin_set:cnw. These functions are documented on page ??.)

`\vcoffin_set:Nnw` The same for vertical coffins.

```

\vcoffin_set:cnw 6957 \cs_new_protected_nopar:Npn \vcoffin_set:Nnw #1#2
\vcoffin_set_end: 6958 {
                  6959   \coffin_if_exist:NT #1
                  6960   {
                  6961     \vbox_set:Nw #1
                  6962     \dim_set:Nn \tex_hsize:D {#2}
                  6963     \color_group_begin: \color_ensure_current:
                  6964     \cs_set_protected:Npn \vcoffin_set_end:
                  6965     {
                  6966       \color_group_end:
                  6967       \vbox_set_end:
                  6968       \coffin_reset_structure:N #1
                  6969       \coffin_update_poles:N #1
                  6970       \coffin_update_corners:N #1
                  6971       \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
                  6972       \coffin_set_pole:Nnx #1 { T }
                  6973       {
                  6974         { 0 pt }
                  6975         {
                  6976           \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box }
                  6977         }
                  6978         { 1000 pt }
                  6979         { 0 pt }
                  6980       }
                  6981       \box_clear:N \l_coffin_tmp_box
                  6982     }
                  6983   }

```

```

6984     }
6985     \cs_new_protected_nopar:Npn \vcoffin_set_end: { }
6986     \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 6987 \cs_new_protected_nopar:Npn \coffin_set_eq:NN #1#2
\coffin_set_eq:cN 6988 {
\coffin_set_eq:cc 6989     \coffin_if_exist:NT #1
6990     {
6991         \box_set_eq:NN #1 #2
6992         \coffin_set_eq_structure:NN #1 #2
6993     }
6994 }
6995 \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.

```

6996 \coffin_new:N \c_empty_coffin
6997 \hbox_set:Nn \c_empty_coffin { }
6998 \coffin_new:N \l_coffin_aligned_coffin
6999 \coffin_new:N \l_coffin_aligned_internal_coffin
        (End definition for \c_empty_coffin. This function is documented on page ??.)

```

190.3 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.

```

7000 \cs_new_protected_nopar:Npn \coffin_get_pole:NnN #1#2#3
7001 {
7002     \prop_get:cnNF
7003     { l_coffin_poles_ \int_value:w #1 _prop } {#2} #3
7004     {
7005         \msg_kernel_error:nxxx { coffins } { unknown-coffin-pole }
7006         {#2} { \token_to_str:N #1 }
7007         \tl_set:Nn #3 { { 0 pt } { 0 pt } { 0 pt } { 0 pt } }
7008     }
7009 }
        (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.

```

7010 \cs_new_protected_nopar:Npn \coffin_reset_structure:N #1
7011 {
7012     \prop_set_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7013     \c_coffin_corners_prop

```

```

7014     \prop_set_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7015     \c_coffin_poles_prop
7016 }

```

(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
7017 \cs_new_protected_nopar:Npn \coffin_set_eq_structure:NN #1#2
7018 {
7019     \prop_set_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7020     { l_coffin_corners_ \int_value:w #2 _prop }
7021     \prop_set_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7022     { l_coffin_poles_ \int_value:w #2 _prop }
7023 }
7024 \cs_new_protected_nopar:Npn \coffin_gset_eq_structure:NN #1#2
7025 {
7026     \prop_gset_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7027     { l_coffin_corners_ \int_value:w #2 _prop }
7028     \prop_gset_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7029     { l_coffin_poles_ \int_value:w #2 _prop }
7030 }

```

(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
7031 \cs_new_protected_nopar:Npn \coffin_set_user_dimensions:N #1
7032 {
7033     \cs_set_eq:NN \coffin_saved_Height: \Height
7034     \cs_set_eq:NN \coffin_saved_Depth: \Depth
7035     \cs_set_eq:NN \coffin_saved_TotalHeight: \TotalHeight
7036     \cs_set_eq:NN \coffin_saved_Width: \Width
7037     \cs_set_eq:NN \Height \l_coffin_Height_dim
7038     \cs_set_eq:NN \Depth \l_coffin_Depth_dim
7039     \cs_set_eq:NN \TotalHeight \l_coffin_TotalHeight_dim
7040     \cs_set_eq:NN \Width \l_coffin_Width_dim
7041     \dim_set:Nn \Height { \box_ht:N #1 }
7042     \dim_set:Nn \Depth { \box_dp:N #1 }
7043     \dim_set:Nn \TotalHeight { \box_ht:N #1 - \box_dp:N #1 }
7044     \dim_set:Nn \Width { \box_wd:N #1 }
7045 }
7046 \cs_new_protected_nopar:Npn \coffin_end_user_dimensions:
7047 {
7048     \cs_set_eq:NN \Height \coffin_saved_Height:
7049     \cs_set_eq:NN \Depth \coffin_saved_Depth:
7050     \cs_set_eq:NN \TotalHeight \coffin_saved_TotalHeight:
7051     \cs_set_eq:NN \Width \coffin_saved_Width:
7052 }

```

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

`\coffin_set_horizontal_pole:Nnn`
`\coffin_set_horizontal_pole:cnn`
`\coffin_set_vertical_pole:Nnn`
`\coffin_set_vertical_pole:cnn`
`\coffin_set_pole:Nnn`
`\coffin_set_pole:Nnx`

Setting the pole of a coffin at the user/designer level requires a bit more care. The idea here is to provide a reasonable interface to the system, then to do the setting with full expansion. The three-argument version is used internally to do a direct setting.

```

7053 \cs_new_protected_nopar:Npn \coffin_set_horizontal_pole:Nnn #1#2#3
7054 {
7055   \coffin_if_exist:NT #1
7056   {
7057     \coffin_set_user_dimensions:N #1
7058     \coffin_set_pole:Nnx #1 {#2}
7059     {
7060       { 0 pt } { \dim_eval:n {#3} }
7061       { 1000 pt } { 0 pt }
7062     }
7063     \coffin_end_user_dimensions:
7064   }
7065 }
7066 \cs_new_protected_nopar:Npn \coffin_set_vertical_pole:Nnn #1#2#3
7067 {
7068   \coffin_if_exist:NT #1
7069   {
7070     \coffin_set_user_dimensions:N #1
7071     \coffin_set_pole:Nnx #1 {#2}
7072     {
7073       { \dim_eval:n {#3} } { 0 pt }
7074       { 0 pt } { 1000 pt }
7075     }
7076     \coffin_end_user_dimensions:
7077   }
7078 }
7079 \cs_new_protected_nopar:Npn \coffin_set_pole:Nnn #1#2#3
7080 { \prop_put:cnn { l_coffin_poles_ \int_value:w #1 _prop } {#2} {#3} }
7081 \cs_generate_variant:Nn \coffin_set_horizontal_pole:Nnn { c }
7082 \cs_generate_variant:Nn \coffin_set_vertical_pole:Nnn { c }
7083 \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.

```

7084 \cs_new_protected_nopar:Npn \coffin_update_corners:N #1
7085 {
7086   \prop_put:cnn { l_coffin_corners_ \int_value:w #1 _prop } { tl }
7087   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7088   \prop_put:cnn { l_coffin_corners_ \int_value:w #1 _prop } { tr }
7089   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7090   \prop_put:cnn { l_coffin_corners_ \int_value:w #1 _prop } { bl }
7091   { { 0 pt } { \dim_eval:n { - \box_dp:N #1 } } }
7092   \prop_put:cnn { l_coffin_corners_ \int_value:w #1 _prop } { br }
7093   { { \dim_use:N \box_wd:N #1 } { \dim_eval:n { - \box_dp:N #1 } } }

```



```

7094 }
      (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.

```

7095 \cs_new_protected_nopar:Npn \coffin_update_poles:N #1
7096 {
7097   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { hc }
7098   {
7099     { \dim_eval:n { 0.5 \box_wd:N #1 } }
7100     { 0 pt } { 0 pt } { 1000 pt }
7101   }
7102   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { r }
7103   {
7104     { \dim_use:N \box_wd:N #1 }
7105     { 0 pt } { 0 pt } { 1000 pt }
7106   }
7107   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { vc }
7108   {
7109     { 0 pt }
7110     { \dim_eval:n { ( \box_ht:N #1 - \box_dp:N #1 ) / 2 } }
7111     { 1000 pt }
7112     { 0 pt }
7113   }
7114   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { t }
7115   {
7116     { 0 pt }
7117     { \dim_use:N \box_ht:N #1 }
7118     { 1000 pt }
7119     { 0 pt }
7120   }
7121   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { b }
7122   {
7123     { 0 pt }
7124     { \dim_eval:n { - \box_dp:N #1 } }
7125     { 1000 pt }
7126     { 0 pt }
7127   }
7128 }
      (End definition for \coffin_update_poles:N. This function is documented on page ??.)

```

190.4 Coffins: calculation of pole intersections

\coffin_calculate_intersection:Nmn The lead off in finding intersections is to recover the two poles and then hand off to the
\coffin_calculate_intersection:nmmmmmmn auxiliary for the actual calculation. There may of course not be an intersection, for which
\coffin_calculate_intersection_aux:nmmmmN an error trap is needed.

```

7129 \cs_new_protected_nopar:Npn \coffin_calculate_intersection:Nnn #1#2#3
7130 {
7131   \coffin_get_pole:NnN #1 {#2} \l_coffin_pole_a_tl
7132   \coffin_get_pole:NnN #1 {#3} \l_coffin_pole_b_tl
7133   \bool_set_false:N \l_coffin_error_bool
7134   \exp_last_two_unbraced:Noo
7135     \coffin_calculate_intersection:nnnnnnnn
7136     \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7137   \bool_if:NT \l_coffin_error_bool
7138   {
7139     \msg_kernel_error:nn { coffins } { no-pole-intersection }
7140     \dim_zero:N \l_coffin_x_dim
7141     \dim_zero:N \l_coffin_y_dim
7142   }
7143 }

```

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.

```

7144 \cs_new_protected_nopar:Npn \coffin_calculate_intersection:nnnnnnnn
7145   #1#2#3#4#5#6#7#8
7146 {
7147   \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.

```

7148 {
7149   \dim_set:Nn \l_coffin_x_dim {#1}
7150   \dim_compare:nNnTF {#7} = { \c_zero_dim
7151     { \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.

```

7152 {
7153   \dim_compare:nNnTF {#8} = { \c_zero_dim
7154     { \dim_set:Nn \l_coffin_y_dim {#6} }
7155     {
7156       \coffin_calculate_intersection_aux:nnnnnN
7157         {#1} {#5} {#6} {#7} {#8} \l_coffin_y_dim
7158     }
7159   }
7160 }

```

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.

```

7161 {
7162   \dim_compare:nNnTF {#4} = \c_zero_dim
7163   {
7164     \dim_set:Nn \l_coffin_y_dim {#2}
7165     \dim_compare:nNnTF {#8} = { \c_zero_dim }
7166     { \bool_set_true:N \l_coffin_error_bool }
7167     {
7168       \dim_compare:nNnTF {#7} = \c_zero_dim
7169       { \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.

```

7170 {
7171   \coffin_calculate_intersection_aux:nnnnnN
7172   {#2} {#6} {#5} {#8} {#7} \l_coffin_x_dim
7173 }
7174 }
7175 }

```

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.

```

7176 {
7177   \dim_compare:nNnTF {#7} = \c_zero_dim
7178   {
7179     \dim_set:Nn \l_coffin_x_dim {#5}
7180     \coffin_calculate_intersection_aux:nnnnnN
7181     {#5} {#1} {#2} {#3} {#4} \l_coffin_y_dim
7182   }
7183   {
7184     \dim_compare:nNnTF {#8} = \c_zero_dim
7185     {
7186       \dim_set:Nn \l_coffin_x_dim {#6}
7187       \coffin_calculate_intersection_aux:nnnnnN
7188       {#6} {#2} {#1} {#4} {#3} \l_coffin_x_dim
7189     }

```

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.

```

7190 {
7191   \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#3}
7192   \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#4}
7193   \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#7}
7194   \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#8}
7195   \fp_div:Nn \l_coffin_calc_b_fp \l_coffin_calc_a_fp

```

```

7196         \fp_div:Nn \l_coffin_calc_d_fp \l_coffin_calc_c_fp
7197         \fp_compare:nNnTF
7198             \l_coffin_calc_b_fp = \l_coffin_calc_d_fp
7199             { \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.

```

7200         {
7201             \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#6}
7202             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7203             \fp_sub:Nn \l_coffin_calc_result_fp
7204                 { \l_coffin_calc_a_fp }
7205             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#1}
7206             \fp_mul:Nn \l_coffin_calc_a_fp
7207                 { \l_coffin_calc_b_fp }
7208             \fp_add:Nn \l_coffin_calc_result_fp
7209                 { \l_coffin_calc_a_fp }
7210             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#5}
7211             \fp_mul:Nn \l_coffin_calc_a_fp
7212                 { \l_coffin_calc_d_fp }
7213             \fp_sub:Nn \l_coffin_calc_result_fp
7214                 { \l_coffin_calc_a_fp }
7215             \fp_sub:Nn \l_coffin_calc_b_fp
7216                 { \l_coffin_calc_d_fp }
7217             \fp_div:Nn \l_coffin_calc_result_fp
7218                 { \l_coffin_calc_b_fp }
7219             \dim_set:Nn \l_coffin_x_dim
7220                 { \fp_to_dim:N \l_coffin_calc_result_fp }
7221             \coffin_calculate_intersection_aux:nnnnnN
7222                 { \l_coffin_x_dim }
7223                 {#5} {#6} {#8} {#7} \l_coffin_y_dim
7224         }
7225     }
7226 }
7227 }
7228 }
7229 }

```

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.

```

7230 \cs_new_protected_nopar:Npn \coffin_calculate_intersection_aux:nnnnnN
7231   #1#2#3#4#5#6
7232   {
7233     \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#1}
7234     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7235     \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#3}
7236     \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#4}
7237     \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#5}
7238     \fp_sub:Nn \l_coffin_calc_result_fp { \l_coffin_calc_a_fp }
7239     \fp_div:Nn \l_coffin_calc_result_fp { \l_coffin_calc_d_fp }
7240     \fp_mul:Nn \l_coffin_calc_result_fp { \l_coffin_calc_c_fp }
7241     \fp_add:Nn \l_coffin_calc_result_fp { \l_coffin_calc_b_fp }
7242     \dim_set:Nn #6 { \fp_to_dim:N \l_coffin_calc_result_fp }
7243   }

```

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

190.5 Aligning and typesetting of coffins

```

\coffin_join:NnnNnnnn
\coffin_join:cnnNnnnn
\coffin_join:Nnncnnnn
\coffin_join:cnncnnnn

```

This command joins two coffins, using a horizontal and vertical pole from each coffin and making an offset between the two. The result is stored as the as a third coffin, which will have all of its handles reset to standard values. First, the more basic alignment function is used to get things started.

```

7244 \cs_new_protected_nopar:Npn \coffin_join:NnnNnnnn #1#2#3#4#5#6#7#8
7245   {
7246     \coffin_align:NnnNnnnnN
7247     #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.

```

7248   \hbox_set:Nn \l_coffin_aligned_coffin
7249   {
7250     \dim_compare:nNnT { \l_coffin_offset_x_dim } < \c_zero_dim
7251     { \tex_kern:D -\l_coffin_offset_x_dim }
7252     \hbox_unpack:N \l_coffin_aligned_coffin
7253     \dim_set:Nn \l_coffin_tmp_dim
7254     { \l_coffin_offset_x_dim - \box_wd:N #1 + \box_wd:N #4 }
7255     \dim_compare:nNnT \l_coffin_tmp_dim < \c_zero_dim
7256     { \tex_kern:D -\l_coffin_tmp_dim }
7257   }

```

The coffin structure is reset, and the corners are cleared: only those from the two parent coffins are needed.

```

7258   \coffin_reset_structure:N \l_coffin_aligned_coffin
7259   \prop_clear:c
7260   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _ prop }
7261   \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.

```

7262     \dim_compare:nNnTF \l_coffin_offset_x_dim < \c_zero_dim
7263     {
7264         \coffin_offset_poles:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7265         \coffin_offset_poles:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7266         \coffin_offset_corners:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7267         \coffin_offset_corners:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7268     }
7269     {
7270         \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7271         \coffin_offset_poles:Nnn #4
7272             { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7273         \coffin_offset_corners:Nnn #1 { 0 pt } { 0 pt }
7274         \coffin_offset_corners:Nnn #4
7275             { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7276     }
7277     \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7278     \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7279 }
7280 \cs_generate_variant:Nn \coffin_join:NnnNnnnn { c , Nnnc , cncnc }

```

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

```

\coffin_attach:NnnNnnnn
\coffin_attach:cnnNnnnn
\coffin_attach:Nnncnnnn
\coffin_attach:cncncnnn

```

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. The function used when marking a position is hear also as it is similar but without the structure updates.

```
\coffin_attach_mark:NnnNnnnn
```

```

7281 \cs_new_protected_nopar:Npn \coffin_attach:NnnNnnnn #1#2#3#4#5#6#7#8
7282 {
7283     \coffin_align:NnnNnnnnN
7284     #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7285     \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7286     \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7287     \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7288     \coffin_reset_structure:N \l_coffin_aligned_coffin
7289     \prop_set_eq:cc
7290     { \l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7291     { \l_coffin_corners_ \int_value:w #1 _prop }
7292     \coffin_update_poles:N \l_coffin_aligned_coffin
7293     \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7294     \coffin_offset_poles:Nnn #4
7295         { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7296     \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7297     \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7298 }
7299 \cs_new_protected_nopar:Npn \coffin_attach_mark:NnnNnnnn #1#2#3#4#5#6#7#8
7300 {

```

```

7301 \coffin_align:NnnNnnnnN
7302   #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7303 \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7304 \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7305 \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7306 \box_set_eq:NN #1 \l_coffin_aligned_coffin
7307 }
7308 \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.

```

7309 \cs_new_protected_nopar:Npn \coffin_align:NnnNnnnnN #1#2#3#4#5#6#7#8#9
7310 {
7311   \coffin_calculate_intersection:Nnn #4 {#5} {#6}
7312   \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7313   \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7314   \coffin_calculate_intersection:Nnn #1 {#2} {#3}
7315   \dim_set:Nn \l_coffin_offset_x_dim
7316     { \l_coffin_x_dim - \l_coffin_x_prime_dim + #7 }
7317   \dim_set:Nn \l_coffin_offset_y_dim
7318     { \l_coffin_y_dim - \l_coffin_y_prime_dim + #8 }
7319   \hbox_set:Nn \l_coffin_aligned_internal_coffin
7320   {
7321     \box_use:N #1
7322     \tex_kern:D -\box_wd:N #1
7323     \tex_kern:D \l_coffin_offset_x_dim
7324     \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #4 }
7325   }
7326   \coffin_set_eq:NN #9 \l_coffin_aligned_internal_coffin
7327 }
      (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.

```

7328 \cs_new_protected_nopar:Npn \coffin_offset_poles:Nnn #1#2#3
7329 {

```

```

7330 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7331 { \coffin_offset_pole:Nnnnnnn #1 {##1} ##2 {#2} {#3} }
7332 }
7333 \cs_new_protected_nopar:Npn \coffin_offset_pole:Nnnnnnn #1#2#3#4#5#6#7#8
7334 {
7335   \dim_set:Nn \l_coffin_x_dim { #3 + #7 }
7336   \dim_set:Nn \l_coffin_y_dim { #4 + #8 }
7337   \tl_if_in:nnTF {#2} { - }
7338   { \tl_set:Nn \l_coffin_tmp_tl { {#2} } }
7339   { \tl_set:Nn \l_coffin_tmp_tl { { #1 - #2 } } }
7340   \exp_last_unbraced:NNo \coffin_set_pole:Nnx \l_coffin_aligned_coffin
7341   { \l_coffin_tmp_tl }
7342   {
7343     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7344     {#5} {#6}
7345   }
7346 }

```

(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.

```

7347 \cs_new_protected_nopar:Npn \coffin_offset_corners:Nnn #1#2#3
7348 {
7349   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7350   { \coffin_offset_corner:Nnnnn #1 {##1} ##2 {#2} {#3} }
7351 }
7352 \cs_new_protected_nopar:Npn \coffin_offset_corner:Nnnnn #1#2#3#4#5#6
7353 {
7354   \prop_put:cnx
7355   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7356   { #1 - #2 }
7357   {
7358     { \dim_eval:n { #3 + #5 } }
7359     { \dim_eval:n { #4 + #6 } }
7360   }
7361 }

```

(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.

```

7362 \cs_new_protected_nopar:Npn \coffin_update_vertical_poles:NNN #1#2#3
7363 {
7364   \coffin_get_pole:NnN #3 { #1 -T } \l_coffin_pole_a_tl
7365   \coffin_get_pole:NnN #3 { #2 -T } \l_coffin_pole_b_tl
7366   \exp_last_two_unbraced:Noo \coffin_update_T:nnnnnnnnN
7367   \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7368   \coffin_get_pole:NnN #3 { #1 -B } \l_coffin_pole_a_tl
7369   \coffin_get_pole:NnN #3 { #2 -B } \l_coffin_pole_b_tl

```



```

7370 \exp_last_two_unbraced:Noo \coffin_update_B:nnnnnnnnN
7371 \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7372 }
7373 \cs_new_protected_nopar:Npn \coffin_update_T:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7374 {
7375 \dim_compare:nNnTF {#2} < {#6}
7376 {
7377 \coffin_set_pole:Nnx #9 { T }
7378 { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7379 }
7380 {
7381 \coffin_set_pole:Nnx #9 { T }
7382 { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7383 }
7384 }
7385 \cs_new_protected_nopar:Npn \coffin_update_B:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7386 {
7387 \dim_compare:nNnTF {#2} < {#6}
7388 {
7389 \coffin_set_pole:Nnx #9 { B }
7390 { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7391 }
7392 {
7393 \coffin_set_pole:Nnx #9 { B }
7394 { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7395 }
7396 }

```

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

`\coffin_typeset:Nnnnn`
`\coffin_typeset:cnnnn`

Typesetting a coffin means aligning it with the current position, which is done using a coffin with no content at all. As well as aligning to the empty coffin, there is also a need to leave vertical mode, if necessary.

```

7397 \cs_new_protected_nopar:Npn \coffin_typeset:Nnnnn #1#2#3#4#5
7398 {
7399 \coffin_align:NnnNnnnnN \c_empty_coffin { H } { 1 }
7400 #1 {#2} {#3} {#4} {#5} \l_coffin_aligned_coffin
7401 \hbox_unpack:N \c_empty_box
7402 \box_use:N \l_coffin_aligned_coffin
7403 }
7404 \cs_generate_variant:Nn \coffin_typeset:Nnnnn { c }

```

(End definition for `\coffin_typeset:Nnnnn` and `\coffin_typeset:cnnnn`. These functions are documented on page ??.)

190.6 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.

```

7405 \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.

7406 \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

7407 \dim_new:N \l_coffin_left_corner_dim

7408 \dim_new:N \l_coffin_right_corner_dim

7409 \dim_new:N \l_coffin_bottom_corner_dim

7410 \dim_new:N \l_coffin_top_corner_dim

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

\coffin_rotate:Nn Rotating a coffin requires several steps which can be conveniently run together. The
 \coffin_rotate:cn first step is to convert the angle given in degrees to one in radians. This is then used
 to set \l_coffin_sin_fp and \l_coffin_cos_fp, which are carried through unchanged
 for the rest of the procedure.

7411 \cs_new_protected_nopar:Npn \coffin_rotate:Nn #1#2

7412 {

7413 \fp_set:Nn \l_coffin_tmp_fp {#2}

7414 \fp_div:Nn \l_coffin_tmp_fp { 180 }

7415 \fp_mul:Nn \l_coffin_tmp_fp { \c_pi_fp }

7416 \fp_sin:Nn \l_coffin_sin_fp { \l_coffin_tmp_fp }

7417 \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.

7418 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }

7419 { \coffin_rotate_corner:Nnnn #1 {##1} ##2 }

7420 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }

7421 { \coffin_rotate_pole:Nnnnnn #1 {##1} ##2 }

The bounding box of the coffin needs to be rotated, and to do this the corners have to be found first. They are then rotated in the same way as the corners of the coffin material itself.

7422 \coffin_set_bounding:N #1

7423 \prop_map_inline:Nn \l_coffin_bounding_prop

7424 { \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.

7425 \coffin_find_corner_maxima:N #1

7426 \coffin_find_bounding_shift:

7427 \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.

```

7428 \hbox_set:Nn #1
7429 {
7430   \tex_kern:D \l_coffin_bounding_shift_dim
7431   \tex_kern:D -\l_coffin_left_corner_dim
7432   \box_move_down:nn { \l_coffin_bottom_corner_dim }
7433   { \box_use:N #1 }
7434 }

```

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.

```

7435 \box_set_ht:Nn #1
7436 { \l_coffin_top_corner_dim - \l_coffin_bottom_corner_dim }
7437 \box_set_dp:Nn #1 { 0 pt }
7438 \box_set_wd:Nn #1
7439 { \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.

```

7440 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7441 { \coffin_shift_corner:Nnnn #1 {##1} ##2 }
7442 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7443 { \coffin_shift_pole:Nnnnnn #1 {##1} ##2 }
7444 }
7445 \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.

```

7446 \cs_new_protected_nopar:Npn \coffin_set_bounding:N #1
7447 {
7448   \prop_put:Nnx \l_coffin_bounding_prop { tl }
7449   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7450   \prop_put:Nnx \l_coffin_bounding_prop { tr }
7451   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7452   \dim_set:Nn \l_coffin_tmp_dim { - \box_dp:N #1 }
7453   \prop_put:Nnx \l_coffin_bounding_prop { bl }
7454   { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } }
7455   \prop_put:Nnx \l_coffin_bounding_prop { br }
7456   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_tmp_dim } }
7457 }

```

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

`\coffin_rotate_bounding:nnn` Rotating the position of the corner of the coffin is just a case of treating this as a vector
`\coffin_rotate_corner:Nnnn` from the reference point. The same treatment is used for the corners of the material itself
and the bounding box.

```

7458 \cs_new_protected_nopar:Npn \coffin_rotate_bounding:nnn #1#2#3
7459 {
7460   \coffin_rotate_vector:nnNN {#2} {#3} \l_coffin_x_dim \l_coffin_y_dim
7461   \prop_put:Nnx \l_coffin_bounding_prop {#1}
7462   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7463 }
7464 \cs_new_protected_nopar:Npn \coffin_rotate_corner:Nnnn #1#2#3#4
7465 {
7466   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7467   \prop_put:cnx { \l_coffin_corners_ \int_value:w #1 _prop } {#2}
7468   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7469 }

```

(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.

```

7470 \cs_new_protected_nopar:Npn \coffin_rotate_pole:Nnnnnn #1#2#3#4#5#6
7471 {
7472   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7473   \coffin_rotate_vector:nnNN {#5} {#6}
7474   \l_coffin_x_prime_dim \l_coffin_y_prime_dim
7475   \coffin_set_pole:Nnx #1 {#2}
7476   {
7477     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7478     { \dim_use:N \l_coffin_x_prime_dim }
7479     { \dim_use:N \l_coffin_y_prime_dim }
7480   }
7481 }

```

(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.

```

7482 \cs_new_protected_nopar:Npn \coffin_rotate_vector:nnNN #1#2#3#4
7483 {
7484   \fp_set_from_dim:Nn \l_coffin_x_fp {#1}
7485   \fp_set_from_dim:Nn \l_coffin_y_fp {#2}
7486   \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
7487   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7488   \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
7489   \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_sin_fp }
7490   \fp_sub:Nn \l_coffin_x_prime_fp { \l_coffin_tmp_fp }
7491   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7492   \fp_set_eq:NN \l_coffin_tmp_fp \l_coffin_x_fp

```

```

7493     \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
7494     \fp_mul:Nn \l_coffin_tmp_fp      { \l_coffin_sin_fp }
7495     \fp_add:Nn \l_coffin_y_prime_fp { \l_coffin_tmp_fp }
7496     \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_x_prime_fp }
7497     \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_y_prime_fp }
7498 }

```

(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.

```

7499 \cs_new_protected_nopar:Npn \coffin_find_corner_maxima:N #1
7500 {
7501     \dim_set:Nn \l_coffin_top_corner_dim { -\c_max_dim }
7502     \dim_set:Nn \l_coffin_right_corner_dim { -\c_max_dim }
7503     \dim_set:Nn \l_coffin_bottom_corner_dim { \c_max_dim }
7504     \dim_set:Nn \l_coffin_left_corner_dim { \c_max_dim }
7505     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7506     { \coffin_find_corner_maxima_aux:nn ##2 }
7507 }
7508 \cs_new_protected_nopar:Npn \coffin_find_corner_maxima_aux:nn #1#2
7509 {
7510     \dim_set_min:Nn \l_coffin_left_corner_dim {#1}
7511     \dim_set_max:Nn \l_coffin_right_corner_dim {#1}
7512     \dim_set_min:Nn \l_coffin_bottom_corner_dim {#2}
7513     \dim_set_max:Nn \l_coffin_top_corner_dim {#2}
7514 }

```

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

\coffin_find_bounding_shift:
\coffin_find_bounding_shift_aux:nn

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.

```

7515 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift:
7516 {
7517     \dim_set:Nn \l_coffin_bounding_shift_dim { \c_max_dim }
7518     \prop_map_inline:Nn \l_coffin_bounding_prop
7519     { \coffin_find_bounding_shift_aux:nn ##2 }
7520 }
7521 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift_aux:nn #1#2
7522 { \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
\coffin_shift_pole:Nnnnnn

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.

```

7523 \cs_new_protected_nopar:Npn \coffin_shift_corner:Nnnn #1#2#3#4
7524 {

```

```

7525 \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _ prop } {#2}
7526 {
7527   { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7528   { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7529 }
7530 }
7531 \cs_new_protected_nopar:Npn \coffin_shift_pole:Nnnnnn #1#2#3#4#5#6
7532 {
7533   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _ prop } {#2}
7534   {
7535     { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7536     { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7537     {#5} {#6}
7538   }
7539 }

```

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

190.7 Resizing coffins

\l_coffin_scale_x_fp Storage for the scaling factors in x and y , respectively.
 \l_coffin_scale_y_fp

```

7540 \fp_new:N \l_coffin_scale_x_fp
7541 \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

```

7542 \dim_new:N \l_coffin_scaled_total_height_dim
7543 \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.

```

7544 \cs_new_protected_nopar:Npn \coffin_resize:Nnn #1#2#3
7545 {
7546   \coffin_set_user_dimensions:N #1
7547   \box_resize:Nnn #1 {#2} {#3}
7548   \fp_set_from_dim:Nn \l_coffin_scale_x_fp {#2}
7549   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \Width }
7550   \fp_div:Nn \l_coffin_scale_x_fp { \l_coffin_tmp_fp }
7551   \fp_set_from_dim:Nn \l_coffin_scale_y_fp {#3}
7552   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \TotalHeight }
7553   \fp_div:Nn \l_coffin_scale_y_fp { \l_coffin_tmp_fp }
7554   \coffin_resize_common:Nnn #1 {#2} {#3}
7555 }
7556 \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.

```

7557 \cs_new_protected_nopar:Npn \coffin_resize_common:Nnn #1#2#3
7558 {
7559   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7560   { \coffin_scale_corner:Nnnn #1 {##1} ##2 }
7561   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7562   { \coffin_scale_pole:Nnnnnn #1 {##1} ##2 }

```

Negative x -scaling values will place the poles in the wrong location: this is corrected here.

```

7563   \fp_compare:NNNT \l_coffin_scale_x_fp < \c_zero_fp
7564   {
7565     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7566     { \coffin_x_shift_corner:Nnnn #1 {##1} ##2 }
7567     \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7568     { \coffin_x_shift_pole:Nnnnnn #1 {##1} ##2 }
7569   }
7570   \coffin_end_user_dimensions:
7571 }

```

(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.

```

7572 \cs_new_protected_nopar:Npn \coffin_scale:Nnn #1#2#3
7573 {
7574   \box_scale:Nnn #1 {#2} {#3}
7575   \coffin_set_user_dimensions:N #1
7576   \fp_set:Nn \l_coffin_scale_x_fp {#2}
7577   \fp_set:Nn \l_coffin_scale_y_fp {#3}
7578   \fp_compare:NNNTF \l_coffin_scale_y_fp > \c_zero_fp
7579   { \l_coffin_scaled_total_height_dim #3 \TotalHeight }
7580   { \l_coffin_scaled_total_height_dim -#3 \TotalHeight }
7581   \fp_compare:NNNTF \l_coffin_scale_x_fp > \c_zero_fp
7582   { \l_coffin_scaled_width_dim -#2 \Width }
7583   { \l_coffin_scaled_width_dim #2 \Width }
7584   \coffin_resize_common:Nnn #1
7585   { \l_coffin_scaled_width_dim } { \l_coffin_scaled_total_height_dim }
7586 }
7587 \cs_generate_variant:Nn \coffin_scale:Nnn { c }

```

(End definition for \coffin_scale:Nnn and \coffin_scale:cnn. These functions are documented on page ??.)

`\coffin_scale_vector:nnNN` This function scales a vector from the origin using the pre-set scale factors in x and y . This is a much less complex operation than rotation, and as a result the code is a lot clearer.

```

7588 \cs_new_protected_nopar:Npn \coffin_scale_vector:nnNN #1#2#3#4

```

```

7589 {
7590   \fp_set_from_dim:Nn \l_coffin_tmp_fp {#1}
7591   \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_x_fp }
7592   \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_tmp_fp }
7593   \fp_set_from_dim:Nn \l_coffin_tmp_fp {#2}
7594   \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_y_fp }
7595   \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_tmp_fp }
7596 }

```

(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
7597 \cs_new_protected_nopar:Npn \coffin_scale_corner:Nnnn #1#2#3#4
7598 {
7599   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7600   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7601   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7602 }
7603 \cs_new_protected_nopar:Npn \coffin_scale_pole:Nnnnnn #1#2#3#4#5#6
7604 {
7605   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7606   \coffin_set_pole:Nnx #1 {#2}
7607   {
7608     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7609     {#5} {#6}
7610   }
7611 }

```

(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.

```

7612 \cs_new_protected_nopar:Npn \coffin_x_shift_corner:Nnnn #1#2#3#4
7613 {
7614   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7615   {
7616     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7617   }
7618 }
7619 \cs_new_protected_nopar:Npn \coffin_x_shift_pole:Nnnnnn #1#2#3#4#5#6
7620 {
7621   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } {#2}
7622   {
7623     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7624     {#5} {#6}
7625   }
7626 }

```

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

190.8 Coffin diagnostics

`\l_coffin_display_coffin` Used for printing coffins with data structures attached.

```

\l_coffin_display_coord_coffin 7627 \coffin_new:N \l_coffin_display_coffin
\l_coffin_display_pole_coffin 7628 \coffin_new:N \l_coffin_display_coord_coffin
7629 \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.

```

7630 \prop_new:N \l_coffin_display_handles_prop
7631 \prop_put:Nnn \l_coffin_display_handles_prop { tl }
7632 { { b } { r } { -1 } { 1 } }
7633 \prop_put:Nnn \l_coffin_display_handles_prop { thc }
7634 { { b } { hc } { 0 } { 1 } }
7635 \prop_put:Nnn \l_coffin_display_handles_prop { tr }
7636 { { b } { l } { 1 } { 1 } }
7637 \prop_put:Nnn \l_coffin_display_handles_prop { vc1 }
7638 { { vc } { r } { -1 } { 0 } }
7639 \prop_put:Nnn \l_coffin_display_handles_prop { vhc }
7640 { { vc } { hc } { 0 } { 0 } }
7641 \prop_put:Nnn \l_coffin_display_handles_prop { vcr }
7642 { { vc } { l } { 1 } { 0 } }
7643 \prop_put:Nnn \l_coffin_display_handles_prop { bl }
7644 { { t } { r } { -1 } { -1 } }
7645 \prop_put:Nnn \l_coffin_display_handles_prop { bhc }
7646 { { t } { hc } { 0 } { -1 } }
7647 \prop_put:Nnn \l_coffin_display_handles_prop { br }
7648 { { t } { l } { 1 } { -1 } }
7649 \prop_put:Nnn \l_coffin_display_handles_prop { Tl }
7650 { { t } { r } { -1 } { -1 } }
7651 \prop_put:Nnn \l_coffin_display_handles_prop { Thc }
7652 { { t } { hc } { 0 } { -1 } }
7653 \prop_put:Nnn \l_coffin_display_handles_prop { Tr }
7654 { { t } { l } { 1 } { -1 } }
7655 \prop_put:Nnn \l_coffin_display_handles_prop { Hl }
7656 { { vc } { r } { -1 } { 1 } }
7657 \prop_put:Nnn \l_coffin_display_handles_prop { Hhc }
7658 { { vc } { hc } { 0 } { 1 } }
7659 \prop_put:Nnn \l_coffin_display_handles_prop { Hr }
7660 { { vc } { l } { 1 } { 1 } }
7661 \prop_put:Nnn \l_coffin_display_handles_prop { Bl }
7662 { { b } { r } { -1 } { -1 } }
7663 \prop_put:Nnn \l_coffin_display_handles_prop { Bhc }
7664 { { b } { hc } { 0 } { -1 } }
7665 \prop_put:Nnn \l_coffin_display_handles_prop { Br }
7666 { { b } { l } { 1 } { -1 } }

```

(End definition for `\l_coffin_display_handles_prop`. This function is documented on page ??.)

<code>\l_coffin_display_offset_dim</code>	<p>The standard offset for the label from the handle position when displaying handles.</p> <pre> 7667 \dim_new:N \l_coffin_display_offset_dim 7668 \dim_set:Nn \l_coffin_display_offset_dim { 2 pt } (End definition for \l_coffin_display_offset_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_display_x_dim</code> <code>\l_coffin_display_y_dim</code>	<p>As the intersections of poles have to be calculated to find which ones to print, there is a need to avoid repetition. This is done by saving the intersection into two dedicated values.</p> <pre> 7669 \dim_new:N \l_coffin_display_x_dim 7670 \dim_new:N \l_coffin_display_y_dim (End definition for \l_coffin_display_x_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_display_poles_prop</code>	<p>A property list for printing poles: various things need to be deleted from this to get a “nice” output.</p> <pre> 7671 \prop_new:N \l_coffin_display_poles_prop (End definition for \l_coffin_display_poles_prop. This function is documented on page ??.) </pre>
<code>\l_coffin_display_font_tl</code>	<p>Stores the settings used to print coffin data: this keeps things flexible.</p> <pre> 7672 \tl_new:N \l_coffin_display_font_tl 7673 <*initex> 7674 \tl_set:Nn \l_coffin_display_font_tl { } % TODO 7675 </initex> 7676 <*package> 7677 \tl_set:Nn \l_coffin_display_font_tl { \sffamily \tiny } 7678 </package> (End definition for \l_coffin_display_font_tl. This function is documented on page ??.) </pre>
<code>\l_coffin_handles_tmp_prop</code>	<p>Used for displaying coffins, as the handles need to be stored in this case, at least temporarily.</p> <pre> 7679 \prop_new:N \l_coffin_handles_tmp_prop (End definition for \l_coffin_handles_tmp_prop. This function is documented on page ??.) </pre>
<code>\coffin_mark_handle:Nnnn</code> <code>\coffin_mark_handle:cnnn</code> <code>\coffin_mark_handle_aux:nnnnNnn</code>	<p>Marking a single handle is relatively easy. The standard attachment function is used, meaning that there are two calculations for the location. However, this is likely to be okay given the load expected. Contrast with the more optimised version for showing all handles which comes next.</p> <pre> 7680 \cs_new_protected_nopar:Npn \coffin_mark_handle:Nnnn #1#2#3#4 7681 { 7682 \hcoffin_set:Nn \l_coffin_display_pole_coffin 7683 { 7684 <*initex> 7685 \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO 7686 </initex> 7687 <*package> 7688 \color {#4} 7689 \rule { 1 pt } { 1 pt } 7690 </package> </pre>

```

7691     }
7692     \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7693     \l_coffin_display_pole_coffin { hc } { vc } { 0 pt } { 0 pt }
7694     \hcoffin_set:Nn \l_coffin_display_coord_coffin
7695     {
7696     <*initex>
7697         % TODO
7698     </initex>
7699     <*package>
7700         \color {#4}
7701     </package>
7702     \l_coffin_display_font_tl
7703     ( \tl_to_str:n { #2 , #3 } )
7704     }
7705     \prop_get:NnN \l_coffin_display_handles_prop
7706     { #2 #3 } \l_coffin_tmp_tl
7707     \quark_if_no_value:NTF \l_coffin_tmp_tl
7708     {
7709         \prop_get:NnN \l_coffin_display_handles_prop
7710         { #3 #2 } \l_coffin_tmp_tl
7711         \quark_if_no_value:NTF \l_coffin_tmp_tl
7712         {
7713             \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7714             \l_coffin_display_coord_coffin { l } { vc }
7715             { 1 pt } { 0 pt }
7716         }
7717         {
7718             \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7719             \l_coffin_tmp_tl #1 {#2} {#3}
7720         }
7721     }
7722     {
7723         \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7724         \l_coffin_tmp_tl #1 {#2} {#3}
7725     }
7726 }
7727 \cs_new_protected_nopar:Npn \coffin_mark_handle_aux:nnnnNnn #1#2#3#4#5#6#7
7728 {
7729     \coffin_attach_mark:NnnNnnnn #5 {#6} {#7}
7730     \l_coffin_display_coord_coffin {#1} {#2}
7731     { #3 \l_coffin_display_offset_dim }
7732     { #4 \l_coffin_display_offset_dim }
7733 }
7734 \cs_generate_variant:Nn \coffin_mark_handle:Nnnn { c }

```

(End definition for \coffin_mark_handle:Nnnn and \coffin_mark_handle:cnnn. These functions are documented on page ??.)

\coffin_display_handles:Nn Printing the poles starts by removing any duplicates, for which the H poles is used as
\coffin_display_handles:cn the definitive version for the baseline and bottom. Two loops are then used to find the
\coffin_display_handles_aux:nnnnnn
\coffin_display_handles_aux:nnnn
\coffin_display_attach:Nnnnn

combinations of handles for all of these poles. This is done such that poles are removed during the loops to avoid duplication.

```

7735 \cs_new_protected_nopar:Npn \coffin_display_handles:Nn #1#2
7736 {
7737   \hcoffin_set:Nn \l_coffin_display_pole_coffin
7738   {
7739     \*initex
7740     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
7741     \*initex
7742     \*package
7743     \color {#2}
7744     \rule { 1 pt } { 1 pt }
7745     \*package
7746   }
7747   \prop_set_eq:Nc \l_coffin_display_poles_prop
7748   { l_coffin_poles_ \int_value:w #1 _prop }
7749   \coffin_get_pole:NnN #1 { H } \l_coffin_pole_a_tl
7750   \coffin_get_pole:NnN #1 { T } \l_coffin_pole_b_tl
7751   \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7752   { \prop_del:Nn \l_coffin_display_poles_prop { T } }
7753   \coffin_get_pole:NnN #1 { B } \l_coffin_pole_b_tl
7754   \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7755   { \prop_del:Nn \l_coffin_display_poles_prop { B } }
7756   \coffin_set_eq:NN \l_coffin_display_coffin #1
7757   \prop_clear:N \l_coffin_handles_tmp_prop
7758   \prop_map_inline:Nn \l_coffin_display_poles_prop
7759   {
7760     \prop_del:Nn \l_coffin_display_poles_prop {##1}
7761     \coffin_display_handles_aux:nnnnnn {##1} ##2 {#2}
7762   }
7763   \box_use:N \l_coffin_display_coffin
7764 }

```

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.

```

7765 \cs_new_protected_nopar:Npn \coffin_display_handles_aux:nnnnnn #1#2#3#4#5#6
7766 {
7767   \prop_map_inline:Nn \l_coffin_display_poles_prop
7768   {
7769     \bool_set_false:N \l_coffin_error_bool
7770     \coffin_calculate_intersection:nnnnnnnn {#2} {#3} {#4} {#5} ##2
7771     \bool_if:NF \l_coffin_error_bool
7772     {
7773       \dim_set:Nn \l_coffin_display_x_dim { \l_coffin_x_dim }
7774       \dim_set:Nn \l_coffin_display_y_dim { \l_coffin_y_dim }
7775       \coffin_display_attach:Nnnnn
7776       \l_coffin_display_pole_coffin { hc } { vc }
7777       { 0 pt } { 0 pt }
7778       \hcoffin_set:Nn \l_coffin_display_coord_coffin

```

```

7779 {
7780 <*initex>
7781 % TODO
7782 </initex>
7783 <*package>
7784 \color {#6}
7785 </package>
7786 \l_coffin_display_font_tl
7787 ( \tl_to_str:n { #1 , ##1 } )
7788 }
7789 \prop_get:NnN \l_coffin_display_handles_prop
7790 { #1 ##1 } \l_coffin_tmp_tl
7791 \quark_if_no_value:NTF \l_coffin_tmp_tl
7792 {
7793 \prop_get:NnN \l_coffin_display_handles_prop
7794 { ##1 #1 } \l_coffin_tmp_tl
7795 \quark_if_no_value:NTF \l_coffin_tmp_tl
7796 {
7797 \coffin_display_attach:Nnnnn
7798 \l_coffin_display_coord_coffin { 1 } { vc }
7799 { 1 pt } { 0 pt }
7800 }
7801 {
7802 \exp_last_unbraced:No
7803 \coffin_display_handles_aux:nnnn
7804 \l_coffin_tmp_tl
7805 }
7806 }
7807 {
7808 \exp_last_unbraced:No \coffin_display_handles_aux:nnnn
7809 \l_coffin_tmp_tl
7810 }
7811 }
7812 }
7813 }
7814 \cs_new_protected_nopar:Npn \coffin_display_handles_aux:nnnn #1#2#3#4
7815 {
7816 \coffin_display_attach:Nnnnn
7817 \l_coffin_display_coord_coffin {#1} {#2}
7818 { #3 \l_coffin_display_offset_dim }
7819 { #4 \l_coffin_display_offset_dim }
7820 }
7821 \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.

```

7822 \cs_new_protected_nopar:Npn \coffin_display_attach:Nnnnn #1#2#3#4#5
7823 {
7824 \coffin_calculate_intersection:Nnn #1 {#2} {#3}

```

```

7825 \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7826 \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7827 \dim_set:Nn \l_coffin_offset_x_dim
7828 { \l_coffin_display_x_dim - \l_coffin_x_prime_dim + #4 }
7829 \dim_set:Nn \l_coffin_offset_y_dim
7830 { \l_coffin_display_y_dim - \l_coffin_y_prime_dim + #5 }
7831 \hbox_set:Nn \l_coffin_aligned_coffin
7832 {
7833   \box_use:N \l_coffin_display_coffin
7834   \tex_kern:D -\box_wd:N \l_coffin_display_coffin
7835   \tex_kern:D \l_coffin_offset_x_dim
7836   \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #1 }
7837 }
7838 \box_set_ht:Nn \l_coffin_aligned_coffin
7839 { \box_ht:N \l_coffin_display_coffin }
7840 \box_set_dp:Nn \l_coffin_aligned_coffin
7841 { \box_dp:N \l_coffin_display_coffin }
7842 \box_set_wd:Nn \l_coffin_aligned_coffin
7843 { \box_wd:N \l_coffin_display_coffin }
7844 \box_set_eq:NN \l_coffin_display_coffin \l_coffin_aligned_coffin
7845 }

```

(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
7846 \cs_new_protected_nopar:Npn \coffin_show_structure:N #1
7847 {
7848   \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
7849   {
7850     \iow_term:x
7851     {
7852       \iow_newline:
7853       Size-of~coffin~\token_to_str:N #1 : \iow_newline:
7854       > ~ ht~~\dim_use:N \box_ht:N #1 \iow_newline:
7855       > ~ dp~~\dim_use:N \box_dp:N #1 \iow_newline:
7856       > ~ wd~~\dim_use:N \box_wd:N #1 \iow_newline:
7857     }
7858     \iow_term:x { Poles-of~coffin~\token_to_str:N #1 : }
7859     \tl_set:Nx \l_coffin_tmp_tl
7860     {
7861       \prop_map_function:cN
7862       { l_coffin_poles_ \int_value:w #1 _prop }
7863       \coffin_show_aux:nn
7864     }
7865     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
7866     { \exp_after:wN \coffin_show_aux:w \l_coffin_tmp_tl }
7867   }
7868   {
7869     \iow_term:x { ---No~poles~found--- }

```

```

7870         \tl_show:n { Is~this~really~a~coffin? }
7871     }
7872 }
7873 \cs_new:Npn \coffin_show_aux:nn #1#2
7874 {
7875     \iow_newline: > \c_space_tl \c_space_tl
7876     #1 \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
7877 }
7878 \cs_new_nopar:Npn \coffin_show_aux:w #1 > ~ { }
7879 \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 ??.)

190.9 Messages

```

7880 \msg_kernel_new:nnnn { coffins } { no-pole-intersection }
7881 { No~intersection~between~coffin~poles. }
7882 {
7883     \c_msg_coding_error_text_tl
7884     LaTeX~was~asked~to~find~the~intersection~between~two~poles,~
7885     but~they~do~not~have~a~unique~meeting~point:~
7886     the~value~(0~pt,~0~pt)~will~be~used.
7887 }
7888 \msg_kernel_new:nnnn { coffins } { unknown-coffin }
7889 { Unknown~coffin~'#1'. }
7890 { The~coffin~'#1'~was~never~defined. }
7891 \msg_kernel_new:nnnn { coffins } { unknown-coffin-pole }
7892 { Pole~'#1'~unknown~for~coffin~'#2'. }
7893 {
7894     \c_msg_coding_error_text_tl
7895     LaTeX~was~asked~to~find~a~typesetting~pole~for~a~coffin,~
7896     but~either~the~coffin~does~not~exist~or~the~pole~name~is~wrong.
7897 }
7898 </initex | package>

```

191 l3color Implementation

```

7899 <*initex | package>
7900 <*package>
7901 \ProvidesExplPackage
7902   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
7903 \package_check_loaded_expl:
7904 </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.

```

7905 \cs_new_eq:NN \color_group_begin: \group_begin:
7906 \cs_new_protected_nopar:Npn \color_group_end:

```

```

7907 {
7908     \tex_par:D
7909     \group_end:
7910 }

```

(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”.

```

7911 \<initex>
7912 \cs_new_protected_nopar:Npn \color_ensure_current:
7913 { \driver_color_ensure_current: }
7914 \</initex>
7915 \<package>
7916 \cs_new_protected_nopar:Npn \color_ensure_current: { \set@color }
7917 \</package>
7918 \</initex | package>

```

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

192 l3io implementation

```

7919 \<initex | package>
7920 \<package>
7921 \ProvidesExplPackage
7922 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
7923 \package_check_loaded_expl:
7924 \</package>

```

192.1 Primitives

\if_eof:w The primitive conditional

```

7925 \cs_new_eq:NN \if_eof:w \tex_ifeof:D

```

(End definition for \if_eof:w. This function is documented on page 141.)

192.2 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_iow_log_stream
\c_ior_log_stream
7926 \cs_new_eq:NN \c_iow_term_stream \c_sixteen
7927 \cs_new_eq:NN \c_ior_term_stream \c_sixteen
7928 \cs_new_eq:NN \c_iow_log_stream \c_minus_one
7929 \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`

```

7930 \tl_const:Nn \c_iow_streams_tl
7931 {
7932   \c_zero
7933   \c_one
7934   \c_two
7935   \c_three
7936   \c_four
7937   \c_five
7938   \c_six
7939   \c_seven
7940   \c_eight
7941   \c_nine
7942   \c_ten
7943   \c_eleven
7944   \c_twelve
7945   \c_thirteen
7946   \c_fourteen
7947   \c_fifteen
7948 }
7949 \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.

```

7950 \prop_new:N \g_iow_streams_prop
7951 \prop_new:N \g_ior_streams_prop
7952 <*package>
7953 \prop_put:Nnn \g_iow_streams_prop { 0 } { LaTeX2e~reserved }
7954 \prop_put:Nnn \g_iow_streams_prop { 1 } { LaTeX2e~reserved }
7955 \prop_put:Nnn \g_iow_streams_prop { 2 } { LaTeX2e~reserved }
7956 \prop_put:Nnn \g_ior_streams_prop { 0 } { LaTeX2e~reserved }
7957 </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.

```

7958 \int_new:N \l_iow_stream_int
7959 \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 ??.)

192.3 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.

```
\ior_raw_new:N 7960 <*initex>
\ior_raw_new:c 7961 \alloc_setup_type:nnn { ior } \c_zero \c_sixteen
\ior_raw_new:N 7962 \cs_new_protected_nopar:Npn \ior_raw_new:N #1
\ior_raw_new:c 7963 { \alloc_reg:nNN { ior } \tex_chardef:D #1 }
7964 \alloc_setup_type:nnn { iow } \c_zero \c_sixteen
7965 \cs_new_protected_nopar:Npn \iow_raw_new:N #1
7966 { \alloc_reg:nNN { iow } \tex_chardef:D #1 }
7967 </initex>
7968 <*package>
7969 \cs_set_eq:NN \iow_raw_new:N \newwrite
7970 \cs_set_eq:NN \ior_raw_new:N \newread
7971 </package>
7972 \cs_generate_variant:Nn \ior_raw_new:N { c }
7973 \cs_generate_variant:Nn \iow_raw_new:N { c }
```

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

`\ior_new:N` Reserving a new stream is done by defining the stream as the 16 (which must be in error).

```
\ior_new:c 7974 \cs_new_protected_nopar:Npn \ior_new:N #1
\ior_new:N 7975 {
\iow_new:c 7976 \chk_if_free_cs:N #1
7977 \tex_global:D \tex_chardef:D #1 16 ~
7978 }
7979 \cs_new_eq:NN \iow_new:N \ior_new:N
7980 \cs_generate_variant:Nn \ior_new:N { c }
7981 \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.

```
7982 \cs_new_protected_nopar:Npn \ior_open:Nn #1#2
7983 {
7984 \ior_close:N #1
7985 \int_set:Nn \l_ior_stream_int \c_sixteen
7986 \tl_map_function:NN \c_ior_streams_tl \ior_alloc_read:n
7987 \int_compare:nNnTF \l_ior_stream_int = \c_sixteen
7988 { \msg_kernel_error:nn { ior } { streams-exhausted } }
7989 {
7990 \ior_stream_alloc:N #1
7991 \prop_gput:Nvn \g_ior_streams_prop \l_ior_stream_int {#2}
7992 \tex_openin:D #1#2 \scan_stop:
7993 }
7994 }
```

```

7995 \cs_new_protected_nopar:Npn \iow_open:Nn #1#2
7996 {
7997   \iow_close:N #1
7998   \int_set:Nn \l_iow_stream_int \c_sixteen
7999   \tl_map_function:NN \c_iow_streams_tl \iow_alloc_write:n
8000   \int_compare:nNnTF \l_iow_stream_int = \c_sixteen
8001     { \msg_kernel_error:nn { iow } { streams-exhausted } }
8002     {
8003       \iow_stream_alloc:N #1
8004       \prop_gput:NVn \g_iow_streams_prop \l_iow_stream_int {#2}
8005       \tex_immediate:D \tex_openout:D #1#2 \scan_stop:
8006     }
8007   }
8008   \cs_generate_variant:Nn \ior_open:Nn { c }
8009   \cs_generate_variant:Nn \iow_open:Nn { c }

```

(End definition for \ior_open:Nn and \iow_open:Nn. These functions are documented on page ??.)

\ior_alloc_read:n These functions are used to see if a particular stream is available. The property list
\iow_alloc_write:n contains file names for streams in use, so any unused ones are for the taking.

```

8010 \cs_new_protected_nopar:Npn \iow_alloc_write:n #1
8011 {
8012   \prop_if_in:NnF \g_iow_streams_prop {#1}
8013   {
8014     \int_set:Nn \l_iow_stream_int {#1}
8015     \tl_map_break:
8016   }
8017 }
8018 \cs_new_protected_nopar:Npn \ior_alloc_read:n #1
8019 {
8020   \prop_if_in:NnF \g_iow_streams_prop {#1}
8021   {
8022     \int_set:Nn \l_ior_stream_int {#1}
8023     \tl_map_break:
8024   }
8025 }

```

(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.

```

8026 \cs_new_protected_nopar:Npn \iow_stream_alloc:N #1
8027 {
8028   \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _stream }
8029     { \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _stream } }
8030     {

```

```

8031 <*package>
8032   \iow_stream_alloc_aux:
8033   \int_compare:nNnT \l_iow_stream_int = \c_sixteen
8034   {
8035     \iow_raw_new:N \g_iow_tmp_stream
8036     \int_set:Nn \l_iow_stream_int { \g_iow_tmp_stream }
8037     \cs_gset_eq:cN
8038     { g_iow_ \int_use:N \l_iow_stream_int _stream }
8039     \g_iow_tmp_stream
8040   }
8041 </package>
8042 <*initex>
8043   \iow_raw_new:c { g_iow_ \int_use:N \l_iow_stream_int _stream }
8044 </initex>
8045   \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _stream }
8046   }
8047 }
8048 <*package>
8049 \cs_new_protected_nopar:Npn \iow_stream_alloc_aux:
8050 {
8051   \int_incr:N \l_iow_stream_int
8052   \int_compare:nNnT \l_iow_stream_int < \c_sixteen
8053   {
8054     \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _stream }
8055     {
8056       \prop_if_in:NVT \g_iow_streams_prop \l_iow_stream_int
8057       { \iow_stream_alloc_aux: }
8058     }
8059     { \iow_stream_alloc_aux: }
8060   }
8061 }
8062 </package>
8063 \cs_new_protected_nopar:Npn \ior_stream_alloc:N #1
8064 {
8065   \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _stream }
8066   { \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _stream } }
8067   {
8068 <*package>
8069   \ior_stream_alloc_aux:
8070   \int_compare:nNnT \l_ior_stream_int = \c_sixteen
8071   {
8072     \ior_raw_new:N \g_ior_tmp_stream
8073     \int_set:Nn \l_ior_stream_int { \g_ior_tmp_stream }
8074     \cs_gset_eq:cN
8075     { g_ior_ \int_use:N \l_ior_stream_int _stream }
8076     \g_ior_tmp_stream
8077   }
8078 </package>
8079 <*initex>
8080   \ior_raw_new:c { g_ior_ \int_use:N \l_ior_stream_int _stream }

```

```

8081 </initex>
8082 \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _stream }
8083 }
8084 }
8085 <*package>
8086 \cs_new_protected_nopar:Npn \ior_stream_alloc_aux:
8087 {
8088   \int_incr:N \l_ior_stream_int
8089   \int_compare:nNnT \l_ior_stream_int < \c_sixteen
8090   {
8091     \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _stream }
8092     {
8093       \prop_if_in:NVT \g_ior_streams_prop \l_ior_stream_int
8094       { \ior_stream_alloc_aux: }
8095     }
8096     { \ior_stream_alloc_aux: }
8097   }
8098 }
8099 </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
8100 \cs_new_protected_nopar:Npn \ior_close:N #1
8101 {
8102   \cs_if_exist:NT #1
8103   {
8104     \int_compare:nNnF #1 = \c_minus_one
8105     {
8106       \int_compare:nNnF #1 = \c_sixteen
8107       { \tex_closein:D #1 }
8108       \prop_gdel:NV \g_ior_streams_prop #1
8109       \tex_global:D \tex_chardef:D #1 16 ~
8110     }
8111   }
8112 }
8113 \cs_new_protected_nopar:Npn \ior_close:N #1
8114 {
8115   \cs_if_exist:NT #1
8116   {
8117     \int_compare:nNnF #1 = \c_minus_one
8118     {
8119       \int_compare:nNnF #1 = \c_sixteen
8120       { \tex_closein:D #1 }
8121       \prop_gdel:NV \g_ior_streams_prop #1
8122       \tex_global:D \tex_chardef:D #1 16 ~
8123     }
8124   }

```

```

8125 }
8126 \cs_generate_variant:Nn \ior_close:N { c }
8127 \cs_generate_variant:Nn \ior_close:N { c }
      (End definition for \ior_close:N and \ior_close:c. These functions are documented on page
??.)

```

\ior_list_streams: Show the property lists, but with some “pretty printing”.

```

\ior_list_streams:
\ior_list_streams: 8128 \cs_new_protected_nopar:Npn \ior_list_streams:
\ior_show_aux:nn 8129 {
\ior_show_aux:nn 8130   \prop_if_empty:NTF \g_ior_streams_prop
8131   {
8132     \ior_term:x { No~input~streams~are~open }
8133     \tl_show:n { }
8134   }
8135   {
8136     \ior_term:x { The~following~input~streams~are~in~use: }
8137     \tl_set:Nx \l_prop_show_tl
8138     { \prop_map_function:NN \g_ior_streams_prop \ior_show_aux:nn }
8139     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
8140     { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
8141   }
8142 }
8143 \cs_new:Npn \ior_show_aux:nn #1#2
8144 {
8145   \ior_newline: > \c_space_tl \c_space_tl
8146   #1 \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
8147 }
8148 \cs_new_protected_nopar:Npn \ior_list_streams:
8149 {
8150   \prop_if_empty:NTF \g_iow_streams_prop
8151   {
8152     \ior_term:x { No~output~streams~are~open }
8153     \tl_show:n { }
8154   }
8155   {
8156     \ior_term:x { The~following~output~streams~are~in~use: }
8157     \tl_set:Nx \l_prop_show_tl
8158     { \prop_map_function:NN \g_iow_streams_prop \ior_show_aux:nn }
8159     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
8160     { \exp_after:wN \prop_show_aux:w \l_prop_show_tl }
8161   }
8162 }
8163 \cs_new_eq:NN \ior_show_aux:nn \ior_show_aux:nn
      (End definition for \ior_list_streams:. This function is documented on page ??.)
      Text for the error messages.
8164 \msg_kernel_new:nnnn { iow } { streams-exhausted }
8165 { Output~streams~exhausted }
8166 {
8167   TeX~can~only~open~up~to~16~output~streams~at~one~time.\\

```

```

8168     All~16 are currently~in~use,~and~something~wanted~to~open
8169     another~one.
8170   }
8171   \msg_kernel_new:nnnn { ior } { streams-exhausted }
8172   { Input~streams-exhausted }
8173   {
8174     TeX~can~only~open~up~to~16~input~streams~at~one~time.\\
8175     All~16 are currently~in~use,~and~something~wanted~to~open
8176     another~one.
8177   }

```

192.4 Deferred writing

`\iow_shipout_x:Nn` First the easy part, this is the primitive.

```

\iow_shipout_x:Nx
8178 \cs_new_eq:NN \iow_shipout_x:Nn \tex_write:D
8179 \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
8180 \cs_new_protected_nopar:Npn \iow_shipout:Nn #1#2
8181   { \iow_shipout_x:Nn #1 { \exp_not:n {#2} } }
8182 \cs_generate_variant:Nn \iow_shipout:Nn { Nx }
      (End definition for \iow_shipout:Nn and \iow_shipout:Nx. These functions are documented on
      page ??.)

```

192.5 Immediate writing

`\iow_now:Nx` An abbreviation for an often used operation, which immediately writes its second argument expanded to the output stream.

```

8183 \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.

```

8184 \cs_new_protected_nopar:Npn \iow_now:Nn #1#2
8185   { \iow_now:Nx #1 { \exp_not:n {#2} } }
      (End definition for \iow_now:Nn. This function is documented on page 137.)

```

`\iow_log:n` Writing to the log and the terminal directly are relatively easy.

```

\iow_log:x
8186 \cs_set_protected_nopar:Npn \iow_log:x { \iow_now:Nx \c_iow_log_stream }
\iow_term:n
8187 \cs_new_protected_nopar:Npn \iow_log:n { \iow_now:Nn \c_iow_log_stream }
\iow_term:x
8188 \cs_set_protected_nopar:Npn \iow_term:x { \iow_now:Nx \c_iow_term_stream }
8189 \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`

```

8190 \cs_new_protected_nopar:Npn \iow_now_when_avail:Nn #1
8191 { \cs_if_free:NTF #1 { \use_none:n } { \iow_now:Nn #1 } }
8192 \cs_new_protected_nopar:Npn \iow_now_when_avail:Nx #1
8193 { \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 ??.)

192.6 Special characters for writing

`\iow_newline:` Global variable holding the character that forces a new line when something is written to an output stream

```

8194 \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.

```

8195 \cs_new_eq:NN \iow_char:N \cs_to_str:N

```

(End definition for `\iow_char:N`. This function is documented on page 138.)

192.7 Hard-wrapping lines based on length

The code here implements a generic hard-wrapping function. This is used by the messaging system, but is designed such that it is available for other uses.

`\l_iow_line_length_int` The is the “raw” length of a line which can be written to a file. The standard value is the line length typically used by `TEXLive` and `MikTEX`.

```

8196 \int_new:N \l_iow_line_length_int
8197 \int_set:Nn \l_iow_line_length_int { 78 }

```

(End definition for `\l_iow_line_length_int`. This function is documented on page 139.)

`\l_iow_target_length_int` This stores the target line length: the full length minus any part for a leader at the start of each line.

```

8198 \int_new:N \l_iow_target_length_int

```

(End definition for `\l_iow_target_length_int`. This function is documented on page ??.)

`\l_iow_current_line_int` These store the number of characters in the line and word currently being constructed,
`\l_iow_current_word_int` and the current indentation, respectively.

```

\l_iow_current_indentation_int
8199 \int_new:N \l_iow_current_line_int
8200 \int_new:N \l_iow_current_word_int
8201 \int_new:N \l_iow_current_indentation_int

```

(End definition for `\l_iow_current_line_int`, `\l_iow_current_word_int`, and `\l_iow_current_indentation_int`. These functions are documented on page ??.)

`\l_iow_current_line_tl` These hold the current line of text and current word, and a number of spaces for inden-
`\l_iow_current_word_tl` tation, respectively.

```

\l_iow_current_indentation_tl
8202 \tl_new:N \l_iow_current_line_tl
8203 \tl_new:N \l_iow_current_word_tl
8204 \tl_new:N \l_iow_current_indentation_tl

```


(End definition for `\l_iow_current_line_tl`, `\l_iow_current_word_tl`, and `\l_iow_current_indentation_tl`. These functions are documented on page ??.)

`\l_iow_wrap_tl` Used for the expansion step before detokenizing.

8205 `\tl_new:N \l_iow_wrap_tl`

(End definition for `\l_iow_wrap_tl`. This function is documented on page ??.)

`\l_iow_wrapped_tl` The output from wrapping text: fully expanded and with lines which are not overly long.

8206 `\tl_new:N \l_iow_wrapped_tl`

(End definition for `\l_iow_wrapped_tl`. This function is documented on page ??.)

`\l_iow_line_start_bool` Boolean to avoid adding a space at the beginning of forced newlines.

8207 `\bool_new:N \l_iow_line_start_bool`

(End definition for `\l_iow_line_start_bool`. This function is documented on page ??.)

`\c_catcode_other_space_tl` Lowercase a character with category code 12 to produce an “other” space. We can do everything within the group, because `\tl_const:Nn` defines its argument globally.

8208 `\group_begin:`

8209 `\char_set_catcode_other:N *`

8210 `\char_set_lccode:nn {'*} {'\ }`

8211 `\tl_to_lowercase:n { \tl_const:Nn \c_catcode_other_space_tl { * } }`

8212 `\group_end:`

(End definition for `\c_catcode_other_space_tl`. This function is documented on page 139.)

`\c_iow_wrap_marker_tl` Every special action of the wrapping code is 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

`\c_iow_wrap_newline_marker_tl` is not very important, but makes `\c_iow_wrap_marker_tl` look nicer. Note that `\iow-`

`\c_iow_wrap_indent_marker_tl` `wrap_new_marker:n` does not survive the group, but all constants are defined globally.

`\iow_wrap_new_marker:n`

8213 `\group_begin:`

8214 `\int_set_eq:NN \tex_escapechar:D \c_minus_one`

8215 `\tl_const:Nx \c_iow_wrap_marker_tl`

8216 `{ \tl_to_str:n { \^^I \^^O \^^W \^^_ \^^W \^^R \^^A \^^P } }`

8217 `\cs_set:Npn \iow_wrap_new_marker:n #1`

8218 `{`

8219 `\tl_const:cx { c_iow_wrap_#1 _marker_tl }`

8220 `{`

8221 `\c_catcode_other_space_tl`

8222 `\c_iow_wrap_marker_tl`

8223 `\c_catcode_other_space_tl`

8224 `#1`

8225 `\c_catcode_other_space_tl`

8226 `}`

8227 `}`

8228 `\iow_wrap_new_marker:n { end }`

8229 `\iow_wrap_new_marker:n { newline }`

8230 `\iow_wrap_new_marker:n { indent }`

8231 `\iow_wrap_new_marker:n { unindent }`

8232 `\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.

```

8233 \cs_new_protected:Npn \iow_indent:n #1 { }
8234 \cs_new:Npx \iow_indent_expandable:n #1
8235 {
8236   \c_iow_wrap_indent_marker_tl
8237   #1
8238   \c_iow_wrap_unindent_marker_tl
8239 }

```

(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 T_EX to end a number or other f-type expansions. The \tl_to_str:N step converts the “other” space back to a normal space.

```

8240 \cs_new_protected:Npn \iow_wrap:xnnnN #1#2#3#4#5
8241 {
8242   \group_begin:
8243     \int_set:Nn \l_iow_target_length_int { \l_iow_line_length_int - ( #3 ) }
8244     \int_zero:N \l_iow_current_indentation_int
8245     \tl_clear:N \l_iow_current_indentation_tl
8246     \int_zero:N \l_iow_current_line_int
8247     \tl_clear:N \l_iow_current_line_tl
8248     \tl_clear:N \l_iow_wrap_tl
8249     \bool_set_true:N \l_iow_line_start_bool
8250     \int_set_eq:NN \tex_escapechar:D \c_minus_one
8251     \cs_set_nopar:Npx \{ { \token_to_str:N \{ }
8252     \cs_set_nopar:Npx \# { \token_to_str:N \# }
8253     \cs_set_nopar:Npx \} { \token_to_str:N \} }
8254     \cs_set_nopar:Npx \% { \token_to_str:N \% }
8255     \cs_set_nopar:Npx \~ { \token_to_str:N \~ }
8256     \int_set:Nn \tex_escapechar:D { 92 }
8257     \cs_set_eq:NN \ \c_iow_wrap_newline_marker_tl
8258     \cs_set_eq:NN \ \c_catcode_other_space_tl
8259     \cs_set_eq:NN \iow_indent:n \iow_indent_expandable:n
8260     #4
8261     \*initex
8262     \tl_set:Nx \l_iow_wrap_tl {#1}
8263     \*initex
8264     \*package

```

```

8265 \protected@edef \l_iow_wrap_tl {#1}
8266 </package>
8267 \cs_set:Npn \l { \iow_newline: #2 }
8268 \use:x
8269 {
8270   \iow_wrap_loop:w
8271   \tl_to_str:N \l_iow_wrap_tl
8272   \tl_to_str:N \c_iow_wrap_end_marker_tl
8273   \c_space_tl \c_space_tl
8274   \exp_not:N \q_stop
8275 }
8276 \exp_args:NNo \group_end:
8277 #5 \l_iow_wrapped_tl
8278 }

```

(End definition for \iow_wrap:xnnnN. This function is documented on page 139.)

\iow_wrap_loop:w The loop grabs one word in the input, and checks whether it is the special marker, or a normal word.

```

8279 \cs_new_protected:Npn \iow_wrap_loop:w #1 ~ %
8280 {
8281   \tl_set:Nn \l_iow_current_word_tl {#1}
8282   \tl_if_eq:NNTF \l_iow_current_word_tl \c_iow_wrap_marker_tl
8283   { \iow_wrap_special:w }
8284   { \iow_wrap_word: }
8285 }

```

(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, add it to the line, preceded by a space unless it is the first word of the line. Otherwise, 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.

```

8286 \cs_new_protected_nopar:Npn \iow_wrap_word:
8287 {
8288   \int_set:Nn \l_iow_current_word_int
8289   { \str_length_skip_spaces:N \l_iow_current_word_tl }
8290   \int_add:Nn \l_iow_current_line_int { \l_iow_current_word_int }
8291   \int_compare:nNnTF \l_iow_current_line_int < \l_iow_target_length_int
8292   { \iow_wrap_word_fits: }
8293   { \iow_wrap_word_newline: }
8294   \iow_wrap_loop:w
8295 }
8296 \cs_new_protected_nopar:Npn \iow_wrap_word_fits:
8297 {
8298   \bool_if:NNTF \l_iow_line_start_bool
8299   {
8300     \bool_set_false:N \l_iow_line_start_bool
8301     \tl_put_right:Nx \l_iow_current_line_tl
8302     { \l_iow_current_indentation_tl \l_iow_current_word_tl }

```

```

8303     \int_add:Nn \l_iow_current_line_int
8304     { \l_iow_current_indentation_int }
8305   }
8306   {
8307     \tl_put_right:Nx \l_iow_current_line_tl
8308     { ~ \l_iow_current_word_tl }
8309     \int_incr:N \l_iow_current_line_int
8310   }
8311 }
8312 \cs_new_protected_nopar:Npn \iow_wrap_word_newline:
8313 {
8314   \tl_put_right:Nx \l_iow_wrapped_tl
8315   { \l_iow_current_line_tl \\ }
8316   \int_set:Nn \l_iow_current_line_int
8317   {
8318     \l_iow_current_word_int
8319     + \l_iow_current_indentation_int
8320   }
8321   \tl_set:Nx \l_iow_current_line_tl
8322   { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8323 }

```

(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. `\iow_wrap_indent:w` To indent more, add four spaces to the start of the indentation token list. `\iow_wrap_unindent:w` To reduce indentation, rebuild the indentation token list using `\prg_replicate:nn`. `\iow_wrap_end:w` At the end, we simply save the last line (without the run-on text), and prevent the loop.

```

8324 \cs_new_protected_nopar:Npn \iow_wrap_special:w #1 ~ #2 ~ #3 ~ %
8325 {
8326   \cs_if_exist:cTF { iow_wrap_#1: }
8327   { \use:c { iow_wrap_#1: } }
8328   { \msg_expandable_error:n {#1} }
8329   \str_if_eq:xxTF { #2~#3 } { ~ \c_iow_wrap_marker_tl }
8330   { \iow_wrap_special:w }
8331   { \iow_wrap_loop:w #2 ~ #3 ~ }
8332 }
8333 \cs_new_protected_nopar:Npn \iow_wrap_newline:
8334 {
8335   \tl_put_right:Nx \l_iow_wrapped_tl
8336   { \l_iow_current_line_tl \\ }
8337   \int_zero:N \l_iow_current_line_int
8338   \tl_clear:N \l_iow_current_line_tl
8339   \bool_set_true:N \l_iow_line_start_bool
8340 }
8341 \cs_new_protected_nopar:Npx \iow_wrap_indent:
8342 {

```

```

8343 \int_add:Nn \l_iow_current_indentation_int \c_four
8344 \tl_put_right:Nx \exp_not:N \l_iow_current_indentation_tl
8345 { \c_space_tl \c_space_tl \c_space_tl \c_space_tl }
8346 }
8347 \cs_new_protected_nopar:Npn \iow_wrap_unindent:
8348 {
8349 \int_sub:Nn \l_iow_current_indentation_int \c_four
8350 \tl_set:Nx \l_iow_current_indentation_tl
8351 { \prg_replicate:nn \l_iow_current_indentation_int { ~ } }
8352 }
8353 \cs_new_protected_nopar:Npn \iow_wrap_end:
8354 {
8355 \tl_put_right:Nx \l_iow_wrapped_tl
8356 { \l_iow_current_line_tl }
8357 \use_none_delimit_by_q_stop:w
8358 }

```

(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.

```

8359 \cs_new_nopar:Npn \str_length_skip_spaces:N
8360 { \exp_args:No \str_length_skip_spaces:n }
8361 \cs_new:Npn \str_length_skip_spaces:n #1
8362 {
8363 \int_value:w \int_eval:w
8364 \exp_after:wN \str_length_loop:NNNNNNNNN \tl_to_str:n {#1}
8365 {X8}{X7}{X6}{X5}{X4}{X3}{X2}{X1}{X0} \q_stop
8366 \int_eval_end:
8367 }
8368 \cs_new:Npn \str_length_loop:NNNNNNNNN #1#2#3#4#5#6#7#8#9
8369 {
8370 \if_catcode:w X #9
8371 \exp_after:wN \use_none_delimit_by_q_stop:w
8372 \else:
8373 9 +
8374 \exp_after:wN \str_length_loop:NNNNNNNNN
8375 \fi:
8376 }

```

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

192.8 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.

```

8377 \prg_new_conditional:Nnn \ior_if_eof:N { p , T , F , TF }
8378 {
8379 \cs_if_exist:NTF #1
8380 {

```

```

8381         \if_int_compare:w #1 = \c_sixteen
8382         \prg_return_true:
8383     \else:
8384         \if_eof:w #1
8385         \prg_return_true:
8386     \else:
8387         \prg_return_false:
8388     \fi:
8389 \fi:
8390 }
8391 { \prg_return_true: }
8392 }

```

(End definition for `\ior_if_eof_p:N`. This function is documented on page 141.)

`\ior_to:NN` And here we read from files.

`\ior_gto:NN`

```

8393 \cs_new_protected_nopar:Npn \ior_to:NN #1#2
8394 { \tex_read:D #1 to #2 }
8395 \cs_new_protected_nopar:Npn \ior_gto:NN #1#2
8396 { \tex_global:D \tex_read:D #1 to #2 }

```

(End definition for `\ior_to:NN`. This function is documented on page 140.)

`\ior_str_to:NN` Reading as strings is also a primitive wrapper.

`\ior_str_gto:NN`

```

8397 \cs_new_protected_nopar:Npn \ior_str_to:NN #1#2
8398 { \etex_readline:D #1 to #2 }
8399 \cs_new_protected_nopar:Npn \ior_str_gto:NN #1#2
8400 { \tex_global:D \etex_readline:D #1 to #2 }

```

(End definition for `\ior_str_to:NN`. This function is documented on page 140.)

192.9 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.

```

8401 \*deprecated
8402 \cs_new_protected:Npn \iow_now_buffer_safe:Nn #1#2
8403 { \iow_wrap:xnnnN { \exp_not:n {#2} } { } \c_zero { } \iow_now:Nn #1 }
8404 \cs_new_protected:Npn \iow_now_buffer_safe:Nx #1#2
8405 { \iow_wrap:xnnnN {#2} { } \c_zero { } \iow_now:Nn #1 }
8406 \*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.

`\iow_open_streams:`

```

8407 \*deprecated
8408 \cs_new_eq:NN \ior_open_streams: \ior_list_streams:
8409 \cs_new_eq:NN \iow_open_streams: \iow_list_streams:
8410 \*deprecated

```

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

```

8411 \*initex | package

```

193 l3msg implementation

```

8412 <*initex | package>
8413 <*package>
8414 \ProvidesExplPackage
8415   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
8416   \package_check_loaded_expl:
8417 </package>

\l_msg_tmp_tl A general scratch for the module.

8418 \tl_new:N \l_msg_tmp_tl
      (End definition for \l_msg_tmp_tl. This function is documented on page ??.)

```

194 Creating messages

Messages are created and used separately, so there two parts to the code here. First, a mechanism for creating message text. This is pretty simple, as there is not actually a lot to do.

```

\c_msg_text_prefix_tl Locations for the text of messages.
\c_msg_more_text_prefix_tl
8419 \tl_const:Nn \c_msg_text_prefix_tl { msg~text~>~ }
8420 \tl_const:Nn \c_msg_more_text_prefix_tl { msg~extra~text~>~ }
      (End definition for \c_msg_text_prefix_tl and \c_msg_more_text_prefix_tl. These functions
      are documented on page ??.)

\msg_new:nnnn Setting a message simply means saving the appropriate text into two functions. A sanity
\msg_new:nnn check first.
\msg_gset:nnnn 8421 \cs_new_protected:Npn \msg_new:nnnn #1#2
\msg_gset:nnn 8422 {
\msg_set:nnnn 8423   \cs_if_exist:cT { \c_msg_text_prefix_tl #1 / #2 }
\msg_set:nnn 8424   {
8425     \msg_kernel_error:nn { msg } { message-already-defined }
8426     {#1} {#2}
8427   }
8428   \msg_gset:nnnn {#1} {#2}
8429 }
8430 \cs_new_protected:Npn \msg_new:nnn #1#2#3
8431 { \msg_new:nnnn {#1} {#2} {#3} { } }
8432 \cs_new_protected:Npn \msg_set:nnnn #1#2#3#4
8433 {
8434   \cs_set:cpn { \c_msg_text_prefix_tl #1 / #2 }
8435   ##1##2##3##4 {#3}
8436   \cs_set:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8437   ##1##2##3##4 {#4}
8438 }
8439 \cs_new_protected:Npn \msg_set:nnn #1#2#3
8440 { \msg_set:nnnn {#1} {#2} {#3} { } }
8441 \cs_new_protected:Npn \msg_gset:nnnn #1#2#3#4

```

```

8442 {
8443   \cs_gset:cpn { \c_msg_text_prefix_tl #1 / #2 }
8444     ##1##2##3##4 {#3}
8445   \cs_gset:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8446     ##1##2##3##4 {#4}
8447 }
8448 \cs_new_protected:Npn \msg_gset:nnn #1#2#3
8449 { \msg_gset:nnnn {#1} {#2} {#3} { } }

```

(End definition for \msg_new:nnnn and \msg_new:nnn. These functions are documented on page ??.)

194.1 Messages: support functions and text

```

\c_msg_coding_error_text_tl Simple pieces of text for messages.
\c_msg_continue_text_tl      8450 \tl_const:Nn \c_msg_coding_error_text_tl
\c_msg_critical_text_tl      8451 {
\c_msg_fatal_text_tl          8452   This-is-a-coding-error.
\c_msg_help_text_tl           8453   \\\ \\
\c_msg_no_info_text_tl        8454 }
\c_msg_on_line_text_tl         8455 \tl_const:Nn \c_msg_continue_text_tl
\c_msg_return_text_tl          8456 { Type~<return>~to~continue }
\c_msg_trouble_text_tl         8457 \tl_const:Nn \c_msg_critical_text_tl
                                8458 { Reading~the~current~file~will~stop }
                                8459 \tl_const:Nn \c_msg_fatal_text_tl
                                8460 { This-is-a-fatal-error:~LaTeX~will~abort }
                                8461 \tl_const:Nn \c_msg_help_text_tl
                                8462 { For~immediate~help~type~H~<return> }
                                8463 \tl_const:Nn \c_msg_no_info_text_tl
                                8464 {
                                8465   LaTeX~does~not~know~anything~more~about~this~error,~sorry.
                                8466   \c_msg_return_text_tl
                                8467 }
                                8468 \tl_const:Nn \c_msg_on_line_text_tl { on-line }
                                8469 \tl_const:Nn \c_msg_return_text_tl
                                8470 {
                                8471   \\\ \\
                                8472   Try~typing~<return>~to~proceed.
                                8473   \\\
                                8474   If~that~doesn't~work,~type~X~<return>~to~quit.
                                8475 }
                                8476 \tl_const:Nn \c_msg_trouble_text_tl
                                8477 {
                                8478   \\\ \\
                                8479   More~errors~will~almost~certainly~follow: \\\
                                8480   the~LaTeX~run~should~be~aborted.
                                8481 }

```

(End definition for \c_msg_coding_error_text_tl and others. These functions are documented on page 143.)

`\msg_newline:` New lines are printed in the same way as for low-level file writing.
`\msg_two_newlines:`

```

8482 \cs_new_nopar:Npn \msg_newline: { ^^J }
8483 \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:`

```

8484 \cs_new_nopar:Npn \msg_line_number: { \int_use:N \tex_inputlineno:D }
8485 \cs_set_nopar:Npn \msg_line_context:
8486 {
8487   \c_msg_on_line_text_tl
8488   \c_space_tl
8489   \msg_line_number:
8490 }

```

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

194.2 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.

```

8491 \char_set_catcode_letter:N \.
8492 \tl_new:N
8493   \c_msg_hide_tl.....
8494 \tl_const:Nn \c_msg_hide_tl
8495   { \c_msg_hide_tl..... }
8496 \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.

```

8497 \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.

```

8498 \cs_new_protected:Npn \msg_interrupt:xxx #1#2#3
8499 {
8500   \group_begin:
8501     \tl_if_empty:nTF {#3}
8502       { \msg_interrupt_no_details:xx {#1} {#2} }
8503       { \msg_interrupt_details:xxx {#1} {#2} {#3} }
8504   \msg_interrupt_aux:
8505   \group_end:
8506 }

```

```

8507 % Depending on the availability of more information there is a choice of
8508 % how to set up the further help. The extra help text has to be set
8509 % before the message itself can be issued. Everything is done using
8510 % \texttt{x}-type expansion as the new line markers are different for
8511 % the two type of text and need to be correctly set up.
8512 % \begin{macrocode}
8513 \cs_new_protected:Npn \msg_interrupt_no_details:xx #1#2
8514 {
8515   \iow_wrap:xnnnN
8516   { \ \ \c_msg_no_info_text_tl }
8517   { |~ } { 2 } { } \msg_interrupt_more_text:n
8518   \iow_wrap:xnnnN { #1 \ \ \ #2 \ \ \ \c_msg_continue_text_tl }
8519   { ! ~ } { 2 } { } \msg_interrupt_text:n
8520 }
8521 \cs_new_protected:Npn \msg_interrupt_details:xxx #1#2#3
8522 {
8523   \iow_wrap:xnnnN
8524   { \ \ \ #3 }
8525   { |~ } { 2 } { } \msg_interrupt_more_text:n
8526   \iow_wrap:xnnnN { #1 \ \ \ #2 \ \ \ \c_msg_help_text_tl }
8527   { ! ~ } { 2 } { } \msg_interrupt_text:n
8528 }
8529 \cs_new_protected:Npn \msg_interrupt_text:n #1
8530 { \tl_set:Nn \l_msg_text_tl {#1} }
8531 \cs_new_protected:Npn \msg_interrupt_more_text:n #1
8532 {
8533   <*initex>
8534   \tl_set:Nx \l_msg_tmp_tl
8535   </initex>
8536   <*package>
8537   \protected@edef \l_msg_tmp_tl
8538   </package>
8539   {
8540     |,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
8541     #1
8542     \msg_newline:
8543     |.....
8544   }
8545   \tex_errhelp:D \exp_after:wN { \l_msg_tmp_tl }
8546 }

```

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.

```

8547 \group_begin: % {
8548   \char_set_lccode:nn {'\} } {'\ }
8549   \char_set_lccode:nn {'&} {'\!}
8550   \char_set_catcode_active:N \&

```

```

8551 \tl_to_lowercase:n
8552 {
8553   \group_end:
8554   \cs_new_protected:Npn \msg_interrupt_aux:
8555   {
8556     \iow_term:x
8557     {
8558       \iow_newline:
8559       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
8560       \iow_newline:
8561       !
8562     }
8563     \tl_put_right:No \l_msg_text_tl { \c_msg_hide_tl }
8564     \cs_set_protected_nopar:Npx &
8565     { \tex_errmessage:D { \exp_not:o { \l_msg_text_tl } } }
8566     &
8567   }
8568 }

```

(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.

```

8569 \cs_new_protected:Npn \msg_log:x #1
8570 {
8571   \iow_log:x { ..... }
8572   \iow_wrap:xnnnN { . ~ #1 } { . ~ } { 2 } { }
8573   \iow_log:x
8574   \iow_log:x { ..... }
8575 }
8576 \cs_new_protected:Npn \msg_term:x #1
8577 {
8578   \iow_term:x { ***** }
8579   \iow_wrap:xnnnN { * ~ #1 } { * ~ } { 2 } { }
8580   \iow_term:x
8581   \iow_term:x { ***** }
8582 }

```

(End definition for \msg_log:x. This function is documented on page 147.)

194.3 Displaying messages

L^AT_EX is handling error messages and so the T_EX ones are disabled.

```

8583 \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 8584 \cs_new_nopar:Npn \msg_fatal_text:n #1 { Fatal~#1~error }
\msg_error_text:n    8585 \cs_new_nopar:Npn \msg_critical_text:n #1 { Critical~#1~error }
\msg_warning_text:n  8586 \cs_new_nopar:Npn \msg_error_text:n #1 { #1~error }
\msg_info_text:n     8587 \cs_new_nopar:Npn \msg_warning_text:n #1 { #1~warning }
                    8588 \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 144.)

`\msg_see_documentation_text:n` Contextual footer information.

```
8589 \cs_new_nopar:Npn \msg_see_documentation_text:n #1
8590 { \ \ \ See~the~#1~documentation~for~further~information. }
```

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

`\l_msg_redirect_classes_prop` For filtering messages, a list of all messages and of those which have to be modified is required.
`\l_msg_redirect_names_prop`

```
8591 \prop_new:N \l_msg_redirect_classes_prop
8592 \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.

```
8593 \cs_new_protected_nopar:Npn \msg_class_set:nn #1#2
8594 {
8595   \prop_clear_new:c { l_msg_redirect_ #1 _prop }
8596   \cs_set_protected:cpn { msg_ #1 :nnxxxx } ##1##2##3##4##5##6
8597   { \msg_use:nnnnxxxx {#1} {#2} {##1} {##2} {##3} {##4} {##5} {##6} }
8598   \cs_set_protected:cpx { msg_ #1 :nnxxx } ##1##2##3##4##5
8599   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} {##5} { } }
8600   \cs_set_protected:cpx { msg_ #1 :nnxx } ##1##2##3##4
8601   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} { } { } }
8602   \cs_set_protected:cpx { msg_ #1 :nnx } ##1##2##3
8603   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} { } { } { } }
8604   \cs_set_protected:cpx { msg_ #1 :nn } ##1##2
8605   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} { } { } { } { } }
8606 }
```

(End definition for `\msg_class_set:nn`. This function is documented on page 144.)

`\msg_if_more_text:N` A test to see if any more text is available, using a permanently-empty text function.

```
\msg_if_more_text:c
\msg_no_more_text:xxxx
8607 \prg_set_conditional:Npnn \msg_if_more_text:N #1 { p , T , F , TF }
8608 {
8609   \cs_if_eq:NNTF #1 \msg_no_more_text:xxxx
8610   { \prg_return_false: }
8611   { \prg_return_true: }
8612 }
8613 \cs_new:Npn \msg_no_more_text:xxxx #1#2#3#4 { }
8614 \cs_generate_variant:Nn \msg_if_more_text_p:N { c }
8615 \cs_generate_variant:Nn \msg_if_more_text:NT { c }
8616 \cs_generate_variant:Nn \msg_if_more_text:NF { c }
8617 \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:nnxxx      8618 \msg_class_set:nn { fatal }
\msg_fatal:nnxx      8619 {
\msg_fatal:nnx      8620   \msg_interrupt:xxx
\msg_fatal:nn      8621   { \msg_fatal_text:n {#1} : ~ "#2" }
      8622   {
      8623     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
      8624     \msg_see_documentation_text:n {#1}
      8625   }
      8626   { \c_msg_fatal_text_tl }
      8627   \tex_end:D
      8628 }

```

(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  8629 \msg_class_set:nn { critical }
\msg_critical:nnxx  8630 {
\msg_critical:nnx  8631   \msg_interrupt:xxx
\msg_critical:nn  8632   { \msg_critical_text:n {#1} : ~ "#2" }
      8633   {
      8634     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
      8635     \msg_see_documentation_text:n {#1}
      8636   }
      8637   { \c_msg_critical_text_tl }
      8638   \tex_endinput:D
      8639 }

```

(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      8640 \msg_class_set:nn { error }
\msg_error:nnxx      8641 {
\msg_error:nnx      8642   \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl #1 / #2 }
\msg_error:nn      8643   {
      8644     \msg_interrupt:xxx
      8645     { \msg_error_text:n {#1} : ~ "#2" }
      8646     {
      8647       \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
      8648       \msg_see_documentation_text:n {#1}
      8649     }
      8650     { \use:c { \c_msg_more_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
      8651   }
      8652   {
      8653     \msg_interrupt:xxx
      8654     { \msg_error_text:n {#1} : ~ "#2" }
      8655     {
      8656       \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
      8657       \msg_see_documentation_text:n {#1}
      8658     }

```

```

8659         { }
8660     }
8661 }

```

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

\msg_warning:nnxxxx Warnings are printed to the terminal.

```

\msg_warning:nnxxxx 8662 \msg_class_set:nn { warning }
\msg_warning:nnxxx 8663 {
\msg_warning:nnxx 8664     \msg_term:x
\msg_warning:nn 8665     {
8666         \msg_warning_text:n {#1} : ~ "#2" \\ \\
8667         \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8668     }
8669 }

```

(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 8670 \msg_class_set:nn { info }
\msg_info:nnxxx 8671 {
\msg_info:nnxx 8672     \msg_log:x
\msg_info:nn 8673     {
8674         \msg_info_text:n {#1} : ~ "#2" \\ \\
8675         \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8676     }
8677 }

```

(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 8678 \msg_class_set:nn { log }
\msg_log:nnxxx 8679 {
\msg_log:nnxx 8680     \msg_log:x
\msg_log:nn 8681     { \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8682 }

```

(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 8683 \msg_class_set:nn { none } { }

```

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

\l_msg_redirect\l_msg_redirect_classes_seq Support variables needed for the redirection system.

```

\l_msg_redirect\l_msg_redirect_classes_seq 8684 \seq_new:N \l_msg_redirect_classes_seq
\l_msg_class_tl 8685 \tl_new:N \l_msg_class_tl
\l_msg_current_class_tl 8686 \tl_new:N \l_msg_current_class_tl
\l_msg_current_module_tl 8687 \tl_new:N \l_msg_current_module_tl

```

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

`\msg_use:nnnnxxxx` The main message-using macro creates two auxiliary functions: one containing the code
`\msg_use_aux:nnn` for the message, and the second a loop function. There is then a hand-off to the system
`\msg_use_aux:nn` for checking if redirection is needed.
`\msg_use_loop_check:nn`
`\msg_use_code:`
`\msg_use_loop:n`
`\msg_use_loop:o`

```

8688 \cs_new_protected:Npn \msg_use:nnnnxxxx #1#2#3#4#5#6#7#8
8689 {
8690   \cs_set_protected_nopar:Npx \msg_use_code:
8691   {
8692     \seq_clear:N \exp_not:N \l_msg_redirect_classes_seq
8693     \exp_not:n {#2}
8694   }
8695   \cs_set_protected:Npx \msg_use_loop:n ##1
8696   {
8697     \seq_if_in:NnTF \exp_not:n \l_msg_redirect_classes_seq {#1}
8698     { \msg_kernel_error:nn { msg } { message-loop } {#1} }
8699     {
8700       \seq_put_right:Nn \exp_not:N \l_msg_redirect_classes_seq {#1}
8701       \exp_not:N \cs_if_exist:cTF { msg_ ##1 :nnxxxx }
8702       {
8703         \exp_not:N \use:c { msg_ ##1 :nnxxxx }
8704         \exp_not:n { {#3} {#4} {#5} {#6} {#7} {#8} }
8705       }
8706       {
8707         \msg_kernel_error:nnx { msg } { message-class-unknown } {##1}
8708       }
8709     }
8710   }
8711   \cs_if_exist:cTF { \c_msg_text_prefix_tl #3 / #4 }
8712   { \msg_use_aux:nnn {#1} {#3} {#4} }
8713   { \msg_kernel_error:nnxx { msg } { message-unknown } {#3} {#4} }
8714 }

```

The first auxiliary macro looks for a match by name: the most restrictive check.

```

8715 \cs_new_protected_nopar:Npn \msg_use_aux:nnn #1#2#3
8716 {
8717   \tl_set:Nn \l_msg_current_class_tl {#1}
8718   \tl_set:Nn \l_msg_current_module_tl {#2}
8719   \prop_if_in:NnTF \l_msg_redirect_names_prop { // #2 / #3 / }
8720   { \msg_use_loop_check:nn { names } { // #2 / #3 / } }
8721   { \msg_use_aux:nn {#1} {#2} }
8722 }

```

The second function checks for general matches by module or for all modules.

```

8723 \cs_new_protected_nopar:Npn \msg_use_aux:nn #1#2
8724 {
8725   \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } {#2}
8726   { \msg_use_loop_check:nn {#1} {#2} }
8727   {
8728     \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } { * }
8729     { \msg_use_loop_check:nn {#1} { * } }
8730     { \msg_use_code: }

```

```

8731     }
8732 }

```

When checking whether to loop, the same code is needed in a few places.

```

8733 \cs_new_protected:Npn \msg_use_loop_check:nn #1#2
8734 {
8735     \prop_get:cnN { l_msg_redirect_ #1 _prop } {#2} \l_msg_class_tl
8736     \tl_if_eq:NNTF \l_msg_current_class_tl \l_msg_class_tl
8737     {
8738         { \msg_use_code: }
8739         { \msg_use_loop:o \l_msg_class_tl }
8740     }
8741 }
8742 \cs_new_protected_nopar:Npn \msg_use_code: { }
8743 \cs_new_protected:Npn \msg_use_loop:n #1 { }
8744 \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.

```

8745 \cs_new_protected_nopar:Npn \msg_redirect_class:nn #1#2
8746 { \prop_put:cnn { l_msg_redirect_ #1 _prop } { * } {#2} }

```

(End definition for \msg_redirect_class:nn. This function is documented on page 146.)

`\msg_redirect_module:nnn` For when all messages of a class should be altered for a given module.

```

8747 \cs_new_protected_nopar:Npn \msg_redirect_module:nnn #1#2#3
8748 { \prop_put:cnn { l_msg_redirect_ #2 _prop } {#1} {#3} }

```

(End definition for \msg_redirect_module:nnn. This function is documented on page 146.)

`\msg_redirect_name:nnn` Named message will always use the given class.

```

8749 \cs_new_protected_nopar:Npn \msg_redirect_name:nnn #1#2#3
8750 { \prop_put:Nnn \l_msg_redirect_names_prop { // #1 / #2 / } {#3} }

```

(End definition for \msg_redirect_name:nnn. This function is documented on page 146.)

194.4 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
`\msg_kernel_set:nnnn` part.
`\msg_kernel_set:nnn`

```

8751 \cs_new_protected_nopar:Npn \msg_kernel_new:nnnn #1#2
8752 { \msg_new:nnnn { LaTeX } { #1 / #2 } }
8753 \cs_new_protected_nopar:Npn \msg_kernel_new:nnn #1#2
8754 { \msg_new:nnn { LaTeX } { #1 / #2 } }
8755 \cs_new_protected_nopar:Npn \msg_kernel_set:nnnn #1#2
8756 { \msg_set:nnnn { LaTeX } { #1 / #2 } }
8757 \cs_new_protected_nopar:Npn \msg_kernel_set:nnn #1#2
8758 { \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 8759 \cs_new_protected:Npn \msg_kernel_fatal:nnxxxx #1#2#3#4#5#6
\msg_kernel_fatal:nnxx 8760 {
\msg_kernel_fatal:nnx 8761 \msg_interrupt:xxx
\msg_kernel_fatal:nn 8762 { \msg_fatal_text:n { LaTeX } : ~ "#1 / #2" }
8763 {
8764 \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8765 {#3} {#4} {#5} {#6}
8766 \msg_see_documentation_text:n { LaTeX3 }
8767 }
8768 { \c_msg_fatal_text_tl }
8769 \tex_end:D
8770 }
8771 \cs_new_protected:Npn \msg_kernel_fatal:nnxxx #1#2#3#4#5
8772 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8773 \cs_new_protected:Npn \msg_kernel_fatal:nnxx #1#2#3#4
8774 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8775 \cs_new_protected:Npn \msg_kernel_fatal:nnx #1#2#3
8776 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} { } { } { } }
8777 \cs_new_protected:Npn \msg_kernel_fatal:nn #1#2
8778 { \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 8779 \cs_new_protected:Npn \msg_kernel_error:nnxxxx #1#2#3#4#5#6
\msg_kernel_error:nnxx 8780 {
\msg_kernel_error:nnx 8781 \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
\msg_kernel_error:nn 8782 {
8783 \msg_interrupt:xxx
8784 { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8785 {
8786 \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8787 {#3} {#4} {#5} {#6}
8788 \msg_see_documentation_text:n { LaTeX3 }
8789 }
8790 {
8791 \use:c { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
8792 {#3} {#4} {#5} {#6}
8793 }
8794 }
8795 {
8796 \msg_interrupt:xxx
8797 { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8798 {
8799 \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8800 {#3} {#4} {#5} {#6}
8801 \msg_see_documentation_text:n { LaTeX3 }
8802 }
8803 { }

```

```

8804     }
8805   }
8806   \cs_new_protected:Npn \msg_kernel_error:nnxxx #1#2#3#4#5
8807   { \msg_kernel_error:nnxxx {#1} {#2} {#3} {#4} {#5} { } }
8808   \cs_set_protected:Npn \msg_kernel_error:nnxx #1#2#3#4
8809   { \msg_kernel_error:nnxxx {#1} {#2} {#3} {#4} { } { } }
8810   \cs_set_protected:Npn \msg_kernel_error:nnx #1#2#3
8811   { \msg_kernel_error:nnxxx {#1} {#2} {#3} { } { } { } }
8812   \cs_set_protected:Npn \msg_kernel_error:nn #1#2
8813   { \msg_kernel_error:nnxxx {#1} {#2} { } { } { } { } }

```

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

\msg_kernel_warning:nnxxx Kernel messages which can be redirected.

```

\msg_kernel_warning:nnxxx
\msg_kernel_warning:nnxxx 8814 \prop_new:N \l_msg_redirect_kernel_warning_prop
\msg_kernel_warning:nnxx 8815 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5#6
\msg_kernel_warning:nnx 8816 {
\msg_kernel_warning:nn 8817   \msg_use:nnnnxxxx { warning }
\msg_kernel_info:nnxxx 8818   {
\msg_kernel_info:nnxxx 8819     \msg_term:x
\msg_kernel_info:nnxx 8820     {
\msg_kernel_info:nnxx 8821       \msg_warning_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
\msg_kernel_info:nnx 8822       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
\msg_kernel_info:nn 8823       {#3} {#4} {#5} {#6}
8824     }
8825   }
8826   { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
8827 }
8828 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5
8829 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} {#4} {#5} { } }
8830 \cs_new_protected:Npn \msg_kernel_warning:nnxx #1#2#3#4
8831 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} {#4} { } { } }
8832 \cs_new_protected:Npn \msg_kernel_warning:nnx #1#2#3
8833 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} { } { } { } }
8834 \cs_new_protected:Npn \msg_kernel_warning:nn #1#2
8835 { \msg_kernel_warning:nnxxx {#1} {#2} { } { } { } { } }
8836 \prop_new:N \l_msg_redirect_kernel_info_prop
8837 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5#6
8838 {
8839   \msg_use:nnnnxxxx { info }
8840   {
8841     \msg_log:x
8842     {
8843       \msg_info_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
8844       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8845       {#3} {#4} {#5} {#6}
8846     }
8847   }
8848   { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
8849 }
8850 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5

```

```

8851 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8852 \cs_new_protected:Npn \msg_kernel_info:nnxx #1#2#3#4
8853 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8854 \cs_new_protected:Npn \msg_kernel_info:nnx #1#2#3
8855 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} { } { } { } }
8856 \cs_new_protected:Npn \msg_kernel_info:nn #1#2
8857 { \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.
8858 \msg_kernel_new:nnnn { msg } { message-already-defined }
8859 { Message~'#2'~for~module~'#1'~already-defined. }
8860 {
8861   \c_msg_coding_error_text_tl
8862   LaTeX~was~asked~to~define~a~new~message~called~'#2'
8863   by~the~module~'#1'~module:\\
8864   this~message~already~exists.
8865   \c_msg_return_text_tl
8866 }
8867 \msg_kernel_new:nnnn { msg } { message-unknown }
8868 { Unknown~message~'#2'~for~module~'#1'. }
8869 {
8870   \c_msg_coding_error_text_tl
8871   LaTeX~was~asked~to~display~a~message~called~'#2'\\
8872   by~the~module~'#1'~module::~this~message~does~not~exist.
8873   \c_msg_return_text_tl
8874 }
8875 \msg_kernel_new:nnnn { msg } { message-class-unknown }
8876 { Unknown~message~class~'#1'. }
8877 {
8878   LaTeX~has~been~asked~to~redirect~messages~to~a~class~'#1':\\
8879   this~was~never~defined.
8880   \c_msg_return_text_tl
8881 }
8882 \msg_kernel_new:nnnn { msg } { redirect-loop }
8883 { Message~redirection~loop~for~message~class~'#1'. }
8884 {
8885   LaTeX~has~been~asked~to~redirect~messages~in~an~infinite~loop.\\
8886   The~original~message~here~has~been~lost.
8887   \c_msg_return_text_tl
8888 }

```

Messages for earlier kernel modules.

```

8889 \msg_kernel_new:nnnn { kernel } { bad-number-of-arguments }
8890 { Function~'#1'~cannot~be~defined~with~#2~arguments. }
8891 {
8892   \c_msg_coding_error_text_tl
8893   LaTeX~has~been~asked~to~define~a~function~'#1'~with~
8894   #2~arguments. \\
8895   TeX~allows~between~0~and~9~arguments~for~a~single~function.
8896 }

```

```

8897 \msg_kernel_new:nnnn { kernel } { command-already-defined }
8898 { Control~sequence~#1~already~defined. }
8899 {
8900   \c_msg_coding_error_text_tl
8901   LaTeX~has~been~asked~to~create~a~new~control~sequence~'~#1~'~
8902   but~this~name~has~already~been~used~elsewhere. \\ \\
8903   The~current~meaning~is:\\
8904   \ \ #2
8905 }
8906 \msg_kernel_new:nnnn { kernel } { command-not-defined }
8907 { Control~sequence~#1~undefined. }
8908 {
8909   \c_msg_coding_error_text_tl
8910   LaTeX~has~been~asked~to~use~a~command~#1,~but~this~has~not~
8911   been~defined~yet.
8912 }
8913 \msg_kernel_new:nnnn { kernel } { variable-not-defined }
8914 { Variable~#1~undefined. }
8915 {
8916   \c_msg_coding_error_text_tl
8917   LaTeX~has~been~asked~to~show~a~variable~#1,~but~this~has~not~
8918   been~defined~yet.
8919 }
8920 \msg_kernel_new:nnnn { seq } { empty-sequence }
8921 { Empty~sequence~#1. }
8922 {
8923   \c_msg_coding_error_text_tl
8924   LaTeX~has~been~asked~to~recover~an~entry~from~a~sequence~that~
8925   has~no~content:~that~cannot~happen!
8926 }
8927 \msg_kernel_new:nnnn { tl } { empty-search-pattern }
8928 { Empty~search~pattern. }
8929 {
8930   \c_msg_coding_error_text_tl
8931   LaTeX~has~been~asked~to~replace~an~empty~pattern~by~'~#1':~that~%
8932   would~lead~to~an~infinite~loop!
8933 }

```

```

\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.

```

8934 \cs_set_protected:Npn \msg_kernel_bug:x #1
8935 {
8936   \msg_interrupt:xxx { \c_msg_kernel_bug_text_tl }
8937   {
8938     #1
8939     \msg_see_documentation_text:n { LaTeX3 }
8940   }
8941   { \c_msg_kernel_bug_more_text_tl }
8942 }
8943 \tl_const:Nn \c_msg_kernel_bug_text_tl
8944 { This~is~a~LaTeX~bug:~check~coding! }

```

```

8945 \tl_const:Nn \c_msg_kernel_bug_more_text_tl
8946 {
8947   There~is~a~coding~bug~somewhere~around~here. \\\
8948   This~probably~needs~examining~by~an~expert.
8949   \c_msg_return_text_tl
8950 }

```

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

194.5 Expandable errors

`\msg_expandable_error:n` In expansion only context, we cannot use the normal means of reporting errors. Instead, we feed TeX an undefined control sequence, `\LaTeX3 error:.` It is thus interrupted, and shows the context, which thanks to the odd-looking `\use:n` is

```

<argument> \LaTeX3 error:
                The error message.

```

In other words, TeX is processing the argument of `\use:n`, which is `\LaTeX3 error: <error message>`. Then `\msg_expandable_error_aux:w` cleans up. In fact, there is an extra subtlety: if the user inserts tokens for error recovery, they should be kept. Thus we also use an odd space character (with category code 7) and keep tokens until that space character, dropping everything else until `\q_stop`. The `\c_zero` prevents losing braces around the user-inserted text if any, and stops the expansion of `\romannumeral`.

```

8951 \group_begin:
8952 \char_set_catcode_math_superscript:N \^
8953 \char_set_lccode:nn {'^} {'\ }
8954 \char_set_lccode:nn {'L} {'L}
8955 \char_set_lccode:nn {'T} {'T}
8956 \char_set_lccode:nn {'X} {'X}
8957 \tl_to_lowercase:n
8958 {
8959   \cs_new:Npx \msg_expandable_error:n #1
8960   {
8961     \exp_not:n
8962     {
8963       \tex_romannumeral:D
8964       \exp_after:wN \exp_after:wN
8965       \exp_after:wN \msg_expandable_error_aux:w
8966       \exp_after:wN \exp_after:wN
8967       \exp_after:wN \c_zero
8968     }
8969     \exp_not:N \use:n { \exp_not:c { LaTeX3-error: } ^ #1 }
8970     \exp_not:N \q_stop
8971   }
8972   \cs_new:Npn \msg_expandable_error_aux:w #1 ^ #2 \q_stop { #1 }
8973 }
8974 \group_end:

```

(End definition for `\msg_expandable_error:n`. This function is documented on page 149.)

194.6 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.

```
8975 <*deprecated>
8976 \cs_new_eq:NN \msg_class_new:nn \msg_class_set:nn
8977 </deprecated>
      (End definition for \msg_class_new:nn. This function is documented on page ??.)
```

`\msg_trace:nnxxxx` The performance here is never going to be good enough for tracing code, so let's be realistic.

```
\msg_trace:nnxxx
\msg_trace:nnxx
\msg_trace:nnx
\msg_trace:nn
8978 <*deprecated>
8979 \cs_new_eq:NN \msg_trace:nnxxxx \msg_log:nnxxxx
8980 \cs_new_eq:NN \msg_trace:nnxxx \msg_log:nnxxx
8981 \cs_new_eq:NN \msg_trace:nnxx \msg_log:nnxx
8982 \cs_new_eq:NN \msg_trace:nnx \msg_log:nnx
8983 \cs_new_eq:NN \msg_trace:nn \msg_log:nn
8984 </deprecated>
      (End definition for \msg_trace:nnxxxx and others. These functions are documented on page ??.)
```

`\msg_generic_new:nnn` These were all too low-level.

```
\msg_generic_new:nn
\msg_generic_set:nnn
\msg_generic_set:nn
\msg_direct_interrupt:xxxxx
\msg_direct_log:xx
\msg_direct_term:xx
8985 <*deprecated>
8986 \cs_new_protected:Npn \msg_generic_new:nnn #1#2#3 { \deprecated }
8987 \cs_new_protected:Npn \msg_generic_new:nn #1#2 { \deprecated }
8988 \cs_new_protected:Npn \msg_generic_set:nnn #1#2#3 { \deprecated }
8989 \cs_new_protected:Npn \msg_generic_set:nn #1#2 { \deprecated }
8990 \cs_new_protected:Npn \msg_direct_interrupt:xxxxx #1#2#3#4#5 { \deprecated }
8991 \cs_new_protected:Npn \msg_direct_log:xx #1#2 { \deprecated }
8992 \cs_new_protected:Npn \msg_direct_term:xx #1#2 { \deprecated }
8993 </deprecated>
      (End definition for \msg_generic_new:nnn. This function is documented on page ??.)
8994 </initex | package>
```

195 l3keys Implementation

```
8995 <*initex | package>
8996 <*package>
8997 \ProvidesExplPackage
8998   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
8999 \package_check_loaded_expl:
9000 </package>
```

195.1 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.

```
9001 \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 9002 \tl_new:N \l_keyval_key_tl
9003 \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 9004 \tl_new:N \l_keyval_sanitise_tl
9005 \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.

```
9006 \group_begin:
9007   \char_set_catcode_active:n { '=' }
9008   \char_set_catcode_active:n { '\', }
9009   \char_set_lccode:nn { '\8' } { '=' }
9010   \char_set_lccode:nn { '\9' } { '\', }
9011   \tl_to_lowercase:n
9012   {
9013     \group_end:
9014     \cs_new_protected:Npn \keyval_parse:n #1
9015     {
9016       \group_begin:
9017       \tl_clear:N \l_keyval_sanitise_tl
9018       \tl_set:Nn \l_keyval_sanitise_tl {#1}
9019       \tl_replace_all:Nnn \l_keyval_sanitise_tl { = } { 8 }
9020       \tl_replace_all:Nnn \l_keyval_sanitise_tl { , } { 9 }
9021       \tl_clear:N \l_keyval_parse_tl
9022       \exp_after:wN \keyval_parse_elt:w \exp_after:wN
9023       \q_no_value \l_keyval_sanitise_tl 9 \q_nil 9
9024       \exp_after:wN \group_end:
9025       \l_keyval_parse_tl
9026     }
9027   }
```

(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.

```
9028 \cs_new_protected:Npn \keyval_parse_elt:w #1 ,
9029 {
9030   \tl_if_blank:OTF { \use_none:n #1 }
```

```

9031     { \keyval_parse_elt:w \q_no_value }
9032     {
9033         \quark_if_nil:oF { \use_ii:nn #1 }
9034         {
9035             \keyval_split_key_value:w #1 = = \q_stop
9036             \keyval_parse_elt:w \q_no_value
9037         }
9038     }
9039 }

```

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

\keyval_split_key_value:w The key and value are handled separately. First the key is grabbed and saved as `\l_keyval_key_tl`. Then a check is need to see if there is a value at all: if not then the key name is simply added to the output. If there is a value then there is a check to ensure that there was only one `=` in the input (remembering some extra ones are around at the moment to prevent errors). All being well, there is an hand-off to find the value: the `\q_nil` is there to prevent loss of braces.

\keyval_split_key_value_aux:wTF

```

9040 \cs_new_protected:Npn \keyval_split_key_value:w #1 = #2 \q_stop
9041 {
9042     \keyval_split_key:w #1 \q_stop
9043     \str_if_eq:nnTF {#2} { = }
9044     {
9045         \tl_put_right:Nx \l_keyval_parse_tl
9046         {
9047             \exp_not:c
9048             { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n }
9049             { \exp_not:o \l_keyval_key_tl }
9050         }
9051     }
9052     {
9053         \keyval_split_key_value_aux:wTF #2 \q_no_value \q_stop
9054         { \keyval_split_value:w \q_nil #2 }
9055         { \msg_kernel_error:nn { keyval } { misplaced-equals-sign } }
9056     }
9057 }
9058 \cs_new:Npn \keyval_split_key_value_aux:wTF #1 = #2#3 \q_stop
9059 { \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`.

```

9060 \cs_new_protected:Npn \keyval_split_key:w #1 \q_stop
9061 {
9062     \tl_set:Nx \l_keyval_key_tl
9063     { \exp_after:wN \tl_trim_spaces:n \exp_after:wN { \use_none:n #1 } }
9064 }

```

(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.

```

9065 \cs_new_protected:Npn \keyval_split_value:w #1 = =
9066 {
9067   \tl_put_right:Nx \l_keyval_parse_tl
9068   {
9069     \exp_not:c
9070     { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn }
9071     { \exp_not:o \l_keyval_key_tl }
9072   }
9073   \tl_set:Nx \l_keyval_value_tl
9074   { \exp_not:o { \use_none:nnn #1 \q_nil \q_nil } }
9075   \tl_if_empty:NTF \l_keyval_value_tl
9076   { \tl_put_right:Nn \l_keyval_parse_tl { { } } }
9077   {
9078     \quark_if_nil:NTF \l_keyval_value_tl
9079     {
9080       \tl_put_right:Nx \l_keyval_parse_tl
9081       { { \exp_not:o { \use_ii:nnn #1 \q_nil } } }
9082     }
9083     { \keyval_split_value_aux:w #1 \q_stop }
9084   }
9085 }

```

A similar idea to the key code: remove the spaces from each end and deal with one set of braces.

```

9086 \cs_new_protected:Npn \keyval_split_value_aux:w \q_nil #1 \q_stop
9087 {
9088   \tl_set:Nx \l_keyval_value_tl { \tl_trim_spaces:n {#1} }
9089   \tl_put_right:Nx \l_keyval_parse_tl
9090   { { \exp_not:o \l_keyval_value_tl } }
9091 }

```

(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.

```

9092 \cs_new_protected:Npn \keyval_parse:NNn #1#2#3
9093 {
9094   \int_gincr:N \g_keyval_level_int
9095   \cs_gset_eq:cN
9096   { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n } #1
9097   \cs_gset_eq:cN
9098   { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn } #2
9099   \keyval_parse:n {#3}
9100   \int_gdecr:N \g_keyval_level_int

```

```

9101 }
      (End definition for \keyval_parse:Nn. This function is documented on page 160.)
      One message for the low level parsing system.
9102 \msg_kernel_new:nnnn { keyval } { misplaced-equals-sign }
9103 { Misplaced-equals-sign-in-key-value-input~\msg_line_number: }
9104 {
9105   LaTeX-is-attempting-to-parse-some-key-value-input-but-found~
9106   two-equals-signs-not-separated-by-a-comma.
9107 }

```

195.2 Constants and variables

\c_keys_code_root_tl	The prefixes for the code and variables of the keys themselves.
\c_keys_vars_root_tl	<pre> 9108 \tl_const:Nn \c_keys_code_root_tl { key~code~>~ } 9109 \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 ??.) </pre>
\c_keys_props_root_tl	<p>The prefix for storing properties.</p> <pre> 9110 \tl_const:Nn \c_keys_props_root_tl { key~prop~>~ } (End definition for \c_keys_props_root_tl. This function is documented on page ??.) </pre>
\c_keys_value_forbidden_tl	Two marker token lists.
\c_keys_value_required_tl	<pre> 9111 \tl_const:Nn \c_keys_value_forbidden_tl { forbidden } 9112 \tl_const:Nn \c_keys_value_required_tl { required } (End definition for \c_keys_value_forbidden_tl and \c_keys_value_required_tl. These functions are documented on page ??.) </pre>
\l_keys_choice_int	Publicly accessible data on which choice is being used when several are generated as a set.
\l_keys_choices_tl	<pre> 9113 \int_new:N \l_keys_choice_int 9114 \tl_new:N \l_keys_choices_tl (End definition for \l_keys_choice_int and \l_keys_choices_tl. These functions are documented on page ??.) </pre>
\l_keys_key_tl	<p>The name of a key itself: needed when setting keys.</p> <pre> 9115 \tl_new:N \l_keys_key_tl (End definition for \l_keys_key_tl. This function is documented on page 158.) </pre>
\l_keys_module_tl	<p>The module for an entire set of keys.</p> <pre> 9116 \tl_new:N \l_keys_module_tl (End definition for \l_keys_module_tl. This function is documented on page ??.) </pre>
\l_keys_no_value_bool	<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> 9117 \bool_new:N \l_keys_no_value_bool (End definition for \l_keys_no_value_bool. This function is documented on page ??.) </pre>

`\l_keys_path_tl` The “path” of the current key is stored here: this is available to the programmer and so is public.

```
9118 \tl_new:N \l_keys_path_tl
      (End definition for \l_keys_path_tl. This function is documented on page 158.)
```

`\l_keys_property_tl` The “property” begin set for a key at definition time is stored here.

```
9119 \tl_new:N \l_keys_property_tl
      (End definition for \l_keys_property_tl. This function is documented on page ??.)
```

`\l_keys_unknown_clist` Used when setting only known keys to store those left over.

```
9120 \tl_new:N \l_keys_unknown_clist
      (End definition for \l_keys_unknown_clist. This function is documented on page ??.)
```

`\l_keys_value_tl` The value given for a key: may be empty if no value was given.

```
9121 \tl_new:N \l_keys_value_tl
      (End definition for \l_keys_value_tl. This function is documented on page 158.)
```

195.3 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).

```
9122 \cs_new_protected:Npn \keys_define:nn
9123 { \keys_define_aux:onnn \l_keys_module_tl }
9124 \cs_new_protected:Npn \keys_define_aux:nnn #1#2#3
9125 {
9126   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9127   \keyval_parse:NNn \keys_define_elt:n \keys_define_elt:nn {#3}
9128   \tl_set:Nn \l_keys_module_tl {#1}
9129 }
9130 \cs_generate_variant:Nn \keys_define_aux:nnn { o }
      (End definition for \keys_define:nn. This function is documented on page ??.)
```

`\keys_define_elt:n` The outer functions here record whether a value was given and then converge on a common internal mechanism. There is first a search for a property in the current key name, then a check to make sure it is known before the code hands off to the next step.

```
\keys_define_elt:nn
\keys_define_elt_aux:nn
9131 \cs_new_protected_nopar:Npn \keys_define_elt:n #1
9132 {
9133   \bool_set_true:N \l_keys_no_value_bool
9134   \keys_define_elt_aux:nn {#1} { }
9135 }
9136 \cs_new_protected:Npn \keys_define_elt:nn #1#2
9137 {
9138   \bool_set_false:N \l_keys_no_value_bool
9139   \keys_define_elt_aux:nn {#1} {#2}
9140 }
9141 \cs_new_protected:Npn \keys_define_elt_aux:nn #1#2 {
```

```

9142 \keys_property_find:n {#1}
9143 \cs_if_exist:cTF { \c_keys_props_root_tl \l_keys_property_tl }
9144 { \keys_define_key:n {#2} }
9145 {
9146     \msg_kernel_error:nxx { keys } { property-unknown }
9147     { \l_keys_property_tl } { \l_keys_path_tl }
9148 }
9149 }

```

(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.

```

9150 \cs_new_protected_nopar:Npn \keys_property_find:n #1
9151 {
9152     \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / }
9153     \tl_if_in:nnTF {#1} { . }
9154     { \keys_property_find_aux:w #1 \q_stop }
9155     { \msg_kernel_error:nxx { keys } { key-no-property } {#1} }
9156 }
9157 \cs_new_protected_nopar:Npn \keys_property_find_aux:w #1 . #2 \q_stop
9158 {
9159     \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl \tl_to_str:n {#1} }
9160     \tl_if_in:nnTF {#2} { . }
9161     {
9162         \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl . }
9163         \keys_property_find_aux:w #2 \q_stop
9164     }
9165     { \tl_set:Nn \l_keys_property_tl { . #2 } }
9166 }

```

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

`\keys_define_key:n` Two possible cases. If there is a value for the key, then just use the function. If not, `\keys_define_key_aux:w` then a check to make sure there is no need for a value with the property. If there should be one then complain, otherwise execute it. There is no need to check for a : as if it is missing the earlier tests will have failed.

```

9167 \cs_new_protected:Npn \keys_define_key:n #1
9168 {
9169     \bool_if:NTF \l_keys_no_value_bool
9170     {
9171         \exp_after:wN \keys_define_key_aux:w
9172         \l_keys_property_tl \q_stop
9173         { \use:c { \c_keys_props_root_tl \l_keys_property_tl } }
9174         {
9175             \msg_kernel_error:nxx { keys }
9176             { property-requires-value } { \l_keys_property_tl }
9177             { \l_keys_path_tl }
9178         }
9179     }

```

```

9180         { \use:c { \c_keys_props_root_tl \l_keys_property_tl } {#1} }
9181     }
9182     \cs_new_protected:Npn \keys_define_key_aux:w #1 : #2 \q_stop
9183     { \tl_if_empty:nTF {#2} }

```

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

195.4 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.

```

9184 \cs_new_nopar:Npn \keys_bool_set:NN #1#2
9185 {
9186     \cs_if_exist:NF #1 { \bool_new:N #1 }
9187     \keys_choice_make:
9188     \keys_cmd_set:nx { \l_keys_path_tl / true }
9189     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9190     \keys_cmd_set:nx { \l_keys_path_tl / false }
9191     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9192     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9193     {
9194         \msg_kernel_error:nnx { keys } { boolean-values-only }
9195         { \l_keys_key_tl }
9196     }
9197     \keys_default_set:n { true }
9198 }

```

(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.

```

9199 \cs_new_nopar:Npn \keys_bool_set_inverse:NN #1#2
9200 {
9201     \cs_if_exist:NF #1 { \bool_new:N #1 }
9202     \keys_choice_make:
9203     \keys_cmd_set:nx { \l_keys_path_tl / true }
9204     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9205     \keys_cmd_set:nx { \l_keys_path_tl / false }
9206     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9207     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9208     {
9209         \msg_kernel_error:nnx { keys } { boolean-values-only }
9210         { \l_keys_key_tl }
9211     }
9212     \keys_default_set:n { true }
9213 }

```

(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.

```

9214 \cs_new_protected_nopar:Npn \keys_choice_make:
9215 {

```

```

9216     \keys_cmd_set:nn { \l_keys_path_tl }
9217     { \keys_choice_find:n {##1} }
9218     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9219     {
9220         \msg_kernel_error:nxxx { keys } { choice-unknown }
9221         { \l_keys_path_tl } {##1}
9222     }
9223 }

```

(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.

```

9224 \cs_new_protected:Npn \keys_choices_make:nn #1#2
9225 {
9226     \keys_choice_make:
9227     \int_zero:N \l_keys_choice_int
9228     \clist_map_inline:nn {#1}
9229     {
9230         \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9231         {
9232             \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9233             \int_set:Nn \exp_not:N \l_keys_choice_int
9234             { \int_use:N \l_keys_choice_int }
9235             \exp_not:n {#2}
9236         }
9237         \int_incr:N \l_keys_choice_int
9238     }
9239 }

```

(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

```

9240 \cs_new_protected:Npn \keys_choices_generate:n #1
9241 {
9242     \cs_if_exist:cTF
9243     { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9244     {
9245         \keys_choice_make:
9246         \int_zero:N \l_keys_choice_int
9247         \clist_map_function:nN {#1} \keys_choices_generate_aux:n
9248     }
9249     {
9250         \msg_kernel_error:nxx { keys }
9251         { generate-choices-before-code } { \l_keys_path_tl }
9252     }
9253 }
9254 \cs_new_protected_nopar:Npn \keys_choices_generate_aux:n #1
9255 {
9256     \keys_cmd_set:nx { \l_keys_path_tl / #1 }

```

```

9257     {
9258         \tl_set:Nn \exp_not:N \l_keys_choice_tl {#1}
9259         \int_set:Nn \exp_not:N \l_keys_choice_int
9260             { \int_use:N \l_keys_choice_int }
9261         \exp_not:v
9262             { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9263     }
9264     \int_incr:N \l_keys_choice_int
9265 }

```

(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.

```

9266 \cs_new_protected:Npn \keys_choice_code_store:x #1
9267 {
9268     \cs_if_exist:cF
9269     { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9270     {
9271         \tl_new:c
9272         { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9273     }
9274     \tl_set:cx { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9275     {#1}
9276 }

```

(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 9277 \cs_new_protected:Npn \keys_cmd_set:nn #1#2
9278 {
9279     \keys_cmd_set_aux:n {#1}
9280     \cs_set:cpn { \c_keys_code_root_tl #1 } ##1 {#2}
9281 }
9282 \cs_new_protected:Npn \keys_cmd_set:nx #1#2
9283 {
9284     \keys_cmd_set_aux:n {#1}
9285     \cs_set:cpx { \c_keys_code_root_tl #1 } ##1 {#2}
9286 }
9287 \cs_new_protected_nopar:Npn \keys_cmd_set_aux:n #1
9288 {
9289     \tl_clear_new:c { \c_keys_vars_root_tl #1 .default }
9290     \tl_set:cn { \c_keys_vars_root_tl #1 .default } { \q_no_value }
9291     \tl_clear_new:c { \c_keys_vars_root_tl #1 .req }
9292 }

```

(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 9293 \cs_new_protected:Npn \keys_default_set:n #1
9294 { \tl_set:cn { \c_keys_vars_root_tl \l_keys_path_tl .default } {#1} }
9295 \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 9296 \cs_new_protected_nopar:Npn \keys_meta_make:n #1
9297 {
9298   \exp_args:NNo \keys_cmd_set:nn \l_keys_path_tl
9299   { \exp_after:wN \keys_set:nn \exp_after:wN { \l_keys_module_tl } {#1} }
9300 }
9301 \cs_new_protected_nopar:Npn \keys_meta_make:x #1
9302 {
9303   \keys_cmd_set:nx { \l_keys_path_tl }
9304   { \exp_not:N \keys_set:nn { \l_keys_module_tl } {#1} }
9305 }

```

(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.

```

9306 \cs_new_nopar:Npn \keys_multichoice_find:n #1
9307 { \clist_map_function:nN {#1} \keys_choice_find:n }
9308 \cs_new_protected_nopar:Npn \keys_multichoice_make:
9309 {
9310   \keys_cmd_set:nn { \l_keys_path_tl }
9311   { \keys_multichoice_find:n {##1} }
9312   \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9313   {
9314     \msg_kernel_error:nnxx { keys } { choice-unknown }
9315     { \l_keys_path_tl } {##1}
9316   }
9317 }
9318 \cs_new_protected:Npn \keys_multichoices_make:nn #1#2
9319 {
9320   \keys_multichoice_make:
9321   \int_zero:N \l_keys_choice_int
9322   \clist_map_inline:nn {#1}
9323   {
9324     \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9325     {
9326       \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9327       \int_set:Nn \exp_not:N \l_keys_choice_int
9328       { \int_use:N \l_keys_choice_int }
9329       \exp_not:n {#2}
9330     }
9331     \int_incr:N \l_keys_choice_int
9332   }
9333 }

```

(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.

```

9334 \cs_new_protected_nopar:Npn \keys_value_requirement:n #1
9335 {
9336   \tl_set_eq:cc
9337   { \c_keys_vars_root_tl \l_keys_path_tl .req }
9338   { c_keys_value_ #1 _tl }
9339 }

```

(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`

```

9340 \cs_new_protected_nopar:Npn \keys_variable_set:NnNN #1#2#3#4
9341 {
9342   \cs_if_exist:NF #1 { \use:c { #2 _new:N } #1 }
9343   \keys_cmd_set:nx { \l_keys_path_tl }
9344   { \exp_not:c { #2 _ #3 set:N #4 } \exp_not:N #1 {##1} }
9345 }
9346 \cs_new_protected_nopar:Npn \keys_variable_set:NnN #1#2#3
9347 { \keys_variable_set:NnNN #1 {#2} { } #3 }
9348 \cs_generate_variant:Nn \keys_variable_set:NnNN { c }
9349 \cs_generate_variant:Nn \keys_variable_set:NnN { c }

```

(End definition for \keys_variable_set:NnNN and \keys_variable_set:cnNN. These functions are documented on page ??.)

195.5 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.

```

9350 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_set:N } #1
9351 { \keys_bool_set:NN #1 { } }
9352 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_gset:N } #1
9353 { \keys_bool_set:NN #1 g }

```

(End definition for .bool_set:N. This function is documented on page 152.)

`.bool_set_inverse:N` One function for this.

```

9354 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_set_inverse:N } #1
9355 { \keys_bool_set_inverse:NN #1 { } }
9356 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .bool_gset_inverse:N } #1
9357 { \keys_bool_set_inverse:NN #1 g }

```

(End definition for .bool_set_inverse:N. This function is documented on page 152.)

`.choice:` Making a choice is handled internally, as it is also needed by `.generate_choices:n`.

```

9358 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .choice: }
9359 { \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.

```

9360 \cs_new_protected:cpn { \c_keys_props_root_tl .choices:nn } #1
9361 { \keys_choices_make:nn #1 }

```

(End definition for .choices:nn. This function is documented on page 152.)

`.code:n` Creating code is simply a case of passing through to the underlying `set` function.

`.code:x`

```

9362 \cs_new_protected:cpn { \c_keys_props_root_tl .code:n } #1
9363 { \keys_cmd_set:nn { \l_keys_path_tl } {#1} }
9364 \cs_new_protected:cpn { \c_keys_props_root_tl .code:x } #1
9365 { \keys_cmd_set:nx { \l_keys_path_tl } {#1} }

```

(End definition for .code:n and .code:x. These functions are documented on page 153.)

`.choice_code:n` Storing the code for choices, using `\exp_not:n` to avoid needing two internal functions.

`.choice_code:x`

```

9366 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:n } #1
9367 { \keys_choice_code_store:x { \exp_not:n {#1} } }
9368 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:x } #1
9369 { \keys_choice_code_store:x {#1} }

```

(End definition for .choice_code:n and .choice_code:x. These functions are documented on page 152.)

`.clist_set:N`

`.clist_set:c`

`.clist_gset:N`

`.clist_gset:c`

```

9370 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_set:N } #1
9371 { \keys_variable_set:NnN #1 { clist } n }
9372 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_set:c } #1
9373 { \keys_variable_set:cnN {#1} { clist } n }
9374 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_gset:N } #1
9375 { \keys_variable_set:NnNN #1 { clist } g n }
9376 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .clist_gset:c } #1
9377 { \keys_variable_set:cnNN {#1} { clist } g n }

```

(End definition for .clist_set:N and .clist_set:c. These functions are documented on page 152.)

`.default:n` Expansion is left to the internal functions.

`.default:V`

```

9378 \cs_new_protected:cpn { \c_keys_props_root_tl .default:n } #1
9379 { \keys_default_set:n {#1} }
9380 \cs_new_protected:cpn { \c_keys_props_root_tl .default:V } #1
9381 { \keys_default_set:V #1 }

```

(End definition for .default:n and .default:V. These functions are documented on page 153.)

`.dim_set:N` Setting a variable is very easy: just pass the data along.

`.dim_set:c`

`.dim_gset:N`

`.dim_gset:c`

```

9382 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_set:N } #1
9383 { \keys_variable_set:NnN #1 { dim } n }
9384 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_set:c } #1
9385 { \keys_variable_set:cnN {#1} { dim } n }
9386 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_gset:N } #1
9387 { \keys_variable_set:NnNN #1 { dim } g n }
9388 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .dim_gset:c } #1
9389 { \keys_variable_set:cnNN {#1} { dim } g n }

```

(End definition for `.dim_set:N` and `.dim_set:c`. These functions are documented on page 153.)

```
.fp_set:N Setting a variable is very easy: just pass the data along.
.fp_set:c 9390 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_set:N } #1
.fp_gset:N 9391 { \keys_variable_set:NnN #1 { fp } n }
.fp_gset:c 9392 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_set:c } #1
          9393 { \keys_variable_set:cnN {#1} { fp } n }
          9394 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_gset:N } #1
          9395 { \keys_variable_set:NnNN #1 { fp } g n }
          9396 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .fp_gset:c } #1
          9397 { \keys_variable_set:cnNN {#1} { fp } g n }
```

(End definition for `.fp_set:N` and `.fp_set:c`. These functions are documented on page 153.)

```
.generate_choices:n Making choices is easy.
          9398 \cs_new_protected:cpn { \c_keys_props_root_tl .generate_choices:n } #1
          9399 { \keys_choices_generate:n {#1} }
```

(End definition for `.generate_choices:n`. This function is documented on page 154.)

```
.int_set:N Setting a variable is very easy: just pass the data along.
.int_set:c 9400 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_set:N } #1
.int_gset:N 9401 { \keys_variable_set:NnN #1 { int } n }
.int_gset:c 9402 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_set:c } #1
          9403 { \keys_variable_set:cnN {#1} { int } n }
          9404 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_gset:N } #1
          9405 { \keys_variable_set:NnNN #1 { int } g n }
          9406 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .int_gset:c } #1
          9407 { \keys_variable_set:cnNN {#1} { int } g n }
```

(End definition for `.int_set:N` and `.int_set:c`. These functions are documented on page 154.)

```
.meta:n Making a meta is handled internally.
.meta:x 9408 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:n } #1
          9409 { \keys_meta_make:n {#1} }
          9410 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:x } #1
          9411 { \keys_meta_make:x {#1} }
```

(End definition for `.meta:n` and `.meta:x`. These functions are documented on page 154.)

```
.multichoice: The same idea as .choice: and .choices:nn, but where more than one choice is allowed.
.multichoices:nn 9412 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .multichoice: }
          9413 { \keys_multichoice_make: }
          9414 \cs_new_protected:cpn { \c_keys_props_root_tl .multichoices:nn } #1
          9415 { \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 9416 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_set:N } #1
.skip_gset:N 9417 { \keys_variable_set:NnN #1 { skip } n }
.skip_gset:c 9418 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_set:c } #1
          9419 { \keys_variable_set:cnN {#1} { skip } n }
```

```

9420 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_gset:N } #1
9421   { \keys_variable_set:NnNN #1 { skip } g n }
9422 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .skip_gset:c } #1
9423   { \keys_variable_set:cnNN {#1} { skip } g n }

```

(End definition for `.skip_set:N` and `.skip_set:c`. These functions are documented on page 154.)

```

.tl_set:N Setting a variable is very easy: just pass the data along.
.tl_set:c 9424 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set:N } #1
.tl_gset:N 9425   { \keys_variable_set:NnNN #1 { tl } n }
.tl_gset:c 9426 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set:c } #1
.tl_set_x:N 9427   { \keys_variable_set:cnN {#1} { tl } n }
.tl_set_x:c 9428 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set_x:N } #1
.tl_gset_x:N 9429   { \keys_variable_set:NnNN #1 { tl } x }
.tl_gset_x:c 9430 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_set_x:c } #1
9431   { \keys_variable_set:cnN {#1} { tl } x }
9432 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset:N } #1
9433   { \keys_variable_set:NnNN #1 { tl } g n }
9434 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset:c } #1
9435   { \keys_variable_set:cnNN {#1} { tl } g n }
9436 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset_x:N } #1
9437   { \keys_variable_set:NnNN #1 { tl } g x }
9438 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .tl_gset_x:c } #1
9439   { \keys_variable_set:cnNN {#1} { tl } g x }

```

(End definition for `.tl_set:N` and `.tl_set:c`. These functions are documented on page 155.)

```

.value_forbidden: These are very similar, so both call the same function.
.value_required: 9440 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_forbidden: }
9441   { \keys_value_requirement:n { forbidden } }
9442 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_required: }
9443   { \keys_value_requirement:n { required } }

```

(End definition for `.value_forbidden:..`. This function is documented on page ??.)

195.6 Setting keys

```

\keys_set:nn A simple wrapper again.
\keys_set:nV 9444 \cs_new_protected:Npn \keys_set:nn
\keys_set:nv 9445   { \keys_set_aux:onnn { \l_keys_module_tl } }
\keys_set:no 9446 \cs_new_protected:Npn \keys_set_aux:nnn #1#2#3
\keys_set_aux:nnn 9447   {
9448     \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9449     \keyval_parse:Nnn \keys_set_elt:n \keys_set_elt:nn {#3}
9450     \tl_set:Nn \l_keys_module_tl {#1}
9451   }
9452 \cs_generate_variant:Nn \keys_set:nn { nV , nv , no }
9453 \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
\keys_set_known:nvN
\keys_set_known:noN
\keys_set_known_aux:nnnN
\keys_set_known_aux:onnN
9454 \cs_new_protected:Npn \keys_set_known:nnN
9455 { \keys_set_known_aux:onnN { \l_keys_module_tl } }
9456 \cs_new_protected:Npn \keys_set_known_aux:nnnN #1#2#3#4
9457 {
9458   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9459   \clist_clear:N \l_keys_unknown_clist
9460   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_alt:
9461   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9462   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_std:
9463   \tl_set:Nn \l_keys_module_tl {#1}
9464   \clist_set_eq:NN #4 \l_keys_unknown_clist
9465 }
9466 \cs_generate_variant:Nn \keys_set_known:nnN { nV , nv , no }
9467 \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.

```

9468 \cs_new_protected_nopar:Npn \keys_set_elt:n #1
9469 {
9470   \bool_set_true:N \l_keys_no_value_bool
9471   \keys_set_elt_aux:nn {#1} { }
9472 }
9473 \cs_new_protected:Npn \keys_set_elt:nn #1#2
9474 {
9475   \bool_set_false:N \l_keys_no_value_bool
9476   \keys_set_elt_aux:nn {#1} {#2}
9477 }
9478 \cs_new_protected:Npn \keys_set_elt_aux:nn #1#2
9479 {
9480   \tl_set:Nx \l_keys_key_tl { \tl_to_str:n {#1} }
9481   \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / \l_keys_key_tl }
9482   \keys_value_or_default:n {#2}
9483   \bool_if:nTF
9484   {
9485     \keys_if_value_p:n { required } &&
9486     \l_keys_no_value_bool
9487   }
9488   {
9489     \msg_kernel_error:nnx { keys } { value-required }
9490     { \l_keys_path_tl }
9491   }
9492   {
9493     \bool_if:nTF
9494     {
9495       \keys_if_value_p:n { forbidden } &&

```

```

9496         ! \l_keys_no_value_bool
9497     }
9498     {
9499         \msg_kernel_error:nxxx { keys } { value-forbidden }
9500         { \l_keys_path_tl } { \l_keys_value_tl }
9501     }
9502     { \keys_execute: }
9503 }
9504 }

```

(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.

```

9505 \cs_new_protected:Npn \keys_value_or_default:n #1
9506 {
9507     \tl_set:Nn \l_keys_value_tl {#1}
9508     \bool_if:NT \l_keys_no_value_bool
9509     {
9510         \quark_if_no_value:cF { \c_keys_vars_root_tl \l_keys_path_tl .default }
9511         {
9512             \cs_if_exist:cT { \c_keys_vars_root_tl \l_keys_path_tl .default }
9513             {
9514                 \tl_set_eq:Nc \l_keys_value_tl
9515                 { \c_keys_vars_root_tl \l_keys_path_tl .default }
9516             }
9517         }
9518     }
9519 }

```

(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.

```

9520 \prg_new_conditional:Npnn \keys_if_value:n #1 { p }
9521 {
9522     \tl_if_eq:ccTF { c_keys_value_ #1 _tl }
9523     { \c_keys_vars_root_tl \l_keys_path_tl .req }
9524     { \prg_return_true: }
9525     { \prg_return_false: }
9526 }

```

(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
9527 \cs_new_nopar:Npn \keys_execute:
9528 { \keys_execute:nn { \l_keys_path_tl } { \keys_execute_unknown: } }
9529 \cs_new_nopar:Npn \keys_execute_unknown:
9530 {
9531     \keys_execute:nn { \l_keys_module_tl / unknown }
9532 }

```

```

9533         \msg_kernel_error:nnxx { keys } { key-unknown }
9534         { \l_keys_path_tl } { \l_keys_module_tl }
9535     }
9536 }
9537 \cs_new_eq:NN \keys_execute_unknown_std: \keys_execute_unknown:
9538 \cs_new_nopar:Npn \keys_execute_unknown_alt:
9539 {
9540     \clist_put_right:Nx \l_keys_unknown_clist
9541     {
9542         \exp_not:o \l_keys_key_tl
9543         \bool_if:NF \l_keys_no_value_bool
9544         { = { \exp_not:o \l_keys_value_tl } }
9545     }
9546 }
9547 \cs_new_nopar:Npn \keys_execute:nn #1#2
9548 {
9549     \cs_if_exist:cTF { \c_keys_code_root_tl #1 }
9550     {
9551         \exp_args:Nno \use:c { \c_keys_code_root_tl #1 }
9552         \l_keys_value_tl
9553     }
9554     {#2}
9555 }

```

(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.

```

9556 \cs_new_nopar:Npn \keys_choice_find:n #1
9557 {
9558     \keys_execute:nn { \l_keys_path_tl / \tl_to_str:n {#1} }
9559     { \keys_execute:nn { \l_keys_path_tl / unknown } { } }
9560 }

```

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

195.7 Utilities

\keys_if_exist:nn A utility for others to see if a key exists.

```

9561 \prg_new_conditional:Npnn \keys_if_exist:nn #1#2 { p , T , F , TF }
9562 {
9563     \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 }
9564     { \prg_return_true: }
9565     { \prg_return_false: }
9566 }

```

(End definition for \keys_if_exist:nn. This function is documented on page 159.)

\keys_if_choice_exist:nnn Just an alternative view on \keys_if_exist:nn(TF).

```

9567 \prg_new_conditional:Npnn \keys_if_choice_exist:nnn #1#2#3 { p , T , F , TF }

```

```

9568 {
9569   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 / #3 }
9570   { \prg_return_true: }
9571   { \prg_return_false: }
9572 }

```

(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.

```

9573 \cs_new_nopar:Npn \keys_show:nn #1#2
9574 { \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 159.)

195.8 Messages

For when there is a need to complain.

```

9575 \msg_kernel_new:nnnn { keys } { boolean-values-only }
9576 { Key~'#1'~accepts~boolean-values-only. }
9577 { The~key~'#1'~only~accepts~the~values~'true'~and~'false'. }
9578 \msg_kernel_new:nnnn { keys } { choice-unknown }
9579 { Choice~'#2'~unknown~for~key~'#1'. }
9580 {
9581   The~key~'#1'~takes~a~limited~number~of~values.\\
9582   The~input~given,~'#2',~is~not~on~the~list~accepted.
9583 }
9584 \msg_kernel_new:nnnn { keys } { generate-choices-before-code }
9585 { No~code~available~to~generate~choices~for~key~'#1'. }
9586 {
9587   \c_msg_coding_error_text_tl
9588   Before~using~.generate_choices:n~the~code~should~be~defined~
9589   with~'.choice_code:n'~or~'.choice_code:x'.
9590 }
9591 \msg_kernel_new:nnnn { keys } { key-no-property }
9592 { No~property~given~in~definition~of~key~'#1'. }
9593 {
9594   \c_msg_coding_error_text_tl
9595   Inside~\keys_define:nn~each~key~name
9596   needs~a~property: \\
9597   ~ ~ #1 .<property> \\
9598   LaTeX~did~not~find~a~'. ' ~to~indicate~the~start~of~a~property.
9599 }
9600 \msg_kernel_new:nnnn { keys } { key-unknown }
9601 { The~key~'#1'~is~unknown~and~is~being~ignored. }
9602 {
9603   The~module~'#2'~does~not~have~a~key~called~'#1'.\\
9604   Check~that~you~have~spelled~the~key~name~correctly.
9605 }
9606 \msg_kernel_new:nnnn { keys } { option-unknown }
9607 { Unknown~option~'#1'~for~package~#2. }
9608 {

```



```

9609 LaTeX-has-been-asked-to-set-an-option-called-~'~#1'~
9610 but~the~#2~package~has~not~created~an~option~with~this~name.
9611 }
9612 \msg_kernel_new:nnnn { keys } { property-requires-value }
9613 { The~property~'~#1'~requires~a~value. }
9614 {
9615   \c_msg_coding_error_text_tl
9616   LaTeX~was~asked~to~set~property~'~#2'~for~key~'~#1'~.\\
9617   No~value~was~given~for~the~property,~and~one~is~required.
9618 }
9619 \msg_kernel_new:nnnn { keys } { property-unknown }
9620 { The~key~property~'~#1'~is~unknown. }
9621 {
9622   \c_msg_coding_error_text_tl
9623   LaTeX~has~been~asked~to~set~the~property~'~#1'~for~key~'~#2':~
9624   this~property~is~not~defined.
9625 }
9626 \msg_kernel_new:nnnn { keys } { value-forbidden }
9627 { The~key~'~#1'~does~not~taken~a~value. }
9628 {
9629   The~key~'~#1'~should~be~given~without~a~value.\\
9630   LaTeX~will~ignore~the~given~value~'~#2'.
9631 }
9632 \msg_kernel_new:nnnn { keys } { value-required }
9633 { The~key~'~#1'~requires~a~value. }
9634 {
9635   The~key~'~#1'~must~have~a~value.\\
9636   No~value~was~present:~the~key~will~be~ignored.
9637 }

```

195.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

There is just one function for this now.

```

\KV_process_space_removal_sanitize:NNn
\KV_process_space_removal_no_sanitize:NNn
\KV_process_no_space_removal_no_sanitize:NNn
9638 < *deprecated >
9639 \cs_new_eq:NN \KV_process_space_removal_sanitize:NNn \keyval_parse:NNn
9640 \cs_new_eq:NN \KV_process_space_removal_no_sanitize:NNn \keyval_parse:NNn
9641 \cs_new_eq:NN \KV_process_no_space_removal_no_sanitize:NNn \keyval_parse:NNn
9642 < /deprecated >
(End definition for \KV_process_space_removal_sanitize:NNn. This function is documented on
page ??.)
9643 < /initex | package >

```

196 l3file implementation

The following test files are used for this code: *m3file001*.

```

9644 < *initex | package >

```

```

9645 <*package>
9646 \ProvidesExplPackage
9647   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9648   \package_check_loaded_expl:
9649 </package>

```

`\g_file_current_name_tl` The name of the current file should be available at all times.

```

9650 \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.

```

9651 <*initex>
9652 \tex_everyjob:D \exp_after:wN
9653   {
9654     \tex_the:D \tex_everyjob:D
9655     \tl_gset:Nx \g_file_current_name_tl { \tex_jobname:D }
9656   }
9657 </initex>
9658 <*package>
9659 \tl_gset_eq:NN \g_file_current_name_tl \@currname
9660 </package>

```

(End definition for `\g_file_current_name_tl`. This function is documented on page 161.)

`\g_file_stack_seq` The input list of files is stored as a sequence stack.

```

9661 \seq_new:N \g_file_stack_seq

```

(End definition for `\g_file_stack_seq`. This function is documented on page 162.)

`\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.

```

9662 \seq_new:N \g_file_record_seq

```

The current file name should be included in the file list!

```

9663 <*initex>
9664 \tex_everyjob:D \exp_after:wN
9665   {
9666     \tex_the:D \tex_everyjob:D
9667     \seq_gput_right:NV \g_file_record_seq \g_file_current_name_tl
9668   }
9669 </initex>

```

(End definition for `\g_file_record_seq`. This function is documented on page 162.)

`\l_file_name_tl` Used to return the fully-qualified name of a file.

```

9670 \tl_new:N \l_file_name_tl

```

(End definition for `\l_file_name_tl`. This function is documented on page 162.)

`\l_file_search_path_seq` The current search path.

```

9671 \seq_new:N \l_file_search_path_seq

```

(End definition for `\l_file_search_path_seq`. This function is documented on page 162.)

`\l_file_search_path_saved_seq` The current search path has to be saved for package use.

```

9672 <*package>
9673 \seq_new:N \l_file_search_path_saved_seq
9674 </package>

```

(End definition for `\l_file_search_path_saved_seq`. This function is documented on page 162.)

`\l_file_tmpa_seq` Scratch space for comma list conversion in package mode.

```

9675 <*package>
9676 \seq_new:N \l_file_tmpa_seq
9677 </package>

```

(End definition for `\l_file_tmpa_seq`. This function is documented on page 162.)

`\file_add_path:nN` The way to test if a file exists is to try to open it: if it does not exist then TeX will report end-of-file. For files which are in the current directory, this is straight-forward.

`\g_file_test_stream` 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.

`\file_add_path_search:nN`

```

9678 \cs_new_protected_nopar:Npn \file_add_path:nN #1#2
9679 {
9680   \ior_open:Nn \g_file_test_stream {#1}
9681   \ior_if_eof:NTF \g_file_test_stream
9682   { \file_add_path_search:nN {#1} #2 }
9683   {
9684     \ior_close:N \g_file_test_stream
9685     \tl_set:Nx #2 {#1}
9686   }
9687 }
9688 \cs_new_protected_nopar:Npn \file_add_path_search:nN #1#2
9689 {
9690   \tl_clear:N #2
9691 <*package>
9692   \cs_if_exist:NT \input@path
9693   {
9694     \seq_set_eq:NN \l_file_search_path_saved_seq \l_file_search_path_seq
9695     \seq_set_from_clist:NN \l_file_tmpa_seq \input@path
9696     \seq_concat:NNN \l_file_search_path_seq
9697       \l_file_search_path_seq \l_file_tmpa_seq
9698   }
9699 </package>
9700   \seq_map_inline:Nn \l_file_search_path_seq
9701   {
9702     \ior_open:Nn \g_file_test_stream { ##1 #1 }
9703     \ior_if_eof:NF \g_file_test_stream
9704     {
9705       \tl_set:Nx #2 { ##1 #1 }
9706       \seq_map_break:
9707     }
9708   }
9709 <*package>

```

```

9710 \cs_if_exist:NT \input@path
9711 { \seq_set_eq:NN \l_file_search_path_seq \l_file_search_path_saved_seq }
9712 </package>
9713 \ior_close:N \g_file_test_stream
9714 }

```

(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.

```

9715 \prg_new_protected_conditional:Nnn \file_if_exist:n { T , F , TF }
9716 {
9717   \file_add_path:nN {#1} \l_file_name_tl
9718   \tl_if_empty:NTF \l_file_name_tl
9719   { \prg_return_false: }
9720   { \prg_return_true: }
9721 }

```

(End definition for \file_if_exist:n. This function is documented on page 161.)

\file_input:n Loading a file is done in a safe way, checking first that the file exists and loading only if it does.

```

9722 \cs_new_protected_nopar:Npn \file_input:n #1
9723 {
9724   \file_add_path:nN {#1} \l_file_name_tl
9725   \tl_if_empty:NF \l_file_name_tl
9726   {
9727     <*initex>
9728     \seq_gput_right:Nx \g_file_record_seq {#1}
9729     </initex>
9730     <*package>
9731     \@addtofilelist {#1}
9732     </package>
9733     \seq_gpush:NV \g_file_stack_seq \g_file_current_name_tl
9734     \tl_gset:Nn \g_file_current_name_tl {#1}
9735     \exp_after:wN \tex_input:D \l_file_name_tl \c_space_tl
9736     \seq_gpop:NN \g_file_stack_seq \g_file_current_name_tl
9737   }
9738 }

```

(End definition for \file_input:n. This function is documented on page 162.)

\file_path_include:n Wrapper functions to manage the search path.

\file_path_remove:n

```

9739 \cs_new_protected_nopar:Npn \file_path_include:n #1
9740 {
9741   \seq_if_in:NnF \l_file_search_path_seq {#1}
9742   { \seq_put_right:Nn \l_file_search_path_seq {#1} }
9743 }
9744 \cs_new_protected_nopar:Npn \file_path_remove:n #1
9745 { \seq_remove_all:Nn \l_file_search_path_seq {#1} }

```

(End definition for \file_path_include:n. This function is documented on page 162.)

`\file_list:` A function to list all files used to the log.

```

9746 \cs_new_protected_nopar:Npn \file_list:
9747 {
9748   \seq_remove_duplicates:N \g_file_record_seq
9749   \iow_log:n { *~File~List~* }
9750   \seq_map_inline:Nn \g_file_record_seq { \iow_log:n {##1} }
9751   \iow_log:n { ***** }
9752 }

```

(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.

```

9753 <*package>
9754 \AtBeginDocument
9755 {
9756   \seq_set_from_clist:NN \l_file_tmpa_seq \@filelist
9757   \seq_gconcat:NNN \g_file_record_seq \g_file_record_seq \l_file_tmpa_seq
9758 }
9759 </package>
9760 </initex | package>

```

197 l3fp Implementation

The following test files are used for this code: *m3fp003.lvt*.

```

9761 <*initex | package>
9762 <*package>
9763 \ProvidesExplPackage
9764   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9765 \package_check_loaded_expl:
9766 </package>

```

197.1 Constants

`\c_forty_four` There is some speed to gain by moving numbers into fixed positions.

```

\c_one_million      9767 \int_const:Nn \c_forty_four { 44 }
\c_one_hundred_million 9768 \int_const:Nn \c_one_million { 1 000 000 }
\c_five_hundred_million 9769 \int_const:Nn \c_one_hundred_million { 100 000 000 }
\c_one_thousand_million 9770 \int_const:Nn \c_five_hundred_million { 500 000 000 }
9771 \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 9772 \int_new:N \c_fp_pi_by_four_decimal_int
\c_fp_pi_decimal_int          9773 \int_set:Nn \c_fp_pi_by_four_decimal_int { 785 398 158 }
\c_fp_pi_extended_int         9774 \int_new:N \c_fp_pi_by_four_extended_int
\c_fp_two_pi_decimal_int      9775 \int_set:Nn \c_fp_pi_by_four_extended_int { 897 448 310 }
\c_fp_two_pi_extended_int     9776 \int_new:N \c_fp_pi_decimal_int

```

```

9777 \int_set:Nn \c_fp_pi_decimal_int { 141 592 653 }
9778 \int_new:N \c_fp_pi_extended_int
9779 \int_set:Nn \c_fp_pi_extended_int { 589 793 238 }
9780 \int_new:N \c_fp_two_pi_decimal_int
9781 \int_set:Nn \c_fp_two_pi_decimal_int { 283 185 307 }
9782 \int_new:N \c_fp_two_pi_extended_int
9783 \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”.

```

9784 \tl_const:Nn \c_e_fp { + 2.718281828 e 0 }
(End definition for \c_e_fp. This function is documented on page 170.)

```

`\c_one_fp` The constant value 1: used for fast comparisons.

```

9785 \tl_const:Nn \c_one_fp { + 1.000000000 e 0 }
(End definition for \c_one_fp. This function is documented on page 170.)

```

`\c_pi_fp` The value π as a “machine number”.

```

9786 \tl_const:Nn \c_pi_fp { + 3.141592654 e 0 }
(End definition for \c_pi_fp. This function is documented on page 170.)

```

`\c_undefined_fp` A marker for undefined values.

```

9787 \tl_const:Nn \c_undefined_fp { X 0.000000000 e 0 }
(End definition for \c_undefined_fp. This function is documented on page 170.)

```

`\c_zero_fp` The constant zero value.

```

9788 \tl_const:Nn \c_zero_fp { + 0.000000000 e 0 }
(End definition for \c_zero_fp. This function is documented on page 170.)

```

197.2 Variables

`\l_fp_arg_tl` A token list to store the formalised representation of the input for transcendental functions.

```

9789 \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.

```

9790 \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.

```

9791 \int_new:N \l_fp_div_offset_int
(End definition for \l_fp_div_offset_int. This function is documented on page ??.)

```

<pre> \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int \l_fp_exp_exponent_int </pre>	<p>Used for the calculation of exponent values.</p> <pre> 9792 \int_new:N \l_fp_exp_integer_int 9793 \int_new:N \l_fp_exp_decimal_int 9794 \int_new:N \l_fp_exp_extended_int 9795 \int_new:N \l_fp_exp_exponent_int </pre> <p>(End definition for \l_fp_exp_integer_int. This function is documented on page ??.)</p>
<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> 9796 \int_new:N \l_fp_input_a_sign_int 9797 \int_new:N \l_fp_input_a_integer_int 9798 \int_new:N \l_fp_input_a_decimal_int 9799 \int_new:N \l_fp_input_a_exponent_int 9800 \int_new:N \l_fp_input_b_sign_int 9801 \int_new:N \l_fp_input_b_integer_int 9802 \int_new:N \l_fp_input_b_decimal_int 9803 \int_new:N \l_fp_input_b_exponent_int </pre> <p>(End definition for \l_fp_input_a_sign_int. This function is documented on page ??.)</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> 9804 \int_new:N \l_fp_input_a_extended_int 9805 \int_new:N \l_fp_input_b_extended_int </pre> <p>(End definition for \l_fp_input_a_extended_int. This function is documented on page ??.)</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> 9806 \int_new:N \l_fp_mul_a_i_int 9807 \int_new:N \l_fp_mul_a_ii_int 9808 \int_new:N \l_fp_mul_a_iii_int 9809 \int_new:N \l_fp_mul_a_iv_int 9810 \int_new:N \l_fp_mul_a_v_int 9811 \int_new:N \l_fp_mul_a_vi_int 9812 \int_new:N \l_fp_mul_b_i_int 9813 \int_new:N \l_fp_mul_b_ii_int 9814 \int_new:N \l_fp_mul_b_iii_int 9815 \int_new:N \l_fp_mul_b_iv_int 9816 \int_new:N \l_fp_mul_b_v_int 9817 \int_new:N \l_fp_mul_b_vi_int </pre> <p>(End definition for \l_fp_mul_a_i_int. This function is documented on page ??.)</p>
<pre> \l_fp_mul_output_int \l_fp_mul_output_tl </pre>	<p>Space for multiplication results.</p> <pre> 9818 \int_new:N \l_fp_mul_output_int 9819 \tl_new:N \l_fp_mul_output_tl </pre> <p>(End definition for \l_fp_mul_output_int. This function is documented on page ??.)</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> 9820 \int_new:N \l_fp_output_sign_int 9821 \int_new:N \l_fp_output_integer_int 9822 \int_new:N \l_fp_output_decimal_int 9823 \int_new:N \l_fp_output_exponent_int </pre>

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

- `\l_fp_output_extended_int` Again, for calculations an extended part.
`9824 \int_new:N \l_fp_output_extended_int`
(End definition for \l_fp_output_extended_int. This function is documented on page ??.)
- `\l_fp_round_carry_bool` To indicate that a digit needs to be carried forward.
`9825 \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.
`9826 \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` `9827 \int_new:N \l_fp_round_position_int`
`9828 \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.
`9829 \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.
`9830 \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.
`9831 \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.
`9832 \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.
`9833 \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` `9834 \int_new:N \l_fp_trig_sign_int`
`\l_fp_trig_extended_int` `9835 \int_new:N \l_fp_trig_decimal_int`
`9836 \int_new:N \l_fp_trig_extended_int`
(End definition for \l_fp_trig_sign_int. This function is documented on page ??.)

197.3 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).

```

9837 \cs_new_protected_nopar:Npn \fp_read:N #1
9838 { \exp_after:wN \fp_read_aux:w #1 \q_stop }
9839 \cs_new_protected_nopar:Npn \fp_read_aux:w #1#2 . #3 e #4 \q_stop
9840 {
9841   \if:w #1 -
9842     \l_fp_input_a_sign_int \c_minus_one
9843   \else:
9844     \l_fp_input_a_sign_int \c_one
9845   \fi:
9846   \l_fp_input_a_integer_int #2 \scan_stop:
9847   \l_fp_input_a_decimal_int #3 \scan_stop:
9848   \l_fp_input_a_exponent_int #4 \scan_stop:
9849 }

```

(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
9850 \cs_new_protected_nopar:Npn \fp_split:Nn #1#2
9851 {
9852   \tl_set:Nx \l_fp_tmp_tl {#2}
9853   \tl_set_rescan:Nno \l_fp_tmp_tl { \char_set_catcode_ignore:n { 32 } }
9854   { \l_fp_tmp_tl }
9855   \l_fp_split_sign_int \c_one
9856   \fp_split_sign:
9857   \use:c { l_fp_input_ #1 _sign_int } \l_fp_split_sign_int
9858   \exp_after:wN \fp_split_exponent:w \l_fp_tmp_tl e e \q_stop #1
9859 }
9860 \cs_new_protected_nopar:Npn \fp_split_sign:
9861 {
9862   \if_int_compare:w \pdfTeX_strcmp:D
9863   { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { - }
9864   = \c_zero
9865   \tl_set:Nx \l_fp_tmp_tl
9866   {
9867     \exp_after:wN
9868     \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
9869   }
9870   \l_fp_split_sign_int -\l_fp_split_sign_int
9871   \exp_after:wN \fp_split_sign:
9872   \else:

```

```

9873 \if_int_compare:w \pdfTeX_strcmp:D
9874 { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { + }
9875 = \c_zero
9876 \tl_set:Nx \l_fp_tmp_tl
9877 {
9878 \exp_after:wN
9879 \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
9880 }
9881 \exp_after:wN \exp_after:wN \exp_after:wN \fp_split_sign:
9882 \fi:
9883 \fi:
9884 }
9885 \cs_new_protected_nopar:Npn \fp_split_exponent:w #1 e #2 e #3 \q_stop #4
9886 {
9887 \use:c { l_fp_input_ #4 _exponent_int }
9888 \int_eval:w 0 #2 \scan_stop:
9889 \tex_afterassignment:D \fp_split_aux_i:w
9890 \use:c { l_fp_input_ #4 _integer_int }
9891 \int_eval:w 0 #1 . . \q_stop #4
9892 }
9893 \cs_new_protected_nopar:Npn \fp_split_aux_i:w #1 . #2 . #3 \q_stop
9894 { \fp_split_aux_ii:w #2 000000000 \q_stop }
9895 \cs_new_protected_nopar:Npn \fp_split_aux_ii:w #1#2#3#4#5#6#7#8#9
9896 { \fp_split_aux_iii:w {#1#2#3#4#5#6#7#8#9} }
9897 \cs_new_protected_nopar:Npn \fp_split_aux_iii:w #1#2 \q_stop
9898 {
9899 \l_fp_tmp_int 1 #1 \scan_stop:
9900 \exp_after:wN \fp_split_decimal:w
9901 \int_use:N \l_fp_tmp_int 000000000 \q_stop
9902 }
9903 \cs_new_protected_nopar:Npn \fp_split_decimal:w #1#2#3#4#5#6#7#8#9
9904 { \fp_split_decimal_aux:w {#2#3#4#5#6#7#8#9} }
9905 \cs_new_protected_nopar:Npn \fp_split_decimal_aux:w #1#2#3 \q_stop #4
9906 {
9907 \use:c { l_fp_input_ #4 _decimal_int } #1#2 \scan_stop:
9908 \if_int_compare:w
9909 \int_eval:w
9910 \use:c { l_fp_input_ #4 _integer_int } +
9911 \use:c { l_fp_input_ #4 _decimal_int }
9912 \scan_stop:
9913 = \c_zero
9914 \use:c { l_fp_input_ #4 _sign_int } \c_one
9915 \fi:
9916 \if_int_compare:w
9917 \use:c { l_fp_input_ #4 _integer_int } < \c_one_thousand_million
9918 \else:
9919 \exp_after:wN \fp_overflow_msg:
9920 \fi:
9921 }

```

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

`\fp_standardise:NNNN` The idea here is to shift the input into a known exponent range. This is done using \TeX tokens where possible, as this is faster than arithmetic.

```

\fp_standardise_aux:NNNN
\fp_standardise_aux:
\fp_standardise_aux:w
9922 \cs_new_protected_nopar:Npn \fp_standardise:NNNN #1#2#3#4
9923 {
9924   \if_int_compare:w
9925     \int_eval:w #2 + #3 = \c_zero
9926     #1 \c_one
9927     #4 \c_zero
9928     \exp_after:wN \use_none:nnnn
9929   \else:
9930     \exp_after:wN \fp_standardise_aux:NNNN
9931   \fi:
9932   #1#2#3#4
9933 }
9934 \cs_new_protected_nopar:Npn \fp_standardise_aux:NNNN #1#2#3#4
9935 {
9936   \cs_set_protected_nopar:Npn \fp_standardise_aux:
9937   {
9938     \if_int_compare:w #2 = \c_zero
9939     \tex_advance:D #3 \c_one_thousand_million
9940     \exp_after:wN \fp_standardise_aux:w
9941     \int_use:N #3 \q_stop
9942     \exp_after:wN \fp_standardise_aux:
9943     \fi:
9944   }
9945   \cs_set_protected_nopar:Npn
9946     \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9 \q_stop
9947   {
9948     #2 ##2 \scan_stop:
9949     #3 ##3##4##5##6##7##8##9 0 \scan_stop:
9950     \tex_advance:D #4 \c_minus_one
9951   }
9952   \fp_standardise_aux:
9953   \cs_set_protected_nopar:Npn \fp_standardise_aux:
9954   {
9955     \if_int_compare:w #2 > \c_nine
9956     \tex_advance:D #2 \c_one_thousand_million
9957     \exp_after:wN \use_i:nn \exp_after:wN
9958     \fp_standardise_aux:w \int_use:N #2
9959     \exp_after:wN \fp_standardise_aux:
9960     \fi:
9961   }
9962   \cs_set_protected_nopar:Npn
9963     \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9
9964   {
9965     #2 ##1##2##3##4##5##6##7##8 \scan_stop:
9966     \tex_advance:D #3 \c_one_thousand_million
9967     \tex_divide:D #3 \c_ten
9968     \tl_set:Nx \l_fp_tmp_tl

```

```

9969         {
9970             ##9
9971             \exp_after:wN \use_none:n \int_use:N #3
9972         }
9973         #3 \l_fp_tmp_tl \scan_stop:
9974         \tex_advance:D #4 \c_one
9975     }
9976     \fp_standardise_aux:
9977     \if_int_compare:w #4 < \c_one_hundred
9978     \if_int_compare:w #4 > -\c_one_hundred
9979     \else:
9980         #1 \c_one
9981         #2 \c_zero
9982         #3 \c_zero
9983         #4 \c_zero
9984     \fi:
9985     \else:
9986     \exp_after:wN \fp_overflow_msg:
9987     \fi:
9988 }
9989 \cs_new_protected_nopar:Npn \fp_standardise_aux: { }
9990 \cs_new_protected_nopar:Npn \fp_standardise_aux:w { }

```

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

197.4 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
\fp_level_input_exponents_a:NNNNNNNNN
\fp_level_input_exponents_b:NNNNNNNNN

```

```

9991 \cs_new_protected_nopar:Npn \fp_level_input_exponents:
9992 {
9993     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
9994     \exp_after:wN \fp_level_input_exponents_a:
9995     \else:
9996     \exp_after:wN \fp_level_input_exponents_b:
9997     \fi:
9998 }
9999 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:
10000 {
10001     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10002     \tex_advance:D \l_fp_input_b_integer_int \c_one_thousand_million
10003     \exp_after:wN \use_i:nn \exp_after:wN
10004     \fp_level_input_exponents_a:NNNNNNNNN
10005     \int_use:N \l_fp_input_b_integer_int
10006     \exp_after:wN \fp_level_input_exponents_a:
10007     \fi:
10008 }
10009 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:NNNNNNNNN
10010 #1#2#3#4#5#6#7#8#9
10011 {

```

```

10012 \l_fp_input_b_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10013 \tex_advance:D \l_fp_input_b_decimal_int \c_one_thousand_million
10014 \tex_divide:D \l_fp_input_b_decimal_int \c_ten
10015 \tl_set:Nx \l_fp_tmp_tl
10016 {
10017     #9
10018     \exp_after:wN \use_none:n
10019     \int_use:N \l_fp_input_b_decimal_int
10020 }
10021 \l_fp_input_b_decimal_int \l_fp_tmp_tl \scan_stop:
10022 \tex_advance:D \l_fp_input_b_exponent_int \c_one
10023 }
10024 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:
10025 {
10026     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
10027     \tex_advance:D \l_fp_input_a_integer_int \c_one_thousand_million
10028     \exp_after:wN \use_i:nn \exp_after:wN
10029     \fp_level_input_exponents_b:NNNNNNNNN
10030     \int_use:N \l_fp_input_a_integer_int
10031     \exp_after:wN \fp_level_input_exponents_b:
10032     \fi:
10033 }
10034 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:NNNNNNNNN
10035 #1#2#3#4#5#6#7#8#9
10036 {
10037     \l_fp_input_a_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10038     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10039     \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10040     \tl_set:Nx \l_fp_tmp_tl
10041     {
10042         #9
10043         \exp_after:wN \use_none:n
10044         \int_use:N \l_fp_input_a_decimal_int
10045     }
10046     \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
10047     \tex_advance:D \l_fp_input_a_exponent_int \c_one
10048 }

```

(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.

```

10049 \cs_new_protected_nopar:Npn \fp_tmp:w #1#2 { }

```

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

197.5 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 10050 \cs_new_protected_nopar:Npn \fp_new:N #1
10051 {
10052   \tl_new:N #1
10053   \tl_gset_eq:NN #1 \c_zero_fp
10054 }
10055 \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 10056 \cs_new_protected_nopar:Npn \fp_const:Nn #1#2
10057 {
10058   \fp_new:N #1
10059   \fp_gset:Nn #1 {#2}
10060 }
10061 \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 10062 \cs_new_protected_nopar:Npn \fp_zero:N #1
\fp_gzero:N 10063 { \tl_set_eq:NN #1 \c_zero_fp }
\fp_gzero:c 10064 \cs_new_protected_nopar:Npn \fp_gzero:N #1
10065 { \tl_gset_eq:NN #1 \c_zero_fp }
10066 \cs_generate_variant:Nn \fp_zero:N { c }
10067 \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:Nn 10068 \cs_new_protected_nopar:Npn \fp_set:Nn { \fp_set_aux:NNn \tl_set:Nn }
\fp_set_aux:NNn 10069 \cs_new_protected_nopar:Npn \fp_gset:Nn { \fp_set_aux:NNn \tl_gset:Nn }
10070 \cs_new_protected_nopar:Npn \fp_set_aux:NNn #1#2#3
10071 {
10072   \group_begin:
10073   \fp_split:Nn a {#3}
10074   \fp_standardise:NNNN
10075   \l_fp_input_a_sign_int
10076   \l_fp_input_a_integer_int
10077   \l_fp_input_a_decimal_int
10078   \l_fp_input_a_exponent_int
10079   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10080   \cs_set_protected_nopar:Npx \fp_tmp:w
10081   {
10082     \group_end:
```

```

10083      #1 \exp_not:N #2
10084      {
10085        \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10086        -
10087        \else:
10088        +
10089        \fi:
10090        \int_use:N \l_fp_input_a_integer_int
10091        .
10092        \exp_after:wN \use_none:n
10093        \int_use:N \l_fp_input_a_decimal_int
10094        e
10095        \int_use:N \l_fp_input_a_exponent_int
10096      }
10097    }
10098    \fp_tmp:w
10099  }
10100  \cs_generate_variant:Nn \fp_set:Nn { c }
10101  \cs_generate_variant:Nn \fp_gset:Nn { c }
  (End definition for \fp_set:Nn and \fp_set:cn. These functions are documented on page ??.)

```

\fp_set_from_dim:Nn Here, dimensions are converted to fixed-points *via* a temporary variable. This ensures
 \fp_set_from_dim:cn that they always convert as points. The code is then essentially the same as for \fp_
 \fp_gset_from_dim:Nn set:Nn, but with the dimension passed so that it will be striped of the pt on the way
 \fp_gset_from_dim:cn through. The passage through a skip is used to remove any rubber part.

```

\fp_set_from_dim_aux:NNn 10102 \cs_new_protected_nopar:Npn \fp_set_from_dim:Nn
\fp_set_from_dim_aux:w 10103 { \fp_set_from_dim_aux:NNn \tl_set:Nx }
  \l_fp_tmp_dim 10104 \cs_new_protected_nopar:Npn \fp_gset_from_dim:Nn
  \l_fp_tmp_skip 10105 { \fp_set_from_dim_aux:NNn \tl_gset:Nx }
10106 \cs_new_protected_nopar:Npn \fp_set_from_dim_aux:NNn #1#2#3
10107 {
10108   \group_begin:
10109   \l_fp_tmp_skip \etex_glueexpr:D #3 \scan_stop:
10110   \l_fp_tmp_dim \l_fp_tmp_skip
10111   \fp_split:Nn a
10112   {
10113     \exp_after:wN \fp_set_from_dim_aux:w
10114     \dim_use:N \l_fp_tmp_dim
10115   }
10116   \fp_standardise:NNNN
10117   \l_fp_input_a_sign_int
10118   \l_fp_input_a_integer_int
10119   \l_fp_input_a_decimal_int
10120   \l_fp_input_a_exponent_int
10121   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10122   \cs_set_protected_nopar:Npx \fp_tmp:w
10123   {
10124     \group_end:
10125     #1 \exp_not:N #2

```

```

10126         {
10127             \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10128             -
10129             \else:
10130             +
10131             \fi:
10132             \int_use:N \l_fp_input_a_integer_int
10133             .
10134             \exp_after:wN \use_none:n
10135             \int_use:N \l_fp_input_a_decimal_int
10136             e
10137             \int_use:N \l_fp_input_a_exponent_int
10138         }
10139     }
10140     \fp_tmp:w
10141 }
10142 \cs_set_protected_nopar:Npx \fp_set_from_dim_aux:w
10143 {
10144     \cs_set_nopar:Npn \exp_not:N \fp_set_from_dim_aux:w
10145     ##1 \tl_to_str:n { pt } {##1}
10146 }
10147 \fp_set_from_dim_aux:w
10148 \cs_generate_variant:Nn \fp_set_from_dim:Nn { c }
10149 \cs_generate_variant:Nn \fp_gset_from_dim:Nn { c }
10150 \dim_new:N \l_fp_tmp_dim
10151 \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 10152 \cs_new_eq:NN \fp_set_eq:NN \tl_set_eq:NN
\fp_set_eq:Nc 10153 \cs_new_eq:NN \fp_set_eq:cN \tl_set_eq:cN
\fp_set_eq:cc 10154 \cs_new_eq:NN \fp_set_eq:Nc \tl_set_eq:Nc
\fp_gset_eq:NN 10155 \cs_new_eq:NN \fp_set_eq:cc \tl_set_eq:cc
\fp_gset_eq:cN 10156 \cs_new_eq:NN \fp_gset_eq:NN \tl_gset_eq:NN
\fp_gset_eq:Nc 10157 \cs_new_eq:NN \fp_gset_eq:cN \tl_gset_eq:cN
\fp_gset_eq:Nc 10158 \cs_new_eq:NN \fp_gset_eq:Nc \tl_gset_eq:Nc
\fp_gset_eq:cc 10159 \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 10160 \cs_new_eq:NN \fp_show:N \tl_show:N
10161 \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 10162 \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

```



```

10163 { \exp_after:wN \fp_use_aux:w #1 \q_stop }
10164 \cs_generate_variant:Nn \fp_use:N { c }
10165 \cs_new_nopar:Npn \fp_use_aux:w #1#2 e #3 \q_stop
10166 {
10167   \if:w #1 -
10168     -
10169   \fi:
10170   \if_int_compare:w #3 > \c_zero
10171     \exp_after:wN \fp_use_large:w
10172   \else:
10173     \if_int_compare:w #3 < \c_zero
10174       \exp_after:wN \exp_after:wN \exp_after:wN
10175       \fp_use_small:w
10176     \else:
10177       \exp_after:wN \exp_after:wN \exp_after:wN \fp_use_none:w
10178     \fi:
10179   \fi:
10180   #2 e #3 \q_stop
10181 }

```

When the exponent is zero, the input is simply returned as output.

```

10182 \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.

```

10183 \cs_new_nopar:Npn \fp_use_small:w #1 . #2 e #3 \q_stop
10184 {
10185   0 .
10186   \prg_replicate:nn { -#3 - 1 } { 0 }
10187   #1#2
10188 }

```

Life is more complex for large numbers. The decimal point needs to be shuffled, with potentially some zero-filling for very large values.

```

10189 \cs_new_nopar:Npn \fp_use_large:w #1 . #2 e #3 \q_stop
10190 {
10191   \if_int_compare:w #3 < \c_ten
10192     \exp_after:wN \fp_use_large_aux_i:w
10193   \else:
10194     \exp_after:wN \fp_use_large_aux_ii:w
10195   \fi:
10196   #1#2 e #3 \q_stop
10197 }
10198 \cs_new_nopar:Npn \fp_use_large_aux_i:w #1#2 e #3 \q_stop
10199 {
10200   #1
10201   \use:c { fp_use_large_aux_ #3 :w } #2 \q_stop
10202 }
10203 \cs_new_nopar:cpn { fp_use_large_aux_1:w } #1#2 \q_stop { #1 . #2 }
10204 \cs_new_nopar:cpn { fp_use_large_aux_2:w } #1#2#3 \q_stop
10205 { #1#2 . #3 }

```

```

10206 \cs_new_nopar:cpn { fp_use_large_aux_3:w } #1#2#3#4 \q_stop
10207 { #1#2#3 . #4 }
10208 \cs_new_nopar:cpn { fp_use_large_aux_4:w } #1#2#3#4#5 \q_stop
10209 { #1#2#3#4 . #5 }
10210 \cs_new_nopar:cpn { fp_use_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10211 { #1#2#3#4#5 . #6 }
10212 \cs_new_nopar:cpn { fp_use_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10213 { #1#2#3#4#5#6 . #7 }
10214 \cs_new_nopar:cpn { fp_use_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10215 { #1#2#3#4#6#7 . #8 }
10216 \cs_new_nopar:cpn { fp_use_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10217 { #1#2#3#4#5#6#7#8 . #9 }
10218 \cs_new_nopar:cpn { fp_use_large_aux_9:w } #1 \q_stop { #1 . }
10219 \cs_new_nopar:Npn \fp_use_large_aux_ii:w #1 e #2 \q_stop
10220 {
10221   #1
10222   \prg_replicate:nn { #2 - 9 } { 0 }
10223   .
10224 }

```

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

197.6 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 10225 \cs_new_nopar:Npn \fp_to_dim:N #1 { \fp_use:N #1 pt }
10226 \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 10227 \cs_new_nopar:Npn \fp_to_int:N #1
\fp_to_int_aux:w 10228 { \exp_after:wN \fp_to_int_aux:w #1 \q_stop }
\fp_to_int_none:w 10229 \cs_generate_variant:Nn \fp_to_int:N { c }
\fp_to_int_small:w 10230 \cs_new_nopar:Npn \fp_to_int_aux:w #1#2 e #3 \q_stop
\fp_to_int_large:w 10231 {
\fp_to_int_large_aux_i:w 10232   \if:w #1 -
\fp_to_int_large_aux_1:w 10233   -
\fp_to_int_large_aux_2:w 10234   \fi:
\fp_to_int_large_aux_3:w 10235   \if_int_compare:w #3 < \c_zero
\fp_to_int_large_aux_4:w 10236   \exp_after:wN \fp_to_int_small:w
\fp_to_int_large_aux_5:w 10237   \else:
\fp_to_int_large_aux_6:w 10238   \exp_after:wN \fp_to_int_large:w
\fp_to_int_large_aux_7:w 10239   \fi:
\fp_to_int_large_aux_8:w 10240   #2 e #3 \q_stop
\fp_to_int_large_aux_i:w 10241 }
\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.

```

10242 \cs_new_nopar:Npn \fp_to_int_small:w #1 . #2 e #3 \q_stop
10243 {
10244   \if_int_compare:w #3 > \c_one
10245   \else:
10246     \if_int_compare:w #1 < \c_five
10247     0
10248   \else:
10249     1
10250   \fi:
10251 \fi:
10252 }

```

For large numbers, the idea is to split off the part for rounding, do the rounding and fill if needed.

```

10253 \cs_new_nopar:Npn \fp_to_int_large:w #1 . #2 e #3 \q_stop
10254 {
10255   \if_int_compare:w #3 < \c_ten
10256   \exp_after:wN \fp_to_int_large_aux_i:w
10257   \else:
10258   \exp_after:wN \fp_to_int_large_aux_ii:w
10259   \fi:
10260   #1#2 e #3 \q_stop
10261 }
10262 \cs_new_nopar:Npn \fp_to_int_large_aux_i:w #1#2 e #3 \q_stop
10263 { \use:c { fp_to_int_large_aux_#3 :w } #2 \q_stop {#1} }
10264 \cs_new_nopar:cpn { fp_to_int_large_aux_1:w } #1#2 \q_stop
10265 { \fp_to_int_large_aux:nnn { #2 0 } {#1} }
10266 \cs_new_nopar:cpn { fp_to_int_large_aux_2:w } #1#2#3 \q_stop
10267 { \fp_to_int_large_aux:nnn { #3 00 } {#1#2} }
10268 \cs_new_nopar:cpn { fp_to_int_large_aux_3:w } #1#2#3#4 \q_stop
10269 { \fp_to_int_large_aux:nnn { #4 000 } {#1#2#3} }
10270 \cs_new_nopar:cpn { fp_to_int_large_aux_4:w } #1#2#3#4#5 \q_stop
10271 { \fp_to_int_large_aux:nnn { #5 0000 } {#1#2#3#4} }
10272 \cs_new_nopar:cpn { fp_to_int_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10273 { \fp_to_int_large_aux:nnn { #6 00000 } {#1#2#3#4#5} }
10274 \cs_new_nopar:cpn { fp_to_int_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10275 { \fp_to_int_large_aux:nnn { #7 000000 } {#1#2#3#4#5#6} }
10276 \cs_new_nopar:cpn { fp_to_int_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10277 { \fp_to_int_large_aux:nnn { #8 0000000 } {#1#2#3#4#5#6#7} }
10278 \cs_new_nopar:cpn { fp_to_int_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10279 { \fp_to_int_large_aux:nnn { #9 00000000 } {#1#2#3#4#5#6#7#8} }
10280 \cs_new_nopar:cpn { fp_to_int_large_aux_9:w } #1 \q_stop {#1}
10281 \cs_new_nopar:Npn \fp_to_int_large_aux:nnn #1#2#3
10282 {
10283   \if_int_compare:w #1 < \c_five_hundred_million
10284   #3#2
10285   \else:
10286   \int_value:w \int_eval:w #3#2 + 1 \int_eval_end:

```

```

10287     \fi:
10288   }
10289   \cs_new_nopar:Npn \fp_to_int_large_aux_ii:w #1 e #2 \q_stop
10290   {
10291     #1
10292     \prg_replicate:nn { #2 - 9 } { 0 }
10293   }

```

(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:NNNNNNNN
\fp_use_ix:NNNNNNNN
\fp_use_i_to_vii:NNNNNNNN
\fp_use_i_to_iix:NNNNNNNN
10294   \cs_new_nopar:Npn \fp_to_tl:N #1
10295   { \exp_after:wN \fp_to_tl_aux:w #1 \q_stop }
10296   \cs_generate_variant:Nn \fp_to_tl:N { c }
10297   \cs_new_nopar:Npn \fp_to_tl_aux:w #1#2 e #3 \q_stop
10298   {
10299     \if:w #1 -
10300     -
10301     \fi:
10302     \if_int_compare:w #3 < \c_zero
10303       \exp_after:wN \fp_to_tl_small:w
10304     \else:
10305       \exp_after:wN \fp_to_tl_large:w
10306     \fi:
10307     #2 e #3 \q_stop
10308   }
For “large” numbers (exponent  $\geq 0$ ) there are two cases. For very large exponents ( $\geq 10$ )
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.
10309   \cs_new_nopar:Npn \fp_to_tl_large:w #1 e #2 \q_stop
10310   {
10311     \if_int_compare:w #2 < \c_ten
10312       \exp_after:wN \fp_to_tl_large_aux_i:w
10313     \else:
10314       \exp_after:wN \fp_to_tl_large_aux_ii:w
10315     \fi:
10316     #1 e #2 \q_stop
10317   }
10318   \cs_new_nopar:Npn \fp_to_tl_large_aux_i:w #1 e #2 \q_stop
10319   { \use:c { fp_to_tl_large_#2 :w } #1 \q_stop }
10320   \cs_new_nopar:Npn \fp_to_tl_large_aux_ii:w #1 . #2 e #3 \q_stop
10321   {
10322     #1
10323     \fp_to_tl_large_zeros:NNNNNNNN #2
10324     e #3
10325   }
10326   \cs_new_nopar:cpn { fp_to_tl_large_0:w } #1 . #2 \q_stop
10327   {

```

```

10328     #1
10329     \fp_to_tl_large_zeros:NNNNNNNNN #2
10330   }
10331   \cs_new_nopar:cpn { fp_to_tl_large_1:w } #1 . #2#3 \q_stop
10332   {
10333     #1#2
10334     \fp_to_tl_large_zeros:NNNNNNNNN #3 0
10335   }
10336   \cs_new_nopar:cpn { fp_to_tl_large_2:w } #1 . #2#3#4 \q_stop
10337   {
10338     #1#2#3
10339     \fp_to_tl_large_zeros:NNNNNNNNN #4 00
10340   }
10341   \cs_new_nopar:cpn { fp_to_tl_large_3:w } #1 . #2#3#4#5 \q_stop
10342   {
10343     #1#2#3#4
10344     \fp_to_tl_large_zeros:NNNNNNNNN #5 000
10345   }
10346   \cs_new_nopar:cpn { fp_to_tl_large_4:w } #1 . #2#3#4#5#6 \q_stop
10347   {
10348     #1#2#3#4#5
10349     \fp_to_tl_large_zeros:NNNNNNNNN #6 0000
10350   }
10351   \cs_new_nopar:cpn { fp_to_tl_large_5:w } #1 . #2#3#4#5#6#7 \q_stop
10352   {
10353     #1#2#3#4#5#6
10354     \fp_to_tl_large_zeros:NNNNNNNNN #7 00000
10355   }
10356   \cs_new_nopar:cpn { fp_to_tl_large_6:w } #1 . #2#3#4#5#6#7#8 \q_stop
10357   {
10358     #1#2#3#4#5#6#7
10359     \fp_to_tl_large_zeros:NNNNNNNNN #8 000000
10360   }
10361   \cs_new_nopar:cpn { fp_to_tl_large_7:w } #1 . #2#3#4#5#6#7#8#9 \q_stop
10362   {
10363     #1#2#3#4#5#6#7#8
10364     \fp_to_tl_large_zeros:NNNNNNNNN #9 0000000
10365   }
10366   \cs_new_nopar:cpn { fp_to_tl_large_8:w } #1 .
10367   {
10368     #1
10369     \use:c { fp_to_tl_large_8_aux:w }
10370   }
10371   \cs_new_nopar:cpn { fp_to_tl_large_8_aux:w } #1#2#3#4#5#6#7#8#9 \q_stop
10372   {
10373     #1#2#3#4#5#6#7#8
10374     \fp_to_tl_large_zeros:NNNNNNNNN #9 00000000
10375   }
10376   \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.

```

10377 \cs_new_nopar:Npn \fp_to_tl_small:w #1 e #2 \q_stop
10378 {
10379   \if_int_compare:w #2 = \c_minus_one
10380     \exp_after:wN \fp_to_tl_small_one:w
10381   \else:
10382     \if_int_compare:w #2 = -\c_two
10383       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_two:w
10384     \else:
10385       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_aux:w
10386     \fi:
10387   \fi:
10388   #1 e #2 \q_stop
10389 }
10390 \cs_new_nopar:Npn \fp_to_tl_small_one:w #1 . #2 e #3 \q_stop
10391 {
10392   \if_int_compare:w \fp_use_ix:NNNNNNNN #2 > \c_four
10393     \if_int_compare:w
10394       \int_eval:w #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10395       < \c_one_thousand_million
10396       0.
10397     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10398     \int_value:w \int_eval:w
10399     #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10400     \int_eval_end:
10401   \else:
10402     1
10403   \fi:
10404   \else:
10405     0. #1
10406     \fp_to_tl_small_zeros:NNNNNNNN #2
10407   \fi:
10408 }
10409 \cs_new_nopar:Npn \fp_to_tl_small_two:w #1 . #2 e #3 \q_stop
10410 {
10411   \if_int_compare:w \fp_use_iix_ix:NNNNNNNN #2 > \c_forty_four
10412     \if_int_compare:w
10413       \int_eval:w #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10414       < \c_one_thousand_million
10415       0.0
10416     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10417     \int_value:w \int_eval:w
10418     #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10419     \int_eval_end:
10420   \else:
10421     0.1
10422   \fi:

```

```

10423     \else:
10424         0.0
10425         #1
10426         \fp_to_tl_small_zeros:NNNNNNNN #2
10427     \fi:
10428 }
10429 \cs_new_nopar:Npn \fp_to_tl_small_aux:w #1 . #2 e #3 \q_stop
10430 {
10431     #1
10432     \fp_to_tl_large_zeros:NNNNNNNN #2
10433     e #3
10434 }

```

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.

```

10435 \cs_new_nopar:Npn \fp_to_tl_large_zeros:NNNNNNNN #1#2#3#4#5#6#7#8#9
10436 {
10437     \if_int_compare:w #9 = \c_zero
10438     \if_int_compare:w #8 = \c_zero
10439     \if_int_compare:w #7 = \c_zero
10440     \if_int_compare:w #6 = \c_zero
10441     \if_int_compare:w #5 = \c_zero
10442     \if_int_compare:w #4 = \c_zero
10443     \if_int_compare:w #3 = \c_zero
10444     \if_int_compare:w #2 = \c_zero
10445     \if_int_compare:w #1 = \c_zero
10446     \else:
10447         . #1
10448     \fi:
10449     \else:
10450         . #1#2
10451     \fi:
10452     \else:
10453         . #1#2#3
10454     \fi:
10455     \else:
10456         . #1#2#3#4
10457     \fi:
10458     \else:
10459         . #1#2#3#4#5
10460     \fi:
10461     \else:
10462         . #1#2#3#4#5#6
10463     \fi:
10464     \else:
10465         . #1#2#3#4#5#6#7
10466     \fi:
10467     \else:
10468         . #1#2#3#4#5#6#7#8
10469     \fi:

```

```

10470     \else:
10471         . #1#2#3#4#5#6#7#8#9
10472     \fi:
10473 }
10474 \cs_new_nopar:Npn \fp_to_tl_small_zeros:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10475 {
10476     \if_int_compare:w #9 = \c_zero
10477     \if_int_compare:w #8 = \c_zero
10478     \if_int_compare:w #7 = \c_zero
10479     \if_int_compare:w #6 = \c_zero
10480     \if_int_compare:w #5 = \c_zero
10481     \if_int_compare:w #4 = \c_zero
10482     \if_int_compare:w #3 = \c_zero
10483     \if_int_compare:w #2 = \c_zero
10484     \if_int_compare:w #1 = \c_zero
10485     \else:
10486         #1
10487     \fi:
10488     \else:
10489         #1#2
10490     \fi:
10491     \else:
10492         #1#2#3
10493     \fi:
10494     \else:
10495         #1#2#3#4
10496     \fi:
10497     \else:
10498         #1#2#3#4#5
10499     \fi:
10500     \else:
10501         #1#2#3#4#5#6
10502     \fi:
10503     \else:
10504         #1#2#3#4#5#6#7
10505     \fi:
10506     \else:
10507         #1#2#3#4#5#6#7#8
10508     \fi:
10509     \else:
10510         #1#2#3#4#5#6#7#8#9
10511     \fi:
10512 }

```

Some quick “return a few” functions.

```

10513 \cs_new_nopar:Npn \fp_use_iix_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#8#9}
10514 \cs_new_nopar:Npn \fp_use_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#9}
10515 \cs_new_nopar:Npn \fp_use_i_to_vii:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10516     {#1#2#3#4#5#6#7}
10517 \cs_new_nopar:Npn \fp_use_i_to_iix:NNNNNNNNN #1#2#3#4#5#6#7#8#9

```



```
10518 {#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 ??.)

197.7 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
10519 \cs_new_protected_nopar:Npn \fp_round_figures:Nn
10520 { \fp_round_figures_aux:NNn \tl_set:Nn }
10521 \cs_generate_variant:Nn \fp_round_figures:Nn { c }
10522 \cs_new_protected_nopar:Npn \fp_ground_figures:Nn
10523 { \fp_round_figures_aux:NNn \tl_gset:Nn }
10524 \cs_generate_variant:Nn \fp_ground_figures:Nn { c }
10525 \cs_new_protected_nopar:Npn \fp_round_figures_aux:NNn #1#2#3
10526 {
10527   \group_begin:
10528   \fp_read:N #2
10529   \int_set:Nn \l_fp_round_target_int { #3 - 1 }
10530   \if_int_compare:w \l_fp_round_target_int < \c_ten
10531     \exp_after:wN \fp_round:
10532   \fi:
10533   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10534   \cs_set_protected_nopar:Npx \fp_tmp:w
10535   {
10536     \group_end:
10537     #1 \exp_not:N #2
10538     {
10539       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10540         -
10541       \else:
10542         +
10543       \fi:
10544       \int_use:N \l_fp_input_a_integer_int
10545       .
10546       \exp_after:wN \use_none:n
10547       \int_use:N \l_fp_input_a_decimal_int
10548       e
10549       \int_use:N \l_fp_input_a_exponent_int
10550     }
10551   }
10552   \fp_tmp:w
10553 }
```

(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`

```

10554 \cs_new_protected_nopar:Npn \fp_round_places:Nn
10555 { \fp_round_places_aux:NNn \tl_set:Nn }
10556 \cs_generate_variant:Nn \fp_round_places:Nn { c }
10557 \cs_new_protected_nopar:Npn \fp_ground_places:Nn
10558 { \fp_round_places_aux:NNn \tl_gset:Nn }
10559 \cs_generate_variant:Nn \fp_ground_places:Nn { c }
10560 \cs_new_protected_nopar:Npn \fp_round_places_aux:NNn #1#2#3
10561 {
10562   \group_begin:
10563   \fp_read:N #2
10564   \int_set:Nn \l_fp_round_target_int
10565     { #3 + \l_fp_input_a_exponent_int }
10566   \if_int_compare:w \l_fp_round_target_int < \c_ten
10567     \exp_after:wN \fp_round:
10568   \fi:
10569   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10570   \cs_set_protected_nopar:Npx \fp_tmp:w
10571   {
10572     \group_end:
10573     #1 \exp_not:N #2
10574     {
10575       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10576         -
10577       \else:
10578         +
10579       \fi:
10580       \int_use:N \l_fp_input_a_integer_int
10581       .
10582       \exp_after:wN \use_none:n
10583       \int_use:N \l_fp_input_a_decimal_int
10584       e
10585       \int_use:N \l_fp_input_a_exponent_int
10586     }
10587   }
10588   \fp_tmp:w
10589 }

```

(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.

```

10590 \cs_new_protected_nopar:Npn \fp_round:
10591 {
10592   \bool_set_false:N \l_fp_round_carry_bool
10593   \l_fp_round_position_int \c_eight

```

```

10594 \tl_clear:N \l_fp_round_decimal_tl
10595 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10596 \exp_after:wN \use_i:nn \exp_after:wN
10597 \fp_round_aux:NNNNNNNN \int_use:N \l_fp_input_a_decimal_int
10598 }
10599 \cs_new_protected_nopar:Npn \fp_round_aux:NNNNNNNN #1#2#3#4#5#6#7#8#9
10600 {
10601 \fp_round_loop:N #9#8#7#6#5#4#3#2#1
10602 \bool_if:NT \l_fp_round_carry_bool
10603 { \tex_advance:D \l_fp_input_a_integer_int \c_one }
10604 \l_fp_input_a_decimal_int \l_fp_round_decimal_tl \scan_stop:
10605 \if_int_compare:w \l_fp_input_a_integer_int < \c_ten
10606 \else:
10607 \l_fp_input_a_integer_int \c_one
10608 \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10609 \tex_advance:D \l_fp_input_a_exponent_int \c_one
10610 \fi:
10611 }
10612 \cs_new_protected_nopar:Npn \fp_round_loop:N #1
10613 {
10614 \if_int_compare:w \l_fp_round_position_int < \l_fp_round_target_int
10615 \bool_if:NTF \l_fp_round_carry_bool
10616 { \l_fp_tmp_int \int_eval:w #1 + \c_one \scan_stop: }
10617 { \l_fp_tmp_int \int_eval:w #1 \scan_stop: }
10618 \if_int_compare:w \l_fp_tmp_int = \c_ten
10619 \l_fp_tmp_int \c_zero
10620 \else:
10621 \bool_set_false:N \l_fp_round_carry_bool
10622 \fi:
10623 \tl_set:Nx \l_fp_round_decimal_tl
10624 { \int_use:N \l_fp_tmp_int \l_fp_round_decimal_tl }
10625 \else:
10626 \tl_set:Nx \l_fp_round_decimal_tl { 0 \l_fp_round_decimal_tl }
10627 \if_int_compare:w \l_fp_round_position_int = \l_fp_round_target_int
10628 \if_int_compare:w #1 > \c_four
10629 \bool_set_true:N \l_fp_round_carry_bool
10630 \fi:
10631 \fi:
10632 \fi:
10633 \tex_advance:D \l_fp_round_position_int \c_minus_one
10634 \if_int_compare:w \l_fp_round_position_int > \c_minus_one
10635 \exp_after:wN \fp_round_loop:N
10636 \fi:
10637 }

```

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

197.8 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

```

10638 \cs_new_protected_nopar:Npn \fp_abs:N { \fp_abs_aux:NN \tl_set:Nn }
10639 \cs_new_protected_nopar:Npn \fp_gabs:N { \fp_abs_aux:NN \tl_gset:Nn }
10640 \cs_generate_variant:Nn \fp_abs:N { c }
10641 \cs_generate_variant:Nn \fp_gabs:N { c }
10642 \cs_new_protected_nopar:Npn \fp_abs_aux:NN #1#2
10643 {
10644   \group_begin:
10645   \fp_read:N #2
10646   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10647   \cs_set_protected_nopar:Npx \fp_tmp:w
10648   {
10649     \group_end:
10650     #1 \exp_not:N #2
10651     {
10652       +
10653       \int_use:N \l_fp_input_a_integer_int
10654       .
10655       \exp_after:wN \use_none:n
10656       \int_use:N \l_fp_input_a_decimal_int
10657       e
10658       \int_use:N \l_fp_input_a_exponent_int
10659     }
10660   }
10661   \fp_tmp:w
10662 }

```

(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 10663 \cs_new_protected_nopar:Npn \fp_neg:N { \fp_neg_aux:NN \tl_set:Nn }
\fp_gneg:N 10664 \cs_new_protected_nopar:Npn \fp_gneg:N { \fp_neg_aux:NN \tl_gset:Nn }
\fp_gneg:c 10665 \cs_generate_variant:Nn \fp_neg:N { c }
\fp_neg:NN 10666 \cs_generate_variant:Nn \fp_gneg:N { c }
10667 \cs_new_protected_nopar:Npn \fp_neg_aux:NN #1#2
10668 {
10669   \group_begin:
10670   \fp_read:N #2
10671   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10672   \tl_set:Nx \l_fp_tmp_tl
10673   {
10674     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10675     +
10676     \else:
10677     -
10678     \fi:
10679     \int_use:N \l_fp_input_a_integer_int
10680     .
10681     \exp_after:wN \use_none:n
10682     \int_use:N \l_fp_input_a_decimal_int
10683     e
10684     \int_use:N \l_fp_input_a_exponent_int

```

```

10685     }
10686     \exp_after:wN \group_end: \exp_after:wN
10687     #1 \exp_after:wN #2 \exp_after:wN { \l_fp_tmp_tl }
10688 }

```

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

197.9 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.

```

10693 \cs_new_protected_nopar:Npn \fp_add_aux:NNn #1#2#3
10694 {
10695   \group_begin:
10696   \fp_read:N #2
10697   \fp_split:Nn b {#3}
10698   \fp_standardise:NNNN
10699   \l_fp_input_b_sign_int
10700   \l_fp_input_b_integer_int
10701   \l_fp_input_b_decimal_int
10702   \l_fp_input_b_exponent_int
10703   \fp_add_core:
10704   \fp_tmp:w #1#2
10705 }
10706 \cs_new_protected_nopar:Npn \fp_add_core:
10707 {
10708   \fp_level_input_exponents:
10709   \if_int_compare:w
10710     \int_eval:w
10711       \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
10712       > \c_zero
10713     \exp_after:wN \fp_add_sum:
10714   \else:
10715     \exp_after:wN \fp_add_difference:
10716   \fi:
10717   \l_fp_output_exponent_int \l_fp_input_a_exponent_int
10718   \fp_standardise:NNNN
10719   \l_fp_output_sign_int
10720   \l_fp_output_integer_int
10721   \l_fp_output_decimal_int
10722   \l_fp_output_exponent_int
10723   \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2

```

```

10724 {
10725   \group_end:
10726   ##1 ##2
10727   {
10728     \if_int_compare:w \l_fp_output_sign_int < \c_zero
10729     -
10730     \else:
10731     +
10732     \fi:
10733     \int_use:N \l_fp_output_integer_int
10734     .
10735     \exp_after:wN \use_none:n
10736     \int_value:w \int_eval:w
10737       \l_fp_output_decimal_int + \c_one_thousand_million
10738     e
10739     \int_use:N \l_fp_output_exponent_int
10740   }
10741 }
10742 }

```

Finding the sum of two numbers is trivially easy.

```

10743 \cs_new_protected_nopar:Npn \fp_add_sum:
10744 {
10745   \l_fp_output_sign_int \l_fp_input_a_sign_int
10746   \l_fp_output_integer_int
10747   \int_eval:w
10748     \l_fp_input_a_integer_int + \l_fp_input_b_integer_int
10749   \scan_stop:
10750   \l_fp_output_decimal_int
10751   \int_eval:w
10752     \l_fp_input_a_decimal_int + \l_fp_input_b_decimal_int
10753   \scan_stop:
10754   \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
10755   \else:
10756     \tex_advance:D \l_fp_output_integer_int \c_one
10757     \tex_advance:D \l_fp_output_decimal_int -\c_one_thousand_million
10758   \fi:
10759 }

```

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.

```

10760 \cs_new_protected_nopar:Npn \fp_add_difference:
10761 {
10762   \l_fp_output_integer_int
10763   \int_eval:w
10764     \l_fp_input_a_integer_int - \l_fp_input_b_integer_int
10765   \scan_stop:
10766   \l_fp_output_decimal_int

```

```

10767 \int_eval:w
10768 \l_fp_input_a_decimal_int - \l_fp_input_b_decimal_int
10769 \scan_stop:
10770 \if_int_compare:w \l_fp_output_decimal_int < \c_zero
10771 \tex_advance:D \l_fp_output_integer_int \c_minus_one
10772 \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
10773 \fi:
10774 \if_int_compare:w \l_fp_output_integer_int < \c_zero
10775 \l_fp_output_sign_int \l_fp_input_b_sign_int
10776 \if_int_compare:w \l_fp_output_decimal_int = \c_zero
10777 \l_fp_output_integer_int -\l_fp_output_integer_int
10778 \else:
10779 \l_fp_output_decimal_int
10780 \int_eval:w
10781 \c_one_thousand_million - \l_fp_output_decimal_int
10782 \scan_stop:
10783 \l_fp_output_integer_int
10784 \int_eval:w
10785 - \l_fp_output_integer_int - \c_one
10786 \scan_stop:
10787 \fi:
10788 \else:
10789 \l_fp_output_sign_int \l_fp_input_a_sign_int
10790 \fi:
10791 }

```

(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 10792 \cs_new_protected_nopar:Npn \fp_sub:Nn { \fp_sub_aux:NNn \tl_set:Nn }
10793 \cs_new_protected_nopar:Npn \fp_gsub:Nn { \fp_sub_aux:NNn \tl_gset:Nn }
10794 \cs_generate_variant:Nn \fp_sub:Nn { c }
10795 \cs_generate_variant:Nn \fp_gsub:Nn { c }
10796 \cs_new_protected_nopar:Npn \fp_sub_aux:NNn #1#2#3
10797 {
10798 \group_begin:
10799 \fp_read:N #2
10800 \fp_split:Nn b {#3}
10801 \fp_standardise:NNNN
10802 \l_fp_input_b_sign_int
10803 \l_fp_input_b_integer_int
10804 \l_fp_input_b_decimal_int
10805 \l_fp_input_b_exponent_int
10806 \tex_multiply:D \l_fp_input_b_sign_int \c_minus_one
10807 \fp_add_core:
10808 \fp_tmp:w #1#2
10809 }

```

(End definition for \fp_sub:Nn and \fp_sub:cn. These functions are documented on page ??.)

<pre> \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 </pre>	<p>The pattern is much the same for multiplication.</p> <pre> 10810 \cs_new_protected_nopar:Npn \fp_mul:Nn { \fp_mul_aux:NNn \tl_set:Nn } 10811 \cs_new_protected_nopar:Npn \fp_gmul:Nn { \fp_mul_aux:NNn \tl_gset:Nn } 10812 \cs_generate_variant:Nn \fp_mul:Nn { c } 10813 \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> 10814 \cs_new_protected_nopar:Npn \fp_mul_aux:NNn #1#2#3 10815 { 10816 \group_begin: 10817 \fp_read:N #2 10818 \fp_split:Nn b {#3} 10819 \fp_standardise:NNNN 10820 \l_fp_input_b_sign_int 10821 \l_fp_input_b_integer_int 10822 \l_fp_input_b_decimal_int 10823 \l_fp_input_b_exponent_int 10824 \fp_mul_internal: 10825 \l_fp_output_exponent_int 10826 \int_eval:w 10827 \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int 10828 \scan_stop: 10829 \fp_standardise:NNNN 10830 \l_fp_output_sign_int 10831 \l_fp_output_integer_int 10832 \l_fp_output_decimal_int 10833 \l_fp_output_exponent_int 10834 \cs_set_protected_nopar:Npx \fp_tmp:w 10835 { 10836 \group_end: 10837 #1 \exp_not:N #2 10838 { 10839 \if_int_compare:w 10840 \int_eval:w 10841 \l_fp_input_a_sign_int * \l_fp_input_b_sign_int 10842 < \c_zero 10843 \if_int_compare:w 10844 \int_eval:w 10845 \l_fp_output_integer_int + \l_fp_output_decimal_int 10846 = \c_zero 10847 + 10848 \else: 10849 - 10850 \fi: 10851 \else: 10852 + 10853 \fi: </pre>
---	--


```

10854         \int_use:N \l_fp_output_integer_int
10855         .
10856         \exp_after:wN \use_none:n
10857         \int_value:w \int_eval:w
10858         \l_fp_output_decimal_int + \c_one_thousand_million
10859         e
10860         \int_use:N \l_fp_output_exponent_int
10861     }
10862 }
10863 \fp_tmp:w
10864 }

```

Done separately so that the internal use is a bit easier.

```

10865 \cs_new_protected_nopar:Npn \fp_mul_internal:
10866 {
10867     \fp_mul_split:NNNN \l_fp_input_a_decimal_int
10868     \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
10869     \fp_mul_split:NNNN \l_fp_input_b_decimal_int
10870     \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
10871     \l_fp_mul_output_int \c_zero
10872     \tl_clear:N \l_fp_mul_output_tl
10873     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_iii_int
10874     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_ii_int
10875     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_i_int
10876     \tex_divide:D \l_fp_mul_output_int \c_one_thousand
10877     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_iii_int
10878     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_ii_int
10879     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_i_int
10880     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_input_b_integer_int
10881     \fp_mul_end_level:
10882     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_ii_int
10883     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_i_int
10884     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_input_b_integer_int
10885     \fp_mul_end_level:
10886     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_i_int
10887     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_input_b_integer_int
10888     \fp_mul_end_level:
10889     \l_fp_output_decimal_int 0 \l_fp_mul_output_tl \scan_stop:
10890     \tl_clear:N \l_fp_mul_output_tl
10891     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_input_b_integer_int
10892     \fp_mul_end_level:
10893     \l_fp_output_integer_int 0 \l_fp_mul_output_tl \scan_stop:
10894 }

```

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.

```

10895 \cs_new_protected_nopar:Npn \fp_mul_split:NNNN #1#2#3#4
10896 {

```

```

10897 \tex_advance:D #1 \c_one_thousand_million
10898 \cs_set_protected_nopar:Npn \fp_mul_split_aux:w
10899   ##1##2##3##4##5##6##7##8##9 \q_stop {
10900     #2 ##2##3##4 \scan_stop:
10901     #3 ##5##6##7 \scan_stop:
10902     #4 ##8##9 \scan_stop:
10903   }
10904 \exp_after:wN \fp_mul_split_aux:w \int_use:N #1 \q_stop
10905 \tex_advance:D #1 -\c_one_thousand_million
10906 }
10907 \cs_new_protected_nopar:Npn \fp_mul_product:NN #1#2
10908 {
10909   \l_fp_mul_output_int
10910   \int_eval:w \l_fp_mul_output_int + #1 * #2 \scan_stop:
10911 }

```

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.

```

10912 \cs_new_protected_nopar:Npn \fp_mul_end_level:
10913 {
10914   \tex_advance:D \l_fp_mul_output_int \c_one_thousand_million
10915   \exp_after:wN \use_i:nn \exp_after:wN
10916   \fp_mul_end_level:NNNNNNNN \int_use:N \l_fp_mul_output_int
10917 }
10918 \cs_new_protected_nopar:Npn \fp_mul_end_level:NNNNNNNN #1#2#3#4#5#6#7#8#9
10919 {
10920   \tl_set:Nx \l_fp_mul_output_tl { #7#8#9 \l_fp_mul_output_tl }
10921   \l_fp_mul_output_int #1#2#3#4#5#6 \scan_stop:
10922 }

```

(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 10923 \cs_new_protected_nopar:Npn \fp_div:Nn { \fp_div_aux:NNn \tl_set:Nn }
\fp_gdiv:Nn 10924 \cs_new_protected_nopar:Npn \fp_gdiv:Nn { \fp_div_aux:NNn \tl_gset:Nn }
\fp_gdiv:cn 10925 \cs_generate_variant:Nn \fp_div:Nn { c }
\fp_div_aux:NNn 10926 \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_loop:
\fp_div_divide:
\fp_div_divide_aux: 10927 \cs_new_protected_nopar:Npn \fp_div_aux:NNn #1#2#3
\fp_div_store: 10928 {
\fp_div_store_integer: 10929   \group_begin:
\fp_div_store_decimal: 10930     \fp_read:N #2
10931     \fp_split:Nn b {#3}
10932     \fp_standardise:NNNN
10933     \l_fp_input_b_sign_int
10934     \l_fp_input_b_integer_int
10935     \l_fp_input_b_decimal_int

```

```

10936         \l_fp_input_b_exponent_int
10937     \if_int_compare:w
10938         \int_eval:w
10939         \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
10940         = \c_zero
10941     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
10942     {
10943         \group_end:
10944         #1 \exp_not:N #2 { \c_undefined_fp }
10945     }
10946 \else:
10947     \if_int_compare:w
10948         \int_eval:w
10949         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
10950         = \c_zero
10951     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
10952     {
10953         \group_end:
10954         #1 \exp_not:N #2 { \c_zero_fp }
10955     }
10956 \else:
10957     \exp_after:wN \exp_after:wN \exp_after:wN \fp_div_internal:
10958 \fi:
10959 \fi:
10960 \fp_tmp:w #1#2
10961 }

```

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.

```

10962 \cs_new_protected_nopar:Npn \fp_div_internal: {
10963     \l_fp_output_integer_int \c_zero
10964     \l_fp_output_decimal_int \c_zero
10965     \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
10966     \l_fp_div_offset_int \c_one_hundred_million
10967     \fp_div_loop:
10968     \l_fp_output_exponent_int
10969     \int_eval:w
10970     \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
10971     \scan_stop:
10972     \fp_standardise:NNNN
10973     \l_fp_output_sign_int
10974     \l_fp_output_integer_int
10975     \l_fp_output_decimal_int
10976     \l_fp_output_exponent_int
10977     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
10978     {
10979         \group_end:

```

```

10980     ##1 ##2
10981     {
10982         \if_int_compare:w
10983         \int_eval:w
10984         \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
10985         < \c_zero
10986         \if_int_compare:w
10987         \int_eval:w
10988         \l_fp_output_integer_int + \l_fp_output_decimal_int
10989         = \c_zero
10990         +
10991         \else:
10992         -
10993         \fi:
10994     \else:
10995     +
10996     \fi:
10997     \int_use:N \l_fp_output_integer_int
10998     .
10999     \exp_after:wN \use_none:n
11000     \int_value:w \int_eval:w
11001     \l_fp_output_decimal_int + \c_one_thousand_million
11002     \int_eval_end:
11003     e
11004     \int_use:N \l_fp_output_exponent_int
11005 }
11006 }
11007 }

```

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.

```

11008 \cs_new_protected_nopar:Npn \fp_div_loop:
11009 {
11010     \l_fp_count_int \c_zero
11011     \fp_div_divide:
11012     \fp_div_store:
11013     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11014     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11015     \exp_after:wN \fp_div_loop_step:w
11016     \int_use:N \l_fp_input_a_decimal_int \q_stop
11017     \if_int_compare:w
11018     \int_eval:w \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11019     > \c_zero
11020     \if_int_compare:w \l_fp_div_offset_int > \c_zero
11021     \exp_after:wN \exp_after:wN \exp_after:wN
11022     \fp_div_loop:
11023     \fi:
11024 \fi:
11025 }

```

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.

```

11026 \cs_new_protected_nopar:Npn \fp_div_divide:
11027 {
11028   \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
11029     \exp_after:wN \fp_div_divide_aux:
11030   \else:
11031     \if_int_compare:w \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
11032     \else:
11033       \if_int_compare:w
11034         \l_fp_input_a_decimal_int < \l_fp_input_b_decimal_int
11035       \else:
11036         \exp_after:wN \exp_after:wN \exp_after:wN
11037         \exp_after:wN \exp_after:wN \exp_after:wN
11038         \exp_after:wN \fp_div_divide_aux:
11039       \fi:
11040     \fi:
11041   \fi:
11042 }
11043 \cs_new_protected_nopar:Npn \fp_div_divide_aux:
11044 {
11045   \tex_advance:D \l_fp_count_int \c_one
11046   \tex_advance:D \l_fp_input_a_integer_int -\l_fp_input_b_integer_int
11047   \tex_advance:D \l_fp_input_a_decimal_int -\l_fp_input_b_decimal_int
11048   \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11049     \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11050     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11051   \fi:
11052   \fp_div_divide:
11053 }

```

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.

```

11054 \cs_new_protected_nopar:Npn \fp_div_store: { }
11055 \cs_new_protected_nopar:Npn \fp_div_store_integer:
11056 {
11057   \l_fp_output_integer_int \l_fp_count_int
11058   \cs_set_eq:NN \fp_div_store: \fp_div_store_decimal:
11059 }
11060 \cs_new_protected_nopar:Npn \fp_div_store_decimal:
11061 {
11062   \l_fp_output_decimal_int
11063   \int_eval:w
11064     \l_fp_output_decimal_int +
11065     \l_fp_count_int * \l_fp_div_offset_int
11066   \int_eval_end:
11067   \tex_divide:D \l_fp_div_offset_int \c_ten
11068 }

```

```

11069 \cs_new_protected_nopar:Npn \fp_div_loop_step:w #1#2#3#4#5#6#7#8#9 \q_stop
11070 {
11071   \l_fp_input_a_integer_int
11072   \int_eval:w #2 + \l_fp_input_a_integer_int \int_eval_end:
11073   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11074 }

```

(End definition for `\fp_div:Nn` and `\fp_div:cn`. These functions are documented on page ??.)

197.10 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.

```

11075 \cs_new_protected_nopar:Npn \fp_add:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11076 {
11077   #7 \int_eval:w #1 + #4 \int_eval_end:
11078   #8 \int_eval:w #2 + #5 \int_eval_end:
11079   #9 \int_eval:w #3 + #6 \int_eval_end:
11080   \if_int_compare:w #9 < \c_one_thousand_million
11081   \else:
11082     \tex_advance:D #8 \c_one
11083     \tex_advance:D #9 -\c_one_thousand_million
11084   \fi:
11085   \if_int_compare:w #8 < \c_one_thousand_million
11086   \else:
11087     \tex_advance:D #7 \c_one

```

```

11088     \tex_advance:D #8 -\c_one_thousand_million
11089     \fi:
11090 }

```

(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.

```

11091 \cs_new_protected_nopar:Npn \fp_sub:NNNNNNNN #1#2#3#4#5#6#7#8#9
11092 {
11093   #7 \int_eval:w #1 - #4 \int_eval_end:
11094   #8 \int_eval:w #2 - #5 \int_eval_end:
11095   #9 \int_eval:w #3 - #6 \int_eval_end:
11096   \if_int_compare:w #9 < \c_zero
11097     \tex_advance:D #8 \c_minus_one
11098     \tex_advance:D #9 \c_one_thousand_million
11099   \fi:
11100   \if_int_compare:w #8 < \c_zero
11101     \tex_advance:D #7 \c_minus_one
11102     \tex_advance:D #8 \c_one_thousand_million
11103   \fi:
11104   \if_int_compare:w #7 < \c_zero
11105     \if_int_compare:w \int_eval:w #8 + #9 = \c_zero
11106       #7 -#7
11107   \else:
11108     \tex_advance:D #7 \c_one
11109     #8 \int_eval:w \c_one_thousand_million - #8 \int_eval_end:
11110     #9 \int_eval:w \c_one_thousand_million - #9 \int_eval_end:
11111   \fi:
11112   \fi:
11113 }

```

(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.

```

11114 \cs_new_protected_nopar:Npn \fp_mul:NNNNNN #1#2#3#4#5#6
11115 {
11116   \fp_mul_split:NNNN #1
11117   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11118   \fp_mul_split:NNNN #2
11119   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11120   \fp_mul_split:NNNN #3
11121   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11122   \fp_mul_split:NNNN #4
11123   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11124   \l_fp_mul_output_int \c_zero
11125   \tl_clear:N \l_fp_mul_output_tl
11126   \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_vi_int
11127   \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_v_int
11128   \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_iv_int

```

```

11129 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int
11130 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11131 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11132 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11133 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11134 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11135 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11136 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11137 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11138 \fp_mul_end_level:
11139 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11140 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11141 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11142 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11143 \fp_mul_end_level:
11144 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11145 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11146 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11147 \fp_mul_end_level:
11148 #6 0 \l_fp_mul_output_tl \scan_stop:
11149 \tl_clear:N \l_fp_mul_output_tl
11150 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11151 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11152 \fp_mul_end_level:
11153 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11154 \fp_mul_end_level:
11155 \fp_mul_end_level:
11156 #5 0 \l_fp_mul_output_tl \scan_stop:
11157 }

```

(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.

```

11158 \cs_new_protected_nopar:Npn \fp_mul:NNNNNNNN #1#2#3#4#5#6#7#8#9
11159 {
11160   \fp_mul_split:NNNN #2
11161   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11162   \fp_mul_split:NNNN #3
11163   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11164   \fp_mul_split:NNNN #5
11165   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11166   \fp_mul_split:NNNN #6
11167   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11168   \l_fp_mul_output_int \c_zero
11169   \tl_clear:N \l_fp_mul_output_tl
11170   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_vi_int
11171   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_v_int
11172   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iv_int
11173   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int

```



```

11174 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11175 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11176 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11177 \fp_mul_product:NN #1 \l_fp_mul_b_vi_int
11178 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11179 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11180 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11181 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11182 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11183 \fp_mul_product:NN \l_fp_mul_a_vi_int #4
11184 \fp_mul_end_level:
11185 \fp_mul_product:NN #1 \l_fp_mul_b_v_int
11186 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11187 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11188 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11189 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11190 \fp_mul_product:NN \l_fp_mul_a_v_int #4
11191 \fp_mul_end_level:
11192 \fp_mul_product:NN #1 \l_fp_mul_b_iv_int
11193 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11194 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11195 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11196 \fp_mul_product:NN \l_fp_mul_a_iv_int #4
11197 \fp_mul_end_level:
11198 #9 0 \l_fp_mul_output_tl \scan_stop:
11199 \tl_clear:N \l_fp_mul_output_tl
11200 \fp_mul_product:NN #1 \l_fp_mul_b_iii_int
11201 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11202 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11203 \fp_mul_product:NN \l_fp_mul_a_iii_int #4
11204 \fp_mul_end_level:
11205 \fp_mul_product:NN #1 \l_fp_mul_b_ii_int
11206 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11207 \fp_mul_product:NN \l_fp_mul_a_ii_int #4
11208 \fp_mul_end_level:
11209 \fp_mul_product:NN #1 \l_fp_mul_b_i_int
11210 \fp_mul_product:NN \l_fp_mul_a_i_int #4
11211 \fp_mul_end_level:
11212 #8 0 \l_fp_mul_output_tl \scan_stop:
11213 \tl_clear:N \l_fp_mul_output_tl
11214 \fp_mul_product:NN #1 #4
11215 \fp_mul_end_level:
11216 #7 0 \l_fp_mul_output_tl \scan_stop:
11217 }

```

(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.

```

11218 \cs_new_protected_nopar:Npn \fp_div_integer:NNNNN #1#2#3#4#5
11219 {
11220   \l_fp_tmp_int #1
11221   \tex_divide:D \l_fp_tmp_int #3
11222   \l_fp_tmp_int \int_eval:w #1 - \l_fp_tmp_int * #3 \int_eval_end:
11223   #4 #1
11224   \tex_divide:D #4 #3
11225   #5 #2
11226   \tex_divide:D #5 #3
11227   \tex_multiply:D \l_fp_tmp_int \c_one_thousand
11228   \tex_divide:D \l_fp_tmp_int #3
11229   #5 \int_eval:w #5 + \l_fp_tmp_int * \c_one_million \int_eval_end:
11230   \if_int_compare:w #5 > \c_one_thousand_million
11231     \tex_advance:D #4 \c_one
11232     \tex_advance:D #5 -\c_one_thousand_million
11233   \fi:
11234 }

```

(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
11235 \cs_new_protected_nopar:Npn \fp_extended_normalise:
11236 {
11237   \fp_extended_normalise_aux_i:
11238   \fp_extended_normalise_aux_ii:
11239 }
11240 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:
11241 {
11242   \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
11243     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11244     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11245     \exp_after:wN \fp_extended_normalise_aux_i:w
11246     \int_use:N \l_fp_input_a_decimal_int \q_stop
11247     \exp_after:wN \fp_extended_normalise_aux_i:
11248   \fi:
11249 }
11250 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:w
11251 #1#2#3#4#5#6#7#8#9 \q_stop
11252 {
11253   \l_fp_input_a_integer_int
11254   \int_eval:w \l_fp_input_a_integer_int + #2 \scan_stop:
11255   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11256   \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11257   \exp_after:wN \fp_extended_normalise_aux_ii:w
11258   \int_use:N \l_fp_input_a_extended_int \q_stop
11259 }

```

```

11260 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:w
11261   #1#2#3#4#5#6#7#8#9 \q_stop
11262   {
11263     \l_fp_input_a_decimal_int
11264     \int_eval:w \l_fp_input_a_decimal_int + #2 \scan_stop:
11265     \l_fp_input_a_extended_int #3#4#5#6#7#8#9 0 \scan_stop:
11266     \tex_advance:D \l_fp_input_a_exponent_int \c_minus_one
11267   }
11268 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:
11269   {
11270     \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
11271       \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11272       \exp_after:wN \use_i:nn \exp_after:wN
11273       \fp_extended_normalise_ii_aux:NNNNNNNNN
11274       \int_use:N \l_fp_input_a_decimal_int
11275       \exp_after:wN \fp_extended_normalise_aux_ii:
11276     \fi:
11277   }
11278 \cs_new_protected_nopar:Npn \fp_extended_normalise_ii_aux:NNNNNNNNN
11279   #1#2#3#4#5#6#7#8#9
11280   {
11281     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11282       \l_fp_input_a_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
11283     \else:
11284       \tl_set:Nx \l_fp_tmp_tl
11285       {
11286         \int_use:N \l_fp_input_a_integer_int
11287         #1#2#3#4#5#6#7#8
11288       }
11289       \l_fp_input_a_integer_int \c_zero
11290       \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
11291     \fi:
11292     \tex_divide:D \l_fp_input_a_extended_int \c_ten
11293     \tl_set:Nx \l_fp_tmp_tl
11294     {
11295       #9
11296       \int_use:N \l_fp_input_a_extended_int
11297     }
11298     \l_fp_input_a_extended_int \l_fp_tmp_tl \scan_stop:
11299     \tex_advance:D \l_fp_input_a_exponent_int \c_one
11300   }

```

(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
11301 \cs_new_protected_nopar:Npn \fp_extended_normalise_output:
11302   {
11303     \if_int_compare:w \l_fp_output_integer_int > \c_nine

```

```

11304     \tex_advance:D \l_fp_output_integer_int \c_one_thousand_million
11305     \exp_after:wN \use_i:nn \exp_after:wN
11306         \fp_extended_normalise_output_aux_i:NNNNNNNNN
11307         \int_use:N \l_fp_output_integer_int
11308     \exp_after:wN \fp_extended_normalise_output:
11309     \fi:
11310 }
11311 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux_i:NNNNNNNNN
11312 #1#2#3#4#5#6#7#8#9
11313 {
11314     \l_fp_output_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
11315     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11316     \tl_set:Nx \l_fp_tmp_tl
11317     {
11318         #9
11319         \exp_after:wN \use_none:n
11320         \int_use:N \l_fp_output_decimal_int
11321     }
11322     \exp_after:wN \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11323     \l_fp_tmp_tl
11324 }
11325 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11326 #1#2#3#4#5#6#7#8#9
11327 {
11328     \l_fp_output_decimal_int #1#2#3#4#5#6#7#8#9 \scan_stop:
11329     \fp_extended_normalise_output_aux:N
11330 }
11331 \cs_new_protected_nopar:Npn \fp_extended_normalise_output_aux:N #1
11332 {
11333     \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11334     \tex_divide:D \l_fp_output_extended_int \c_ten
11335     \tl_set:Nx \l_fp_tmp_tl
11336     {
11337         #1
11338         \exp_after:wN \use_none:n
11339         \int_use:N \l_fp_output_extended_int
11340     }
11341     \l_fp_output_extended_int \l_fp_tmp_tl \scan_stop:
11342     \tex_advance:D \l_fp_output_exponent_int \c_one
11343 }

```

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

197.11 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 $-\pi < x \leq \pi$.

```

11344 \cs_new_protected_nopar:Npn \fp_trig_normalise:
11345 {

```

```

11346 \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11347 \l_fp_input_a_extended_int \c_zero
11348 \fp_extended_normalise:
11349 \fp_trig_normalise_aux:
11350 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11351 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11352 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11353 \fi:
11354 \exp_after:wN \fp_trig_octant:
11355 \else:
11356 \l_fp_input_a_sign_int \c_one
11357 \l_fp_output_integer_int \c_zero
11358 \l_fp_output_decimal_int \c_zero
11359 \l_fp_output_exponent_int \c_zero
11360 \exp_after:wN \fp_trig_overflow_msg:
11361 \fi:
11362 }
11363 \cs_new_protected_nopar:Npn \fp_trig_normalise_aux:
11364 {
11365 \if_int_compare:w \l_fp_input_a_integer_int > \c_three
11366 \fp_trig_sub:NNN
11367 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11368 \exp_after:wN \fp_trig_normalise_aux:
11369 \else:
11370 \if_int_compare:w \l_fp_input_a_integer_int > \c_two
11371 \if_int_compare:w \l_fp_input_a_decimal_int > \c_fp_pi_decimal_int
11372 \fp_trig_sub:NNN
11373 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11374 \exp_after:wN \exp_after:wN \exp_after:wN
11375 \exp_after:wN \exp_after:wN \exp_after:wN
11376 \exp_after:wN \fp_trig_normalise_aux:
11377 \fi:
11378 \fi:
11379 \fi:
11380 }

```

Here, there may be a sign change but there will never be any variation in the input. So a dedicated function can be used.

```

11381 \cs_new_protected_nopar:Npn \fp_trig_sub:NNN #1#2#3
11382 {
11383 \l_fp_input_a_integer_int
11384 \int_eval:w \l_fp_input_a_integer_int - #1 \int_eval_end:
11385 \l_fp_input_a_decimal_int
11386 \int_eval:w \l_fp_input_a_decimal_int - #2 \int_eval_end:
11387 \l_fp_input_a_extended_int
11388 \int_eval:w \l_fp_input_a_extended_int - #3 \int_eval_end:
11389 \if_int_compare:w \l_fp_input_a_extended_int < \c_zero
11390 \tex_advance:D \l_fp_input_a_decimal_int \c_minus_one
11391 \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11392 \fi:

```

```

11393 \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11394 \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11395 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11396 \fi:
11397 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11398 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11399 \if_int_compare:w
11400 \int_eval:w
11401 \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
11402 = \c_zero
11403 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11404 \else:
11405 \l_fp_input_a_integer_int
11406 \int_eval:w
11407 - \l_fp_input_a_integer_int - \c_one
11408 \int_eval_end:
11409 \l_fp_input_a_decimal_int
11410 \int_eval:w
11411 \c_one_thousand_million - \l_fp_input_a_decimal_int
11412 \int_eval_end:
11413 \l_fp_input_a_extended_int
11414 \int_eval:w
11415 \c_one_thousand_million - \l_fp_input_a_extended_int
11416 \int_eval_end:
11417 \fi:
11418 \fi:
11419 }

```

(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!

```

11420 \cs_new_protected_nopar:Npn \fp_trig_octant:
11421 {
11422 \l_fp_trig_octant_int \c_one
11423 \fp_trig_octant_aux_i:
11424 \if_int_compare:w \l_fp_input_a_decimal_int < \c_ten
11425 \l_fp_input_a_decimal_int \c_zero
11426 \l_fp_input_a_extended_int \c_zero
11427 \fi:
11428 \if_int_odd:w \l_fp_trig_octant_int
11429 \else:
11430 \fp_sub:NNNNNNNNN
11431 \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11432 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11433 \l_fp_input_a_extended_int
11434 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int

```

```

11435         \l_fp_input_a_extended_int
11436     \fi:
11437 }
11438 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_i:
11439 {
11440     \if_int_compare:w \l_fp_trig_octant_int > \c_four
11441         \l_fp_trig_octant_int \c_four
11442         \l_fp_input_a_decimal_int \c_fp_pi_by_four_decimal_int
11443         \l_fp_input_a_extended_int \c_fp_pi_by_four_extended_int
11444     \else:
11445         \exp_after:wN \fp_trig_octant_aux_ii:
11446     \fi:
11447 }
11448 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_ii:
11449 {
11450     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
11451         \fp_sub:NNNNNNNNN
11452         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11453         \l_fp_input_a_extended_int
11454         \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11455         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11456         \l_fp_input_a_extended_int
11457         \tex_advance:D \l_fp_trig_octant_int \c_one
11458         \exp_after:wN \fp_trig_octant_aux_i:
11459     \else:
11460         \if_int_compare:w
11461             \l_fp_input_a_decimal_int > \c_fp_pi_by_four_decimal_int
11462         \fp_sub:NNNNNNNNN
11463             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11464             \l_fp_input_a_extended_int
11465             \c_zero \c_fp_pi_by_four_decimal_int
11466             \c_fp_pi_by_four_extended_int
11467             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11468             \l_fp_input_a_extended_int
11469             \tex_advance:D \l_fp_trig_octant_int \c_one
11470         \exp_after:wN \exp_after:wN \exp_after:wN
11471         \fp_trig_octant_aux_i:
11472     \fi:
11473 \fi:
11474 }

```

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

<p>\fp_sin:Nn</p> <p>\fp_sin:cn</p> <p>\fp_gsin:Nn</p> <p>\fp_gsin:cn</p> <p>\fp_sin_aux:NNn</p> <p>\fp_sin_aux_i:</p> <p>\fp_sin_aux_ii:</p>	<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> <p>11475 \cs_new_protected_nopar:Npn \fp_sin:Nn { \fp_sin_aux:NNn \tl_set:Nn }</p> <p>11476 \cs_new_protected_nopar:Npn \fp_gsin:Nn { \fp_sin_aux:NNn \tl_gset:Nn }</p> <p>11477 \cs_generate_variant:Nn \fp_sin:Nn { c }</p> <p>11478 \cs_generate_variant:Nn \fp_gsin:Nn { c }</p>
---	---

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} .

```

11479 \cs_new_protected_nopar:Npn \fp_sin_aux:NNn #1#2#3
11480 {
11481   \group_begin:
11482   \fp_split:Nn a {#3}
11483   \fp_standardise:NNNN
11484   \l_fp_input_a_sign_int
11485   \l_fp_input_a_integer_int
11486   \l_fp_input_a_decimal_int
11487   \l_fp_input_a_exponent_int
11488   \tl_set:Nx \l_fp_arg_tl
11489   {
11490     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11491     -
11492     \else:
11493     +
11494     \fi:
11495     \int_use:N \l_fp_input_a_integer_int
11496     .
11497     \exp_after:wN \use_none:n
11498     \int_value:w \int_eval:w
11499     \l_fp_input_a_decimal_int + \c_one_thousand_million
11500     e
11501     \int_use:N \l_fp_input_a_exponent_int
11502   }
11503   \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
11504   \cs_set_protected_nopar:Npx \fp_tmp:w
11505   {
11506     \group_end:
11507     #1 \exp_not:N #2 { \l_fp_arg_tl }
11508   }
11509   \else:
11510     \if_cs_exist:w
11511     c_fp_sin ( \l_fp_arg_tl ) _fp
11512     \cs_end:
11513     \else:
11514       \exp_after:wN \exp_after:wN \exp_after:wN
11515       \fp_sin_aux_i:
11516     \fi:
11517     \cs_set_protected_nopar:Npx \fp_tmp:w
11518     {
11519       \group_end:
11520       #1 \exp_not:N #2
11521       { \use:c { c_fp_sin ( \l_fp_arg_tl ) _fp } }
11522     }
11523   \fi:
11524   \fp_tmp:w

```



```
11525 }
```

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.

```
11526 \cs_new_protected_nopar:Npn \fp_sin_aux_i:
11527 {
11528   \fp_trig_normalise:
11529   \fp_sin_aux_ii:
11530   \if_int_compare:w \l_fp_output_integer_int = \c_one
11531     \l_fp_output_exponent_int \c_zero
11532   \else:
11533     \l_fp_output_integer_int \l_fp_output_decimal_int
11534     \l_fp_output_decimal_int \l_fp_output_extended_int
11535     \l_fp_output_exponent_int -\c_nine
11536   \fi:
11537   \fp_standardise:NNNN
11538   \l_fp_input_a_sign_int
11539   \l_fp_output_integer_int
11540   \l_fp_output_decimal_int
11541   \l_fp_output_exponent_int
11542   \tl_new:c { c_fp_sin ( \l_fp_arg_tl ) _fp }
11543   \tl_gset:cx { c_fp_sin ( \l_fp_arg_tl ) _fp }
11544   {
11545     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11546       +
11547     \else:
11548       -
11549     \fi:
11550     \int_use:N \l_fp_output_integer_int
11551     .
11552     \exp_after:wN \use_none:n
11553     \int_value:w \int_eval:w
11554     \l_fp_output_decimal_int + \c_one_thousand_million
11555     e
11556     \int_use:N \l_fp_output_exponent_int
11557   }
11558 }
11559 \cs_new_protected_nopar:Npn \fp_sin_aux_ii:
11560 {
11561   \if_case:w \l_fp_trig_octant_int
11562   \or:
11563     \exp_after:wN \fp_trig_calc_sin:
11564   \or:
11565     \exp_after:wN \fp_trig_calc_cos:
11566   \or:
11567     \exp_after:wN \fp_trig_calc_cos:
11568   \or:
11569     \exp_after:wN \fp_trig_calc_sin:
11570   \fi:
```

```

11571 }
      (End definition for \fp_sin:Nn and \fp_sin:cn. These functions are documented on page ??.)

\fp_cos:Nn Cosine is almost identical, but there is no short cut code here.
\fp_cos:cn
\fp_gcos:Nn 11572 \cs_new_protected_nopar:Npn \fp_cos:Nn { \fp_cos_aux:NNn \tl_set:Nn }
\fp_gcos:cn 11573 \cs_new_protected_nopar:Npn \fp_gcos:Nn { \fp_cos_aux:NNn \tl_gset:Nn }
\fp_gcos:cn 11574 \cs_generate_variant:Nn \fp_cos:Nn { c }
\fp_cos_aux:NNn 11575 \cs_generate_variant:Nn \fp_gcos:Nn { c }
\fp_cos_aux_i: 11576 \cs_new_protected_nopar:Npn \fp_cos_aux:NNn #1#2#3
\fp_cos_aux_ii: 11577 {
11578   \group_begin:
11579   \fp_split:Nn a {#3}
11580   \fp_standardise:NNNN
11581   \l_fp_input_a_sign_int
11582   \l_fp_input_a_integer_int
11583   \l_fp_input_a_decimal_int
11584   \l_fp_input_a_exponent_int
11585   \tl_set:Nx \l_fp_arg_tl
11586   {
11587     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11588     -
11589     \else:
11590     +
11591     \fi:
11592     \int_use:N \l_fp_input_a_integer_int
11593     .
11594     \exp_after:wN \use_none:n
11595     \int_value:w \int_eval:w
11596     \l_fp_input_a_decimal_int + \c_one_thousand_million
11597     e
11598     \int_use:N \l_fp_input_a_exponent_int
11599   }
11600   \if_cs_exist:w c_fp_cos ( \l_fp_arg_tl ) _fp \cs_end:
11601   \else:
11602     \exp_after:wN \fp_cos_aux_i:
11603   \fi:
11604   \cs_set_protected_nopar:Npx \fp_tmp:w
11605   {
11606     \group_end:
11607     #1 \exp_not:N #2
11608     { \use:c { c_fp_cos ( \l_fp_arg_tl ) _fp } }
11609   }
11610   \fp_tmp:w
11611 }

```

Almost the same as for sine: just a bit of correction for the sign of the output.

```

11612 \cs_new_protected_nopar:Npn \fp_cos_aux_i:
11613 {
11614   \fp_trig_normalise:
11615   \fp_cos_aux_ii:

```

```

11616 \if_int_compare:w \l_fp_output_integer_int = \c_one
11617 \l_fp_output_exponent_int \c_zero
11618 \else:
11619 \l_fp_output_integer_int \l_fp_output_decimal_int
11620 \l_fp_output_decimal_int \l_fp_output_extended_int
11621 \l_fp_output_exponent_int -\c_nine
11622 \fi:
11623 \fp_standardise:NNNN
11624 \l_fp_input_a_sign_int
11625 \l_fp_output_integer_int
11626 \l_fp_output_decimal_int
11627 \l_fp_output_exponent_int
11628 \tl_new:c { c_fp_cos ( \l_fp_arg_tl ) _fp }
11629 \tl_gset:cx { c_fp_cos ( \l_fp_arg_tl ) _fp }
11630 {
11631 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11632 +
11633 \else:
11634 -
11635 \fi:
11636 \int_use:N \l_fp_output_integer_int
11637 .
11638 \exp_after:wN \use_none:n
11639 \int_value:w \int_eval:w
11640 \l_fp_output_decimal_int + \c_one_thousand_million
11641 e
11642 \int_use:N \l_fp_output_exponent_int
11643 }
11644 }
11645 \cs_new_protected_nopar:Npn \fp_cos_aux_ii:
11646 {
11647 \if_case:w \l_fp_trig_octant_int
11648 \or:
11649 \exp_after:wN \fp_trig_calc_cos:
11650 \or:
11651 \exp_after:wN \fp_trig_calc_sin:
11652 \or:
11653 \exp_after:wN \fp_trig_calc_sin:
11654 \or:
11655 \exp_after:wN \fp_trig_calc_cos:
11656 \fi:
11657 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11658 \if_int_compare:w \l_fp_trig_octant_int > \c_two
11659 \l_fp_input_a_sign_int \c_minus_one
11660 \fi:
11661 \else:
11662 \if_int_compare:w \l_fp_trig_octant_int > \c_two
11663 \else:
11664 \l_fp_input_a_sign_int \c_one
11665 \fi:

```

```

11666 \fi:
11667 }

```

(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: 11668 \cs_new_protected_nopar:Npn \fp_trig_calc_cos:
\fp_trig_calc_Taylor: 11669 {
11670 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11671 \l_fp_output_integer_int \c_one
11672 \l_fp_output_decimal_int \c_zero
11673 \else:
11674 \l_fp_trig_sign_int \c_minus_one
11675 \fp_mul:NNNNNN
11676 \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11677 \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11678 \l_fp_trig_decimal_int \l_fp_trig_extended_int
11679 \fp_div_integer:NNNNN
11680 \l_fp_trig_decimal_int \l_fp_trig_extended_int
11681 \c_two
11682 \l_fp_trig_decimal_int \l_fp_trig_extended_int
11683 \l_fp_count_int \c_three
11684 \if_int_compare:w \l_fp_trig_extended_int = \c_zero
11685 \if_int_compare:w \l_fp_trig_decimal_int = \c_zero
11686 \l_fp_output_integer_int \c_one
11687 \l_fp_output_decimal_int \c_zero
11688 \l_fp_output_extended_int \c_zero
11689 \else:
11690 \l_fp_output_integer_int \c_zero
11691 \l_fp_output_decimal_int \c_one_thousand_million
11692 \l_fp_output_extended_int \c_zero
11693 \fi:
11694 \else:
11695 \l_fp_output_integer_int \c_zero
11696 \l_fp_output_decimal_int 999999999 \scan_stop:
11697 \l_fp_output_extended_int \c_one_thousand_million
11698 \fi:
11699 \tex_advance:D \l_fp_output_extended_int -\l_fp_trig_extended_int
11700 \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
11701 \exp_after:wN \fp_trig_calc_Taylor:
11702 \fi:
11703 }
11704 \cs_new_protected_nopar:Npn \fp_trig_calc_sin:
11705 {
11706 \l_fp_output_integer_int \c_zero
11707 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11708 \l_fp_output_decimal_int \c_zero
11709 \else:
11710 \l_fp_output_decimal_int \l_fp_input_a_decimal_int
11711 \l_fp_output_extended_int \l_fp_input_a_extended_int
11712 \l_fp_trig_sign_int \c_one

```

```

11713     \l_fp_trig_decimal_int \l_fp_input_a_decimal_int
11714     \l_fp_trig_extended_int \l_fp_input_a_extended_int
11715     \l_fp_count_int \c_two
11716     \exp_after:wN \fp_trig_calc_Taylor:
11717   \fi:
11718 }

```

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.

```

11719 \cs_new_protected_nopar:Npn \fp_trig_calc_Taylor:
11720 {
11721   \l_fp_trig_sign_int -\l_fp_trig_sign_int
11722   \fp_mul:NNNNNN
11723   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11724   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11725   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11726   \fp_mul:NNNNNN
11727   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11728   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11729   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11730   \fp_div_integer:NNNNN
11731   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11732   \l_fp_count_int
11733   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11734   \tex_advance:D \l_fp_count_int \c_one
11735   \fp_div_integer:NNNNN
11736   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11737   \l_fp_count_int
11738   \l_fp_trig_decimal_int \l_fp_trig_extended_int
11739   \tex_advance:D \l_fp_count_int \c_one
11740   \if_int_compare:w \l_fp_trig_decimal_int > \c_zero
11741     \if_int_compare:w \l_fp_trig_sign_int > \c_zero
11742       \tex_advance:D \l_fp_output_decimal_int \l_fp_trig_decimal_int
11743       \tex_advance:D \l_fp_output_extended_int
11744         \l_fp_trig_extended_int
11745       \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
11746         \else:
11747           \tex_advance:D \l_fp_output_decimal_int \c_one
11748           \tex_advance:D \l_fp_output_extended_int
11749             -\c_one_thousand_million
11750         \fi:
11751       \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
11752         \else:
11753           \tex_advance:D \l_fp_output_integer_int \c_one
11754           \tex_advance:D \l_fp_output_decimal_int
11755             -\c_one_thousand_million
11756         \fi:
11757       \else:
11758         \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
11759         \tex_advance:D \l_fp_output_extended_int

```

```

11760         -\l_fp_input_a_extended_int
11761         \if_int_compare:w \l_fp_output_extended_int < \c_zero
11762             \tex_advance:D \l_fp_output_decimal_int \c_minus_one
11763             \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11764         \fi:
11765         \if_int_compare:w \l_fp_output_decimal_int < \c_zero
11766             \tex_advance:D \l_fp_output_integer_int \c_minus_one
11767             \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11768         \fi:
11769     \fi:
11770     \exp_after:wN \fp_trig_calc_Taylor:
11771 \fi:
11772 }

```

(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:
11773 \cs_new_protected_nopar:Npn \fp_tan:Nn { \fp_tan_aux:NNn \tl_set:Nn }
11774 \cs_new_protected_nopar:Npn \fp_gtan:Nn { \fp_tan_aux:NNn \tl_gset:Nn }
11775 \cs_generate_variant:Nn \fp_tan:Nn { c }
11776 \cs_generate_variant:Nn \fp_gtan:Nn { c }
11777 \cs_new_protected_nopar:Npn \fp_tan_aux:NNn #1#2#3
11778 {
11779     \group_begin:
11780     \fp_split:Nn a {#3}
11781     \fp_standardise:NNNN
11782     \l_fp_input_a_sign_int
11783     \l_fp_input_a_integer_int
11784     \l_fp_input_a_decimal_int
11785     \l_fp_input_a_exponent_int
11786     \tl_set:Nx \l_fp_arg_tl
11787     {
11788         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11789             -
11790         \else:
11791             +
11792         \fi:
11793         \int_use:N \l_fp_input_a_integer_int
11794         .
11795         \exp_after:wN \use_none:n
11796         \int_value:w \int_eval:w
11797         \l_fp_input_a_decimal_int + \c_one_thousand_million
11798         e
11799         \int_use:N \l_fp_input_a_exponent_int
11800     }
11801     \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
11802         \cs_set_protected_nopar:Npx \fp_tmp:w

```

```

11803     {
11804         \group_end:
11805         #1 \exp_not:N #2 { \l_fp_arg_tl }
11806     }
11807 \else:
11808     \if_cs_exist:w
11809         c_fp_tan ( \l_fp_arg_tl ) _fp
11810     \cs_end:
11811 \else:
11812     \exp_after:wN \exp_after:wN \exp_after:wN
11813         \fp_tan_aux_i:
11814 \fi:
11815 \cs_set_protected_nopar:Npx \fp_tmp:w
11816     {
11817         \group_end:
11818         #1 \exp_not:N #2
11819         { \use:c { c_fp_tan ( \l_fp_arg_tl ) _fp } }
11820     }
11821 \fi:
11822 \fp_tmp:w
11823 }

```

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.

```

11824 \cs_new_protected_nopar:Npn \fp_tan_aux_i:
11825 {
11826     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11827         \exp_after:wN \fp_tan_aux_ii:
11828     \else:
11829         \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11830         \c_zero_fp
11831         \exp_after:wN \fp_trig_overflow_msg:
11832     \fi:
11833 }
11834 \cs_new_protected_nopar:Npn \fp_tan_aux_ii:
11835 {
11836     \fp_trig_normalise:
11837     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11838         \if_int_compare:w \l_fp_trig_octant_int > \c_two
11839             \l_fp_output_sign_int \c_minus_one
11840         \else:
11841             \l_fp_output_sign_int \c_one
11842         \fi:
11843     \else:
11844         \if_int_compare:w \l_fp_trig_octant_int > \c_two
11845             \l_fp_output_sign_int \c_one
11846         \else:
11847             \l_fp_output_sign_int \c_minus_one

```

```

11848     \fi:
11849   \fi:
11850   \fp_cos_aux_ii:
11851   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11852     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11853       \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11854       \c_undefined_fp
11855     \else:
11856       \exp_after:wN \exp_after:wN \exp_after:wN
11857       \fp_tan_aux_iii:
11858     \fi:
11859   \else:
11860     \exp_after:wN \fp_tan_aux_iii:
11861   \fi:
11862 }

```

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.

```

11863 \cs_new_protected_nopar:Npn \fp_tan_aux_iii:
11864 {
11865   \l_fp_input_b_integer_int \l_fp_output_decimal_int
11866   \l_fp_input_b_decimal_int \l_fp_output_extended_int
11867   \l_fp_input_b_exponent_int -\c_nine
11868   \fp_standardise:NNNN
11869   \l_fp_input_b_sign_int
11870   \l_fp_input_b_integer_int
11871   \l_fp_input_b_decimal_int
11872   \l_fp_input_b_exponent_int
11873   \fp_sin_aux_ii:
11874   \l_fp_input_a_integer_int \l_fp_output_decimal_int
11875   \l_fp_input_a_decimal_int \l_fp_output_extended_int
11876   \l_fp_input_a_exponent_int -\c_nine
11877   \fp_standardise:NNNN
11878   \l_fp_input_a_sign_int
11879   \l_fp_input_a_integer_int
11880   \l_fp_input_a_decimal_int
11881   \l_fp_input_a_exponent_int
11882   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11883     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11884       \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
11885       \c_zero_fp
11886     \else:
11887       \exp_after:wN \exp_after:wN \exp_after:wN \fp_tan_aux_iv:
11888     \fi:
11889   \else:
11890     \exp_after:wN \fp_tan_aux_iv:
11891   \fi:
11892 }
11893 \cs_new_protected_nopar:Npn \fp_tan_aux_iv:
11894 {

```



```

11895 \l_fp_output_integer_int \c_zero
11896 \l_fp_output_decimal_int \c_zero
11897 \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
11898 \l_fp_div_offset_int \c_one_hundred_million
11899 \fp_div_loop:
11900 \l_fp_output_exponent_int
11901 \int_eval:w
11902 \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
11903 \int_eval_end:
11904 \fp_standardise:NNNN
11905 \l_fp_output_sign_int
11906 \l_fp_output_integer_int
11907 \l_fp_output_decimal_int
11908 \l_fp_output_exponent_int
11909 \tl_new:c { c_fp_tan ( \l_fp_arg_tl ) _fp }
11910 \tl_gset:cx { c_fp_tan ( \l_fp_arg_tl ) _fp }
11911 {
11912 \if_int_compare:w \l_fp_output_sign_int > \c_zero
11913 +
11914 \else:
11915 -
11916 \fi:
11917 \int_use:N \l_fp_output_integer_int
11918 .
11919 \exp_after:wN \use_none:n
11920 \int_value:w \int_eval:w
11921 \l_fp_output_decimal_int + \c_one_thousand_million
11922 e
11923 \int_use:N \l_fp_output_exponent_int
11924 }
11925 }

```

(End definition for `\fp_tan:Nn` and `\fp_tan:cn`. These functions are documented on page ??.)

197.12 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>	11926	<code>\tl_const:cn { c_fp_exp_1_tl }</code>	{ { 2 } { 718281828 } { 459045235 } { 0 } }
<code>\c_fp_exp_3_tl</code>	11927	<code>\tl_const:cn { c_fp_exp_2_tl }</code>	{ { 7 } { 389056098 } { 930650227 } { 0 } }
<code>\c_fp_exp_4_tl</code>	11928	<code>\tl_const:cn { c_fp_exp_3_tl }</code>	{ { 2 } { 008553692 } { 318766774 } { 1 } }
<code>\c_fp_exp_5_tl</code>	11929	<code>\tl_const:cn { c_fp_exp_4_tl }</code>	{ { 5 } { 459815003 } { 314423908 } { 1 } }
<code>\c_fp_exp_6_tl</code>	11930	<code>\tl_const:cn { c_fp_exp_5_tl }</code>	{ { 1 } { 484131591 } { 025766034 } { 2 } }
<code>\c_fp_exp_7_tl</code>	11931	<code>\tl_const:cn { c_fp_exp_6_tl }</code>	{ { 4 } { 034287934 } { 927351226 } { 2 } }
<code>\c_fp_exp_8_tl</code>	11932	<code>\tl_const:cn { c_fp_exp_7_tl }</code>	{ { 1 } { 096633158 } { 428458599 } { 3 } }
<code>\c_fp_exp_9_tl</code>	11933	<code>\tl_const:cn { c_fp_exp_8_tl }</code>	{ { 2 } { 980957987 } { 041728275 } { 3 } }
<code>\c_fp_exp_10_tl</code>	11934	<code>\tl_const:cn { c_fp_exp_9_tl }</code>	{ { 8 } { 103083927 } { 575384008 } { 3 } }
<code>\c_fp_exp_20_tl</code>	11935	<code>\tl_const:cn { c_fp_exp_10_tl }</code>	{ { 2 } { 202646579 } { 480671652 } { 4 } }
<code>\c_fp_exp_30_tl</code>	11936	<code>\tl_const:cn { c_fp_exp_20_tl }</code>	{ { 4 } { 851651954 } { 097902280 } { 8 } }
<code>\c_fp_exp_40_tl</code>	11937	<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>			

```

11938 \tl_const:cn { c_fp_exp_40_tl } { { 2 } { 353852668 } { 370199854 } { 17 } }
11939 \tl_const:cn { c_fp_exp_50_tl } { { 5 } { 184705528 } { 587072464 } { 21 } }
11940 \tl_const:cn { c_fp_exp_60_tl } { { 1 } { 142007389 } { 815684284 } { 26 } }
11941 \tl_const:cn { c_fp_exp_70_tl } { { 2 } { 515438670 } { 919167006 } { 30 } }
11942 \tl_const:cn { c_fp_exp_80_tl } { { 5 } { 540622384 } { 393510053 } { 34 } }
11943 \tl_const:cn { c_fp_exp_90_tl } { { 1 } { 220403294 } { 317840802 } { 39 } }
11944 \tl_const:cn { c_fp_exp_100_tl } { { 2 } { 688117141 } { 816135448 } { 43 } }
11945 \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 11946 \tl_const:cn { c_fp_exp_-1_tl } { { 3 } { 678794411 } { 71442322 } { -1 } }
\c_fp_exp_-3_tl 11947 \tl_const:cn { c_fp_exp_-2_tl } { { 1 } { 353352832 } { 366132692 } { -1 } }
\c_fp_exp_-4_tl 11948 \tl_const:cn { c_fp_exp_-3_tl } { { 4 } { 978706836 } { 786394298 } { -2 } }
\c_fp_exp_-5_tl 11949 \tl_const:cn { c_fp_exp_-4_tl } { { 1 } { 831563888 } { 873418029 } { -2 } }
\c_fp_exp_-6_tl 11950 \tl_const:cn { c_fp_exp_-5_tl } { { 6 } { 737946999 } { 085467097 } { -3 } }
\c_fp_exp_-7_tl 11951 \tl_const:cn { c_fp_exp_-6_tl } { { 2 } { 478752176 } { 666358423 } { -3 } }
\c_fp_exp_-8_tl 11952 \tl_const:cn { c_fp_exp_-7_tl } { { 9 } { 118819655 } { 545162080 } { -4 } }
\c_fp_exp_-9_tl 11953 \tl_const:cn { c_fp_exp_-8_tl } { { 3 } { 354626279 } { 025118388 } { -4 } }
\c_fp_exp_-10_tl 11954 \tl_const:cn { c_fp_exp_-9_tl } { { 1 } { 234098040 } { 866795495 } { -4 } }
\c_fp_exp_-20_tl 11955 \tl_const:cn { c_fp_exp_-10_tl } { { 4 } { 539992976 } { 248451536 } { -5 } }
\c_fp_exp_-30_tl 11956 \tl_const:cn { c_fp_exp_-20_tl } { { 2 } { 061153622 } { 438557828 } { -9 } }
\c_fp_exp_-40_tl 11957 \tl_const:cn { c_fp_exp_-30_tl } { { 9 } { 357622968 } { 840174605 } { -14 } }
\c_fp_exp_-50_tl 11958 \tl_const:cn { c_fp_exp_-40_tl } { { 4 } { 248354255 } { 291588995 } { -18 } }
\c_fp_exp_-60_tl 11959 \tl_const:cn { c_fp_exp_-50_tl } { { 1 } { 928749847 } { 963917783 } { -22 } }
\c_fp_exp_-70_tl 11960 \tl_const:cn { c_fp_exp_-60_tl } { { 8 } { 756510762 } { 696520338 } { -27 } }
\c_fp_exp_-80_tl 11961 \tl_const:cn { c_fp_exp_-70_tl } { { 3 } { 975449735 } { 908646808 } { -31 } }
\c_fp_exp_-90_tl 11962 \tl_const:cn { c_fp_exp_-80_tl } { { 1 } { 804851387 } { 845415172 } { -35 } }
\c_fp_exp_-100_tl 11963 \tl_const:cn { c_fp_exp_-90_tl } { { 8 } { 194012623 } { 990515430 } { -40 } }
\c_fp_exp_-200_tl 11964 \tl_const:cn { c_fp_exp_-100_tl } { { 3 } { 720075976 } { 020835963 } { -44 } }
11965 \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 11966 \cs_new_protected_nopar:Npn \fp_exp:Nn { \fp_exp_aux:NNn \tl_set:Nn }
\fp_exp_internal: 11967 \cs_new_protected_nopar:Npn \fp_gexp:Nn { \fp_exp_aux:NNn \tl_gset:Nn }
\fp_exp_aux: 11968 \cs_generate_variant:Nn \fp_exp:Nn { c }
\fp_exp_integer: 11969 \cs_generate_variant:Nn \fp_gexp:Nn { c }
\fp_exp_integer_tens: 11970 \cs_new_protected_nopar:Npn \fp_exp_aux:NNn #1#2#3
\fp_exp_integer_units: 11971 {
\fp_exp_integer_const:n 11972 \group_begin:
\fp_exp_integer_const:nnnn 11973 \fp_split:Nn a {#3}
11974 \fp_standardise:NNNN
\fp_exp_decimal: 11975 \l_fp_input_a_sign_int
\fp_exp_Taylor: 11976 \l_fp_input_a_integer_int
\fp_exp_const:Nx 11977 \l_fp_input_a_decimal_int
\fp_exp_const:cx

```

```

11978         \l_fp_input_a_exponent_int
11979     \l_fp_input_a_extended_int \c_zero
11980     \tl_set:Nx \l_fp_arg_tl
11981     {
11982         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11983         -
11984         \else:
11985         +
11986         \fi:
11987         \int_use:N \l_fp_input_a_integer_int
11988         .
11989         \exp_after:wN \use_none:n
11990         \int_value:w \int_eval:w
11991         \l_fp_input_a_decimal_int + \c_one_thousand_million
11992         e
11993         \int_use:N \l_fp_input_a_exponent_int
11994     }
11995     \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp \cs_end:
11996     \else:
11997         \exp_after:wN \fp_exp_internal:
11998     \fi:
11999     \cs_set_protected_nopar:Npx \fp_tmp:w
12000     {
12001         \group_end:
12002         #1 \exp_not:N #2
12003         {
12004             \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp
12005             \cs_end:
12006             \use:c { c_fp_exp ( \l_fp_arg_tl ) _fp }
12007         \else:
12008             \c_zero_fp
12009         \fi:
12010         }
12011     }
12012     \fp_tmp:w
12013 }

```

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

```

12014 \cs_new_protected_nopar:Npn \fp_exp_internal:
12015 {
12016     \if_int_compare:w \l_fp_input_a_exponent_int < \c_three
12017     \fp_extended_normalise:
12018     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12019     \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12020     \exp_after:wN \exp_after:wN \exp_after:wN
12021     \exp_after:wN \exp_after:wN \exp_after:wN
12022     \exp_after:wN \fp_exp_aux:
12023     \else:

```

```

12024         \exp_after:wN \exp_after:wN \exp_after:wN
12025         \exp_after:wN \exp_after:wN \exp_after:wN
12026         \exp_after:wN \fp_exp_overflow_msg:
12027         \fi:
12028     \else:
12029         \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12030         \exp_after:wN \exp_after:wN \exp_after:wN
12031         \exp_after:wN \exp_after:wN \exp_after:wN
12032         \exp_after:wN \fp_exp_aux:
12033     \else:
12034         \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12035         { \c_zero_fp }
12036     \fi:
12037 \fi:
12038 \else:
12039     \exp_after:wN \fp_exp_overflow_msg:
12040 \fi:
12041 }

```

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.

```

12042 \cs_new_protected_nopar:Npn \fp_exp_aux:
12043 {
12044     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12045     \exp_after:wN \fp_exp_integer:
12046 \else:
12047     \l_fp_output_integer_int \c_one
12048     \l_fp_output_decimal_int \c_zero
12049     \l_fp_output_extended_int \c_zero
12050     \l_fp_output_exponent_int \c_zero
12051     \exp_after:wN \fp_exp_decimal:
12052 \fi:
12053 }

```

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).

```

12054 \cs_new_protected_nopar:Npn \fp_exp_integer:
12055 {
12056     \if_int_compare:w \l_fp_input_a_integer_int < \c_one_hundred
12057     \l_fp_exp_integer_int \c_one

```

```

12058     \l_fp_exp_decimal_int  \c_zero
12059     \l_fp_exp_extended_int \c_zero
12060     \l_fp_exp_exponent_int \c_zero
12061     \exp_after:wN \fp_exp_integer_tens:
12062 \else:
12063     \tl_set:Nx \l_fp_tmp_tl
12064     {
12065         \exp_after:wN \use_i:nnn
12066         \int_use:N \l_fp_input_a_integer_int
12067     }
12068     \l_fp_input_a_integer_int
12069     \int_eval:w
12070     \l_fp_input_a_integer_int - \l_fp_tmp_tl 00
12071     \int_eval_end:
12072     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12073     \if_int_compare:w \l_fp_output_integer_int > 200 \scan_stop:
12074         \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12075         { \c_zero_fp }
12076     \else:
12077         \fp_exp_integer_const:n { - \l_fp_tmp_tl 00 }
12078         \exp_after:wN \exp_after:wN \exp_after:wN
12079         \exp_after:wN \exp_after:wN \exp_after:wN
12080         \exp_after:wN \fp_exp_integer_tens:
12081     \fi:
12082 \else:
12083     \fp_exp_integer_const:n { \l_fp_tmp_tl 00 }
12084     \exp_after:wN \exp_after:wN \exp_after:wN
12085     \exp_after:wN \fp_exp_integer_tens:
12086 \fi:
12087 \fi:
12088 }

```

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.

```

12089 \cs_new_protected_nopar:Npn \fp_exp_integer_tens:
12090 {
12091     \l_fp_output_integer_int  \l_fp_exp_integer_int
12092     \l_fp_output_decimal_int  \l_fp_exp_decimal_int
12093     \l_fp_output_extended_int \l_fp_exp_extended_int
12094     \l_fp_output_exponent_int \l_fp_exp_exponent_int
12095     \if_int_compare:w \l_fp_input_a_integer_int > \c_nine
12096         \tl_set:Nx \l_fp_tmp_tl
12097         {
12098             \exp_after:wN \use_i:nn
12099             \int_use:N \l_fp_input_a_integer_int
12100         }
12101     \l_fp_input_a_integer_int
12102     \int_eval:w
12103     \l_fp_input_a_integer_int - \l_fp_tmp_tl 0

```

```

12104         \int_eval_end:
12105         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12106             \fp_exp_integer_const:n { \l_fp_tmp_tl 0 }
12107         \else:
12108             \fp_exp_integer_const:n { - \l_fp_tmp_tl 0 }
12109         \fi:
12110         \fp_mul:NNNNNNNNN
12111             \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12112             \l_fp_output_integer_int \l_fp_output_decimal_int
12113             \l_fp_output_extended_int
12114             \l_fp_output_integer_int \l_fp_output_decimal_int
12115             \l_fp_output_extended_int
12116         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12117         \fp_extended_normalise_output:
12118     \fi:
12119     \fp_exp_integer_units:
12120 }
12121 \cs_new_protected_nopar:Npn \fp_exp_integer_units:
12122 {
12123     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12124         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12125             \fp_exp_integer_const:n { \int_use:N \l_fp_input_a_integer_int }
12126         \else:
12127             \fp_exp_integer_const:n
12128             { - \int_use:N \l_fp_input_a_integer_int }
12129         \fi:
12130         \fp_mul:NNNNNNNNN
12131             \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12132             \l_fp_output_integer_int \l_fp_output_decimal_int
12133             \l_fp_output_extended_int
12134             \l_fp_output_integer_int \l_fp_output_decimal_int
12135             \l_fp_output_extended_int
12136         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12137         \fp_extended_normalise_output:
12138     \fi:
12139     \fp_exp_decimal:
12140 }

```

Recovery of the stored constant values into the separate registers is done with a simple expansion then assignment.

```

12141 \cs_new_protected_nopar:Npn \fp_exp_integer_const:n #1
12142 {
12143     \exp_after:wN \exp_after:wN \exp_after:wN
12144     \fp_exp_integer_const:nnnn
12145     \cs:w c_fp_exp_ #1 _tl \cs_end:
12146 }
12147 \cs_new_protected_nopar:Npn \fp_exp_integer_const:nnnn #1#2#3#4
12148 {
12149     \l_fp_exp_integer_int #1 \scan_stop:
12150     \l_fp_exp_decimal_int #2 \scan_stop:

```

```

12151     \l_fp_exp_extended_int #3 \scan_stop:
12152     \l_fp_exp_exponent_int #4 \scan_stop:
12153 }

```

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.

```

12154 \cs_new_protected_nopar:Npn \fp_exp_decimal:
12155 {
12156   \if_int_compare:w \l_fp_input_a_decimal_int > \c_zero
12157   \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12158     \l_fp_exp_integer_int \c_one
12159     \l_fp_exp_decimal_int \l_fp_input_a_decimal_int
12160     \l_fp_exp_extended_int \l_fp_input_a_extended_int
12161   \else:
12162     \l_fp_exp_integer_int \c_zero
12163     \if_int_compare:w \l_fp_exp_extended_int = \c_zero
12164       \l_fp_exp_decimal_int
12165       \int_eval:w
12166         \c_one_thousand_million - \l_fp_input_a_decimal_int
12167       \int_eval_end:
12168     \l_fp_exp_extended_int \c_zero
12169   \else:
12170     \l_fp_exp_decimal_int
12171     \int_eval:w
12172       999999999 - \l_fp_input_a_decimal_int
12173     \scan_stop:
12174     \l_fp_exp_extended_int
12175     \int_eval:w
12176       \c_one_thousand_million - \l_fp_input_a_extended_int
12177     \int_eval_end:
12178   \fi:
12179   \fi:
12180   \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12181   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12182   \l_fp_input_b_extended_int \l_fp_input_a_extended_int
12183   \l_fp_count_int \c_one
12184   \fp_exp_Taylor:
12185   \fp_mul:NNNNNNNNN
12186     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12187     \l_fp_output_integer_int \l_fp_output_decimal_int
12188     \l_fp_output_extended_int
12189     \l_fp_output_integer_int \l_fp_output_decimal_int
12190     \l_fp_output_extended_int
12191   \fi:
12192   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12193   \else:
12194     \tex_advance:D \l_fp_output_decimal_int \c_one
12195     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million

```

```

12196     \else:
12197         \l_fp_output_decimal_int \c_zero
12198         \tex_advance:D \l_fp_output_integer_int \c_one
12199     \fi:
12200 \fi:
12201 \fp_standardise:NNNN
12202     \l_fp_output_sign_int
12203     \l_fp_output_integer_int
12204     \l_fp_output_decimal_int
12205     \l_fp_output_exponent_int
12206 \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12207 {
12208     +
12209     \int_use:N \l_fp_output_integer_int
12210     .
12211     \exp_after:wN \use_none:n
12212     \int_value:w \int_eval:w
12213         \l_fp_output_decimal_int + \c_one_thousand_million
12214     e
12215     \int_use:N \l_fp_output_exponent_int
12216 }
12217 }

```

The Taylor series for $\exp(x)$ is

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

which converges for $-1 < x < 1$. The code above sets up the x part, leaving the loop to multiply the running value by x/n and add it onto the sum. The way that this is done is that the running total is stored in the `exp` set of registers, while the current item is stored as `input_b`.

```

12218 \cs_new_protected_nopar:Npn \fp_exp_Taylor:
12219 {
12220     \tex_advance:D \l_fp_count_int \c_one
12221     \tex_multiply:D \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12222     \fp_mul:NNNNNN
12223         \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12224         \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12225         \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12226     \fp_div_integer:NNNNN
12227         \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12228         \l_fp_count_int
12229         \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12230     \if_int_compare:w
12231         \int_eval:w
12232             \l_fp_input_b_decimal_int + \l_fp_input_b_extended_int
12233         > \c_zero
12234     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12235         \tex_advance:D \l_fp_exp_decimal_int \l_fp_input_b_decimal_int

```



```

12236 \tex_advance:D \l_fp_exp_extended_int
12237 \l_fp_input_b_extended_int
12238 \if_int_compare:w \l_fp_exp_extended_int < \c_one_thousand_million
12239 \else:
12240 \tex_advance:D \l_fp_exp_decimal_int \c_one
12241 \tex_advance:D \l_fp_exp_extended_int
12242 -\c_one_thousand_million
12243 \fi:
12244 \if_int_compare:w \l_fp_exp_decimal_int < \c_one_thousand_million
12245 \else:
12246 \tex_advance:D \l_fp_exp_integer_int \c_one
12247 \tex_advance:D \l_fp_exp_decimal_int
12248 -\c_one_thousand_million
12249 \fi:
12250 \else:
12251 \tex_advance:D \l_fp_exp_decimal_int -\l_fp_input_b_decimal_int
12252 \tex_advance:D \l_fp_exp_extended_int
12253 -\l_fp_input_a_extended_int
12254 \if_int_compare:w \l_fp_exp_extended_int < \c_zero
12255 \tex_advance:D \l_fp_exp_decimal_int \c_minus_one
12256 \tex_advance:D \l_fp_exp_extended_int \c_one_thousand_million
12257 \fi:
12258 \if_int_compare:w \l_fp_exp_decimal_int < \c_zero
12259 \tex_advance:D \l_fp_exp_integer_int \c_minus_one
12260 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12261 \fi:
12262 \fi:
12263 \exp_after:wN \fp_exp_Taylor:
12264 \fi:
12265 }

```

This is set up as a function so that the power code can redirect the effect.

```

12266 \cs_new_protected_nopar:Npn \fp_exp_const:Nx #1#2
12267 {
12268 \tl_new:N #1
12269 \tl_gset:Nx #1 {#2}
12270 }
12271 \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.

```

12272 \tl_const:cn { c_fp_ln_10_1_tl } { { 2 } { 302585092 } { 994045684 } { 0 } }
12273 \tl_const:cn { c_fp_ln_10_2_tl } { { 4 } { 605170185 } { 988091368 } { 0 } }
12274 \tl_const:cn { c_fp_ln_10_3_tl } { { 6 } { 907755278 } { 982137052 } { 0 } }
12275 \tl_const:cn { c_fp_ln_10_4_tl } { { 9 } { 210340371 } { 976182736 } { 0 } }
12276 \tl_const:cn { c_fp_ln_10_5_tl } { { 1 } { 151292546 } { 497022842 } { 1 } }
12277 \tl_const:cn { c_fp_ln_10_6_tl } { { 1 } { 381551055 } { 796427410 } { 1 } }
12278 \tl_const:cn { c_fp_ln_10_7_tl } { { 1 } { 611809565 } { 095831979 } { 1 } }
12279 \tl_const:cn { c_fp_ln_10_8_tl } { { 1 } { 842068074 } { 395226547 } { 1 } }
12280 \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 12281 \tl_const:cn { c_fp_ln_2_1_t1 } { { 0 } { 693147180 } { 559945309 } { 0 } }
\c_fp_ln_2_3_t1 12282 \tl_const:cn { c_fp_ln_2_2_t1 } { { 1 } { 386294361 } { 119890618 } { 0 } }
12283 \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 12284 \cs_new_protected_nopar:Npn \fp_ln:Nn { \fp_ln_aux:NNn \tl_set:Nn }
\fp_ln_aux:NNn 12285 \cs_new_protected_nopar:Npn \fp_gln:Nn { \fp_ln_aux:NNn \tl_gset:Nn }
\fp_ln_aux: 12286 \cs_generate_variant:Nn \fp_ln:Nn { c }
\fp_ln_exponent: 12287 \cs_generate_variant:Nn \fp_gln:Nn { c }
\fp_ln_internal: 12288 \cs_new_protected_nopar:Npn \fp_ln_aux:NNn #1#2#3
\fp_ln_exponent_tens: 12289 {
\fp_ln_exponent_units: 12290 \group_begin:
\fp_ln_normalise: 12291 \fp_split:Nn a {#3}
12292 \fp_standardise:NNNN
12293 \l_fp_input_a_sign_int
12294 \l_fp_input_a_integer_int
\fp_ln_mantissa: 12295 \l_fp_input_a_decimal_int
\fp_ln_mantissa_aux: 12296 \l_fp_input_a_exponent_int
\fp_ln_mantissa_divide_two: 12297 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
\fp_ln_integer_const:nn 12298 \if_int_compare:w
\fp_ln_Taylor: 12299 \int_eval:w
\fp_ln_fixed: 12300 \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
\fp_ln_fixed_aux:NNNNNNNNN 12301 > \c_zero
\fp_ln_Taylor_aux: 12302 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_aux:
12303 \else:
12304 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12305 {
12306 \group_end:
12307 ##1 \exp_not:N ##2 { \c_zero_fp }
12308 }
12309 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_error_msg:
12310 \fi:
12311 \else:
12312 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12313 {
12314 \group_end:
12315 ##1 \exp_not:N ##2 { \c_zero_fp }
12316 }
12317 \exp_after:wN \fp_ln_error_msg:
12318 \fi:
12319 \fp_tmp:w #1 #2
12320 }

```

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.

```

12321 \cs_new_protected_nopar:Npn \fp_ln_aux:
12322 {
12323   \tl_set:Nx \l_fp_arg_tl
12324   {
12325     +
12326     \int_use:N \l_fp_input_a_integer_int
12327     .
12328     \exp_after:wN \use_none:n
12329     \int_value:w \int_eval:w
12330     \l_fp_input_a_decimal_int + \c_one_thousand_million
12331     e
12332     \int_use:N \l_fp_input_a_exponent_int
12333   }
12334   \if_cs_exist:w c_fp_ln ( \l_fp_arg_tl ) _fp \cs_end:
12335   \else:
12336     \exp_after:wN \fp_ln_exponent:
12337     \fi:
12338     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12339     {
12340       \group_end:
12341       ##1 \exp_not:N ##2
12342       { \use:c { c_fp_ln ( \l_fp_arg_tl ) _fp } }
12343     }
12344   }

```

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.

```

12345 \cs_new_protected_nopar:Npn \fp_ln_exponent:
12346 {
12347   \fp_ln_internal:
12348   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12349   \else:
12350     \tex_advance:D \l_fp_output_decimal_int \c_one

```

```

12351 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12352 \else:
12353   \l_fp_output_decimal_int \c_zero
12354   \tex_advance:D \l_fp_output_integer_int \c_one
12355 \fi:
12356 \fi:
12357 \fp_standardise:NNNN
12358   \l_fp_output_sign_int
12359   \l_fp_output_integer_int
12360   \l_fp_output_decimal_int
12361   \l_fp_output_exponent_int
12362 \tl_const:cx { c_fp_ln ( \l_fp_arg_tl ) _fp }
12363 {
12364   \if_int_compare:w \l_fp_output_sign_int > \c_zero
12365     +
12366   \else:
12367     -
12368   \fi:
12369   \int_use:N \l_fp_output_integer_int
12370   .
12371   \exp_after:wN \use_none:n
12372   \int_value:w \int_eval:w
12373     \l_fp_output_decimal_int + \c_one_thousand_million
12374   \scan_stop:
12375   e
12376   \int_use:N \l_fp_output_exponent_int
12377 }
12378 }
12379 \cs_new_protected_nopar:Npn \fp_ln_internal:
12380 {
12381   \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
12382     \l_fp_input_a_exponent_int -\l_fp_input_a_exponent_int
12383     \l_fp_output_sign_int \c_minus_one
12384   \else:
12385     \l_fp_output_sign_int \c_one
12386   \fi:
12387   \if_int_compare:w \l_fp_input_a_exponent_int > \c_nine
12388     \exp_after:wN \fp_ln_exponent_tens:NN
12389     \int_use:N \l_fp_input_a_exponent_int
12390   \else:
12391     \l_fp_output_integer_int \c_zero
12392     \l_fp_output_decimal_int \c_zero
12393     \l_fp_output_extended_int \c_zero
12394     \l_fp_output_exponent_int \c_zero
12395   \fi:
12396   \fp_ln_exponent_units:
12397 }
12398 \cs_new_protected_nopar:Npn \fp_ln_exponent_tens:NN #1 #2
12399 {
12400   \l_fp_input_a_exponent_int #2 \scan_stop:

```

```

12401 \fp_ln_const:nn { 10 } { #1 }
12402 \tex_advance:D \l_fp_exp_exponent_int \c_one
12403 \l_fp_output_integer_int \l_fp_exp_integer_int
12404 \l_fp_output_decimal_int \l_fp_exp_decimal_int
12405 \l_fp_output_extended_int \l_fp_exp_extended_int
12406 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12407 }

```

Next the smaller powers of ten, which will need to be combined with the above: always an additive process.

```

12408 \cs_new_protected_nopar:Npn \fp_ln_exponent_units:
12409 {
12410 \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
12411 \fp_ln_const:nn { 10 } { \int_use:N \l_fp_input_a_exponent_int }
12412 \fp_ln_normalise:
12413 \fp_add:NNNNNNNNN
12414 \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12415 \l_fp_output_integer_int \l_fp_output_decimal_int
12416 \l_fp_output_extended_int
12417 \l_fp_output_integer_int \l_fp_output_decimal_int
12418 \l_fp_output_extended_int
12419 \fi:
12420 \fp_ln_mantissa:
12421 }

```

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.

```

12422 \cs_new_protected_nopar:Npn \fp_ln_normalise:
12423 {
12424 \if_int_compare:w \l_fp_exp_exponent_int < \l_fp_output_exponent_int
12425 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12426 \exp_after:wN \use_i:nn \exp_after:wN
12427 \fp_ln_normalise_aux:NNNNNNNNN
12428 \int_use:N \l_fp_exp_decimal_int
12429 \exp_after:wN \fp_ln_normalise:
12430 \else:
12431 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12432 \fi:
12433 }
12434 \cs_new_protected_nopar:Npn \fp_ln_normalise_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12435 {
12436 \if_int_compare:w \l_fp_exp_integer_int = \c_zero
12437 \l_fp_exp_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12438 \else:
12439 \tl_set:Nx \l_fp_tmp_tl
12440 {
12441 \int_use:N \l_fp_exp_integer_int
12442 #1#2#3#4#5#6#7#8

```

```

12443     }
12444     \l_fp_exp_integer_int \c_zero
12445     \l_fp_exp_decimal_int \l_fp_tmp_tl \scan_stop:
12446 \fi:
12447 \tex_divide:D \l_fp_exp_extended_int \c_ten
12448 \tl_set:Nx \l_fp_tmp_tl
12449 {
12450     #9
12451     \int_use:N \l_fp_exp_extended_int
12452 }
12453 \l_fp_exp_extended_int \l_fp_tmp_tl \scan_stop:
12454 \tex_advance:D \l_fp_exp_exponent_int \c_one
12455 }

```

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*.

```

12456 \cs_new_protected_nopar:Npn \fp_ln_mantissa:
12457 {
12458     \l_fp_count_int \c_zero
12459     \l_fp_input_a_extended_int \c_zero
12460     \fp_ln_mantissa_aux:
12461     \if_int_compare:w \l_fp_count_int > \c_zero
12462         \fp_ln_const:nn { 2 } { \int_use:N \l_fp_count_int }
12463         \fp_ln_normalise:
12464         \if_int_compare:w \l_fp_output_sign_int > \c_zero
12465             \exp_after:wN \fp_add:NNNNNNNNN
12466         \else:
12467             \exp_after:wN \fp_sub:NNNNNNNNN
12468         \fi:
12469         \l_fp_output_integer_int \l_fp_output_decimal_int
12470         \l_fp_output_extended_int
12471         \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12472         \l_fp_output_integer_int \l_fp_output_decimal_int
12473         \l_fp_output_extended_int
12474     \fi:
12475     \if_int_compare:w
12476         \int_eval:w
12477         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int > \c_one
12478         \exp_after:wN \fp_ln_Taylor:
12479     \fi:
12480 }
12481 \cs_new_protected_nopar:Npn \fp_ln_mantissa_aux:
12482 {
12483     \if_int_compare:w \l_fp_input_a_integer_int > \c_one
12484         \tex_advance:D \l_fp_count_int \c_one
12485         \fp_ln_mantissa_divide_two:
12486         \exp_after:wN \fp_ln_mantissa_aux:
12487     \fi:
12488 }

```

A fast one-shot division by two.

```

12489 \cs_new_protected_nopar:Npn \fp_ln_mantissa_divide_two:
12490 {
12491   \if_int_odd:w \l_fp_input_a_decimal_int
12492     \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
12493   \fi:
12494   \if_int_odd:w \l_fp_input_a_integer_int
12495     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
12496   \fi:
12497   \tex_divide:D \l_fp_input_a_integer_int \c_two
12498   \tex_divide:D \l_fp_input_a_decimal_int \c_two
12499   \tex_divide:D \l_fp_input_a_extended_int \c_two
12500 }

```

Recovering constants makes use of the same auxiliary code as for exponents.

```

12501 \cs_new_protected_nopar:Npn \fp_ln_const:nn #1#2
12502 {
12503   \exp_after:wN \exp_after:wN \exp_after:wN
12504   \fp_exp_integer_const:nnnn
12505   \cs:w c_fp_ln_ #1 _ #2 _t1 \cs_end:
12506 }

```

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”.

```

12507 \cs_new_protected_nopar:Npn \fp_ln_Taylor:
12508 {

```

```

12509 \group_begin:
12510   \l_fp_input_a_integer_int \c_zero
12511   \l_fp_input_a_exponent_int \c_zero
12512   \l_fp_input_b_integer_int \c_two
12513   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12514   \l_fp_input_b_exponent_int \c_zero
12515   \fp_div_internal:
12516   \fp_ln_fixed:
12517   \l_fp_input_a_integer_int \l_fp_output_integer_int
12518   \l_fp_input_a_decimal_int \l_fp_output_decimal_int
12519   \l_fp_input_a_extended_int \c_zero
12520   \l_fp_input_a_exponent_int \l_fp_output_exponent_int
12521   \l_fp_output_decimal_int \c_zero %^^A Bug?
12522   \l_fp_output_decimal_int \l_fp_input_a_decimal_int
12523   \l_fp_output_extended_int \l_fp_input_a_extended_int
12524   \fp_mul:NNNNNN
12525     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12526     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12527     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12528   \l_fp_count_int \c_one
12529   \fp_ln_Taylor_aux:
12530   \cs_set_protected_nopar:Npx \fp_tmp:w
12531   {
12532     \group_end:
12533     \l_fp_exp_integer_int \c_zero
12534     \exp_not:N \l_fp_exp_decimal_int
12535       \int_use:N \l_fp_output_decimal_int \scan_stop:
12536     \exp_not:N \l_fp_exp_extended_int
12537       \int_use:N \l_fp_output_extended_int \scan_stop:
12538     \exp_not:N \l_fp_exp_exponent_int
12539       \int_use:N \l_fp_output_exponent_int \scan_stop:
12540   }
12541   \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.

```

12542   \tex_advance:D \l_fp_exp_decimal_int \l_fp_exp_decimal_int
12543   \if_int_compare:w \l_fp_exp_extended_int < \c_five_hundred_million
12544   \else:
12545     \tex_advance:D \l_fp_exp_extended_int -\c_five_hundred_million
12546     \tex_advance:D \l_fp_exp_decimal_int \c_one
12547   \fi:
12548   \tex_advance:D \l_fp_exp_extended_int \l_fp_exp_extended_int
12549   \fp_ln_normalise:
12550   \if_int_compare:w \l_fp_output_sign_int > \c_zero
12551     \exp_after:wN \fp_add:NNNNNNNNN
12552   \else:
12553     \exp_after:wN \fp_sub:NNNNNNNNN
12554   \fi:
12555   \l_fp_output_integer_int \l_fp_output_decimal_int

```



```

12556     \l_fp_output_extended_int
12557     \c_zero \l_fp_exp_decimal_int \l_fp_exp_extended_int
12558     \l_fp_output_integer_int \l_fp_output_decimal_int
12559     \l_fp_output_extended_int
12560 }

```

The usual shifts to move to fixed-point working. This is done using the `output` registers as this saves a reassignment here.

```

12561 \cs_new_protected_nopar:Npn \fp_ln_fixed:
12562 {
12563     \if_int_compare:w \l_fp_output_exponent_int < \c_zero
12564     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
12565     \exp_after:wN \use_i:nn \exp_after:wN
12566     \fp_ln_fixed_aux:NNNNNNNNN
12567     \int_use:N \l_fp_output_decimal_int
12568     \exp_after:wN \fp_ln_fixed:
12569     \fi:
12570 }
12571 \cs_new_protected_nopar:Npn \fp_ln_fixed_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12572 {
12573     \if_int_compare:w \l_fp_output_integer_int = \c_zero
12574     \l_fp_output_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12575     \else:
12576     \tl_set:Nx \l_fp_tmp_tl
12577     {
12578         \int_use:N \l_fp_output_integer_int
12579         #1#2#3#4#5#6#7#8
12580     }
12581     \l_fp_output_integer_int \c_zero
12582     \l_fp_output_decimal_int \l_fp_tmp_tl \scan_stop:
12583     \fi:
12584     \tex_advance:D \l_fp_output_exponent_int \c_one
12585 }

```

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.

```

12586 \cs_new_protected_nopar:Npn \fp_ln_Taylor_aux:
12587 {
12588     \tex_advance:D \l_fp_count_int \c_two
12589     \fp_mul:NNNNNN
12590     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12591     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12592     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12593     \if_int_compare:w
12594     \int_eval:w
12595     \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12596     > \c_zero
12597     \fp_div_integer:NNNNN
12598     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int

```

```

12599 \l_fp_count_int
12600 \l_fp_exp_decimal_int \l_fp_exp_extended_int
12601 \tex_advance:D \l_fp_output_decimal_int \l_fp_exp_decimal_int
12602 \tex_advance:D \l_fp_output_extended_int \l_fp_exp_extended_int
12603 \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
12604 \else:
12605 \tex_advance:D \l_fp_output_decimal_int \c_one
12606 \tex_advance:D \l_fp_output_extended_int
12607 -\c_one_thousand_million
12608 \fi:
12609 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12610 \else:
12611 \tex_advance:D \l_fp_output_integer_int \c_one
12612 \tex_advance:D \l_fp_output_decimal_int
12613 -\c_one_thousand_million
12614 \fi:
12615 \exp_after:wN \fp_ln_Taylor_aux:
12616 \fi:
12617 }

```

(End definition for `\fp_ln:Nn` and `\fp_ln:cn`. These functions are documented on page ??.)

`\fp_pow:Nn` The approach used for working out powers is to first filter out the various special cases and
`\fp_pow:cn` then do most of the work using the logarithm and exponent functions. The two storage
`\fp_gpow:Nn` areas are used in the reverse of the ‘natural’ logic as this avoids some re-assignment in
`\fp_gpow:cn` the sanity checking code.

```

\fp_pow_aux:NNn 12618 \cs_new_protected_nopar:Npn \fp_pow:Nn { \fp_pow_aux:NNn \tl_set:Nn }
\fp_pow_aux_i: 12619 \cs_new_protected_nopar:Npn \fp_gpow:Nn { \fp_pow_aux:NNn \tl_gset:Nn }
\fp_pow_positive: 12620 \cs_generate_variant:Nn \fp_pow:Nn { c }
\fp_pow_negative: 12621 \cs_generate_variant:Nn \fp_gpow:Nn { c }
\fp_pow_aux_ii: 12622 \cs_new_protected_nopar:Npn \fp_pow_aux:NNn #1#2#3
\fp_pow_aux_iii: 12623 {
\fp_pow_aux_iv: 12624 \group_begin:
12625 \fp_read:N #2
12626 \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12627 \l_fp_input_b_integer_int \l_fp_input_a_integer_int
12628 \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12629 \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
12630 \fp_split:Nn a {#3}
12631 \fp_standardise:NNNN
12632 \l_fp_input_a_sign_int
12633 \l_fp_input_a_integer_int
12634 \l_fp_input_a_decimal_int
12635 \l_fp_input_a_exponent_int
12636 \if_int_compare:w
12637 \int_eval:w
12638 \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
12639 = \c_zero
12640 \if_int_compare:w
12641 \int_eval:w

```

```

12642         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12643         = \c_zero
12644         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12645         {
12646             \group_end:
12647             ##1 ##2 { \c_undefined_fp }
12648         }
12649     \else:
12650         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12651         {
12652             \group_end:
12653             ##1 ##2 { \c_zero_fp }
12654         }
12655     \fi:
12656 \else:
12657     \if_int_compare:w
12658     \int_eval:w
12659     \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12660     = \c_zero
12661     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12662     {
12663         \group_end:
12664         ##1 ##2 { \c_one_fp }
12665     }
12666 \else:
12667     \exp_after:wN \exp_after:wN \exp_after:wN
12668     \fp_pow_aux_i:
12669 \fi:
12670 \fi:
12671 \fp_tmp:w #1 #2
12672 }

```

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.

```

12673 \cs_new_protected_nopar:Npn \fp_pow_aux_i:
12674 {
12675     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12676     \tl_set:Nn \l_fp_sign_tl { + }
12677     \exp_after:wN \fp_pow_aux_ii:
12678 \else:
12679     \l_fp_input_a_extended_int \c_zero
12680     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12681     \group_begin:
12682     \fp_extended_normalise:
12683     \if_int_compare:w
12684     \int_eval:w
12685     \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12686     = \c_zero
12687     \group_end:

```

```

12688         \tl_set:Nn \l_fp_sign_tl { - }
12689         \exp_after:wN \exp_after:wN \exp_after:wN
12690         \exp_after:wN \exp_after:wN \exp_after:wN
12691         \exp_after:wN \fp_pow_aux_ii:
12692     \else:
12693         \group_end:
12694         \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12695         {
12696             \group_end:
12697             ##1 ##2 { \c_undefined_fp }
12698         }
12699     \fi:
12700 \else:
12701     \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12702     {
12703         \group_end:
12704         ##1 ##2 { \c_undefined_fp }
12705     }
12706 \fi:
12707 \fi:
12708 }

```

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.

```

12709 \cs_new_protected_nopar:Npn \fp_pow_aux_ii:
12710 {
12711     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12712     \exp_after:wN \fp_pow_aux_iv:
12713 \else:
12714     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12715     \group_begin:
12716     \l_fp_input_a_extended_int \c_zero
12717     \fp_extended_normalise:
12718     \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12719     \if_int_compare:w \l_fp_input_a_integer_int > \c_ten
12720     \group_end:
12721     \exp_after:wN \exp_after:wN \exp_after:wN
12722     \exp_after:wN \exp_after:wN \exp_after:wN
12723     \exp_after:wN \exp_after:wN \exp_after:wN
12724     \exp_after:wN \exp_after:wN \exp_after:wN
12725     \exp_after:wN \exp_after:wN \exp_after:wN
12726     \fp_pow_aux_iv:
12727 \else:
12728     \group_end:
12729     \exp_after:wN \exp_after:wN \exp_after:wN
12730     \exp_after:wN \exp_after:wN \exp_after:wN
12731     \exp_after:wN \exp_after:wN \exp_after:wN

```

```

12732         \exp_after:wN \exp_after:wN \exp_after:wN
12733         \exp_after:wN \exp_after:wN \exp_after:wN
12734         \exp_after:wN \fp_pow_aux_iii:
12735     \fi:
12736 \else:
12737     \group_end:
12738     \exp_after:wN \exp_after:wN \exp_after:wN
12739     \exp_after:wN \exp_after:wN \exp_after:wN
12740     \exp_after:wN \fp_pow_aux_iv:
12741 \fi:
12742 \else:
12743     \exp_after:wN \exp_after:wN \exp_after:wN
12744     \fp_pow_aux_iv:
12745 \fi:
12746 \fi:
12747 \cs_set_protected_nopar:Npx \fp_tmp:w ##1##2
12748 {
12749     \group_end:
12750     ##1 ##2
12751     {
12752         \l_fp_sign_tl
12753         \int_use:N \l_fp_output_integer_int
12754         .
12755         \exp_after:wN \use_none:n
12756         \int_value:w \int_eval:w
12757         \l_fp_output_decimal_int + \c_one_thousand_million
12758         e
12759         \int_use:N \l_fp_output_exponent_int
12760     }
12761 }
12762 }

```

For the small negative integer powers, the calculation is done for the positive power and the reciprocal is then taken.

```

12763 \cs_new_protected_nopar:Npn \fp_pow_aux_iii:
12764 {
12765     \l_fp_input_a_sign_int \c_one
12766     \fp_pow_aux_iv:
12767     \l_fp_input_a_integer_int \c_one
12768     \l_fp_input_a_decimal_int \c_zero
12769     \l_fp_input_a_exponent_int \c_zero
12770     \l_fp_input_b_integer_int \l_fp_output_integer_int
12771     \l_fp_input_b_decimal_int \l_fp_output_decimal_int
12772     \l_fp_input_b_exponent_int \l_fp_output_exponent_int
12773     \fp_div_internal:
12774 }

```

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.

```

12775 \cs_new_protected_nopar:Npn \fp_pow_aux_iv:
12776 {
12777   \group_begin:
12778     \l_fp_input_a_integer_int \l_fp_input_b_integer_int
12779     \l_fp_input_a_decimal_int \l_fp_input_b_decimal_int
12780     \l_fp_input_a_exponent_int \l_fp_input_b_exponent_int
12781     \fp_ln_internal:
12782     \cs_set_protected_nopar:Npx \fp_tmp:w
12783     {
12784       \group_end:
12785       \exp_not:N \l_fp_input_b_sign_int
12786       \int_use:N \l_fp_output_sign_int \scan_stop:
12787       \exp_not:N \l_fp_input_b_integer_int
12788       \int_use:N \l_fp_output_integer_int \scan_stop:
12789       \exp_not:N \l_fp_input_b_decimal_int
12790       \int_use:N \l_fp_output_decimal_int \scan_stop:
12791       \exp_not:N \l_fp_input_b_extended_int
12792       \int_use:N \l_fp_output_extended_int \scan_stop:
12793       \exp_not:N \l_fp_input_b_exponent_int
12794       \int_use:N \l_fp_output_exponent_int \scan_stop:
12795     }
12796     \fp_tmp:w
12797     \l_fp_input_a_extended_int \c_zero
12798     \fp_mul:NNNNNNNNN
12799     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
12800     \l_fp_input_a_extended_int
12801     \l_fp_input_b_integer_int \l_fp_input_b_decimal_int
12802     \l_fp_input_b_extended_int
12803     \l_fp_output_integer_int \l_fp_output_decimal_int
12804     \l_fp_output_extended_int
12805     \l_fp_output_exponent_int
12806     \int_eval:w
12807     \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
12808     \scan_stop:
12809     \fp_extended_normalise_output:
12810     \tex_multiply:D \l_fp_input_a_sign_int \l_fp_input_b_sign_int
12811     \l_fp_input_a_integer_int \l_fp_output_integer_int
12812     \l_fp_input_a_decimal_int \l_fp_output_decimal_int
12813     \l_fp_input_a_extended_int \l_fp_output_extended_int
12814     \l_fp_input_a_exponent_int \l_fp_output_exponent_int
12815     \l_fp_output_integer_int \c_zero
12816     \l_fp_output_decimal_int \c_zero
12817     \l_fp_output_extended_int \c_zero
12818     \l_fp_output_exponent_int \c_zero
12819     \cs_set_eq:NN \fp_exp_const:Nx \use_none:nn

```

```

12820     \fp_exp_internal:
12821   }
      (End definition for \fp_pow:Nn and \fp_pow:cn. These functions are documented on page ??.)

```

197.13 Tests for special values

\fp_if_undefined:N Testing for an undefined value is easy.

```

12822 \prg_new_conditional:Npnn \fp_if_undefined:N #1 { p , T , F , TF }
12823 {
12824   \if_meaning:w #1 \c_undefined_fp
12825   \prg_return_true:
12826   \else:
12827     \prg_return_false:
12828   \fi:
12829 }
      (End definition for \fp_if_undefined:N. This function is documented on page 166.)

```

\fp_if_zero:N Testing for a zero fixed-point is also easy.

```

12830 \prg_new_conditional:Npnn \fp_if_zero:N #1 { p , T , F , TF }
12831 {
12832   \if_meaning:w #1 \c_zero_fp
12833   \prg_return_true:
12834   \else:
12835     \prg_return_false:
12836   \fi:
12837 }
      (End definition for \fp_if_zero:N. This function is documented on page 166.)

```

197.14 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=: 12838 \prg_new_protected_conditional:Npnn \fp_compare:nNn #1#2#3 { T , F , TF }
\fp_compare=: 12839 {
\fp_compare<: 12840   \group_begin:
\fp_compare<_aux: 12841     \fp_split:Nn a {#1}
\fp_compare_absolute_a>b: 12842     \fp_standardise:NNNN
\fp_compare_absolute_a<b: 12843     \l_fp_input_a_sign_int
\fp_compare_>: 12844     \l_fp_input_a_integer_int
12845     \l_fp_input_a_decimal_int
12846     \l_fp_input_a_exponent_int
12847     \fp_split:Nn b {#3}
12848     \fp_standardise:NNNN
12849     \l_fp_input_b_sign_int
12850     \l_fp_input_b_integer_int
12851     \l_fp_input_b_decimal_int
12852     \l_fp_input_b_exponent_int

```

```

12853     \fp_compare_aux:N #2
12854   }
12855   \prg_new_protected_conditional:Npnn \fp_compare:NNN #1#2#3 { T , F , TF }
12856   {
12857     \group_begin:
12858     \fp_read:N #3
12859     \l_fp_input_b_sign_int      \l_fp_input_a_sign_int
12860     \l_fp_input_b_integer_int   \l_fp_input_a_integer_int
12861     \l_fp_input_b_decimal_int   \l_fp_input_a_decimal_int
12862     \l_fp_input_b_exponent_int  \l_fp_input_a_exponent_int
12863     \fp_read:N #1
12864     \fp_compare_aux:N #2
12865   }
12866   \cs_new_protected_nopar:Npn \fp_compare_aux:N #1
12867   {
12868     \cs_if_exist:cTF { fp_compare_#1: }
12869     { \use:c { fp_compare_#1: } }
12870     {
12871       \group_end:
12872       \prg_return_false:
12873     }
12874   }

```

For equality, the test is pretty easy as things are either equal or they are not.

```

12875   \cs_new_protected_nopar:cpn { fp_compare_=: }
12876   {
12877     \if_int_compare:w \l_fp_input_a_sign_int = \l_fp_input_b_sign_int
12878     \if_int_compare:w \l_fp_input_a_integer_int = \l_fp_input_b_integer_int
12879     \if_int_compare:w \l_fp_input_a_decimal_int = \l_fp_input_b_decimal_int
12880     \if_int_compare:w
12881       \l_fp_input_a_exponent_int = \l_fp_input_b_exponent_int
12882     \group_end:
12883     \prg_return_true:
12884   \else:
12885     \group_end:
12886     \prg_return_false:
12887   \fi:
12888   \else:
12889     \group_end:
12890     \prg_return_false:
12891   \fi:
12892   \else:
12893     \group_end:
12894     \prg_return_false:
12895   \fi:
12896   \else:
12897     \group_end:
12898     \prg_return_false:
12899   \fi:
12900   }

```


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.

```

12901 \cs_new_protected_nopar:cpn { fp_compare_>: }
12902 {
12903   \if_int_compare:w \int_eval:w
12904     \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12905     = \c_zero
12906   \if_int_compare:w \int_eval:w
12907     \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
12908     = \c_zero
12909   \group_end:
12910   \prg_return_false:
12911   \else:
12912     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12913     \group_end:
12914     \prg_return_false:
12915     \else:
12916       \group_end:
12917       \prg_return_true:
12918     \fi:
12919   \fi:
12920   \else:
12921     \if_int_compare:w \int_eval:w
12922       \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
12923       = \c_zero
12924     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12925     \group_end:
12926     \prg_return_true:
12927     \else:
12928       \group_end:
12929       \prg_return_false:
12930     \fi:
12931   \else:
12932     \use:c { fp_compare_>_aux: }
12933   \fi:
12934   \fi:
12935 }

```

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.

```

12936 \cs_new_protected_nopar:cpn { fp_compare_>_aux: }
12937 {
12938   \if_int_compare:w \l_fp_input_a_sign_int > \l_fp_input_b_sign_int
12939   \group_end:
12940   \prg_return_true:
12941   \else:
12942     \if_int_compare:w \l_fp_input_a_sign_int < \l_fp_input_b_sign_int
12943     \group_end:
12944     \prg_return_false:
12945   \else:

```

```

12946         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12947         \use:c { fp_compare_absolute_a>b: }
12948     \else:
12949         \use:c { fp_compare_absolute_a<b: }
12950     \fi:
12951 \fi:
12952 \fi:
12953 }

```

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.

```

12954 \cs_new_protected_nopar:cpn { fp_compare_absolute_a>b: }
12955 {
12956     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
12957     \group_end:
12958     \prg_return_true:
12959 \else:
12960     \if_int_compare:w \l_fp_input_a_exponent_int < \l_fp_input_b_exponent_int
12961     \group_end:
12962     \prg_return_false:
12963 \else:
12964     \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
12965     \group_end:
12966     \prg_return_true:
12967 \else:
12968     \if_int_compare:w
12969         \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
12970     \group_end:
12971     \prg_return_false:
12972 \else:
12973     \if_int_compare:w
12974         \l_fp_input_a_decimal_int > \l_fp_input_b_decimal_int
12975     \group_end:
12976     \prg_return_true:
12977 \else:
12978     \group_end:
12979     \prg_return_false:
12980 \fi:
12981 \fi:
12982 \fi:
12983 \fi:
12984 \fi:
12985 }
12986 \cs_new_protected_nopar:cpn { fp_compare_absolute_a<b: }
12987 {
12988     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
12989     \group_end:
12990     \prg_return_true:
12991 \else:

```

```

12992     \if_int_compare:w \l_fp_input_b_exponent_int < \l_fp_input_a_exponent_int
12993     \group_end:
12994     \prg_return_false:
12995   \else:
12996     \if_int_compare:w \l_fp_input_b_integer_int > \l_fp_input_a_integer_int
12997     \group_end:
12998     \prg_return_true:
12999   \else:
13000     \if_int_compare:w
13001       \l_fp_input_b_integer_int < \l_fp_input_a_integer_int
13002     \group_end:
13003     \prg_return_false:
13004   \else:
13005     \if_int_compare:w
13006       \l_fp_input_b_decimal_int > \l_fp_input_a_decimal_int
13007     \group_end:
13008     \prg_return_true:
13009   \else:
13010     \group_end:
13011     \prg_return_false:
13012   \fi:
13013 \fi:
13014 \fi:
13015 \fi:
13016 \fi:
13017 }

```

This is just a case of reversing the two input values and then running the tests already defined.

```

13018 \cs_new_protected_nopar:cpn { fp_compare_<: }
13019 {
13020   \tl_set:Nx \l_fp_tmp_tl
13021   {
13022     \int_set:Nn \exp_not:N \l_fp_input_a_sign_int
13023     { \int_use:N \l_fp_input_b_sign_int }
13024     \int_set:Nn \exp_not:N \l_fp_input_a_integer_int
13025     { \int_use:N \l_fp_input_b_integer_int }
13026     \int_set:Nn \exp_not:N \l_fp_input_a_decimal_int
13027     { \int_use:N \l_fp_input_b_decimal_int }
13028     \int_set:Nn \exp_not:N \l_fp_input_a_exponent_int
13029     { \int_use:N \l_fp_input_b_exponent_int }
13030     \int_set:Nn \exp_not:N \l_fp_input_b_sign_int
13031     { \int_use:N \l_fp_input_a_sign_int }
13032     \int_set:Nn \exp_not:N \l_fp_input_b_integer_int
13033     { \int_use:N \l_fp_input_a_integer_int }
13034     \int_set:Nn \exp_not:N \l_fp_input_b_decimal_int
13035     { \int_use:N \l_fp_input_a_decimal_int }
13036     \int_set:Nn \exp_not:N \l_fp_input_b_exponent_int
13037     { \int_use:N \l_fp_input_a_exponent_int }
13038   }

```

```

13039 \l_fp_tmp_tl
13040 \use:c { fp_compare_>: }
13041 }

```

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

\fp_compare:n 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_aux_i:w
\fp_compare_aux_ii:w
\fp_compare_aux_iii:w 13042 \prg_new_protected_conditional:Npnn \fp_compare:n #1 { T , F , TF }
\fp_compare_aux_iv:w 13043 {
\fp_compare_aux_v:w 13044 \group_begin:
\fp_compare_aux_vi:w 13045 \tl_set:Nx \l_fp_tmp_tl
\fp_compare_aux_vii:w 13046 {
13047 \group_end:
13048 \fp_compare_aux_i:w #1 \exp_not:n { == \q_nil == \q_stop }
13049 }
13050 \l_fp_tmp_tl
13051 }
13052 \cs_new_protected_nopar:Npn \fp_compare_aux_i:w #1 == #2 == #3 \q_stop
13053 {
13054 \quark_if_nil:nTF {#2}
13055 { \fp_compare_aux_ii:w #1 != \q_nil != \q_stop }
13056 { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13057 }
13058 \cs_new_protected_nopar:Npn \fp_compare_aux_ii:w #1 != #2 != #3 \q_stop
13059 {
13060 \quark_if_nil:nTF {#2}
13061 { \fp_compare_aux_iii:w #1 <= \q_nil <= \q_stop }
13062 { \fp_compare:nNnTF {#1} = {#2} \prg_return_false: \prg_return_true: }
13063 }
13064 \cs_new_protected_nopar:Npn \fp_compare_aux_iii:w #1 <= #2 <= #3 \q_stop
13065 {
13066 \quark_if_nil:nTF {#2}
13067 { \fp_compare_aux_iv:w #1 >= \q_nil >= \q_stop }
13068 { \fp_compare:nNnTF {#1} > {#2} \prg_return_false: \prg_return_true: }
13069 }
13070 \cs_new_protected_nopar:Npn \fp_compare_aux_iv:w #1 >= #2 >= #3 \q_stop
13071 {
13072 \quark_if_nil:nTF {#2}
13073 { \fp_compare_aux_v:w #1 = \q_nil \q_stop }
13074 { \fp_compare:nNnTF {#1} < {#2} \prg_return_false: \prg_return_true: }
13075 }
13076 \cs_new_protected_nopar:Npn \fp_compare_aux_v:w #1 = #2 = #3 \q_stop
13077 {
13078 \quark_if_nil:nTF {#2}
13079 { \fp_compare_aux_vi:w #1 < \q_nil < \q_stop }
13080 { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13081 }
13082 \cs_new_protected_nopar:Npn \fp_compare_aux_vi:w #1 < #2 < #3 \q_stop

```

```

13083 {
13084   \quark_if_nil:nTF {#2}
13085   { \fp_compare_aux_vii:w #1 > \q_nil > \q_stop }
13086   { \fp_compare:nNnTF {#1} < {#2} \prg_return_true: \prg_return_false: }
13087 }
13088 \cs_new_protected_nopar:Npn \fp_compare_aux_vii:w #1 > #2 > #3 \q_stop
13089 {
13090   \quark_if_nil:nTF {#2}
13091   { \prg_return_false: }
13092   { \fp_compare:nNnTF {#1} > {#2} \prg_return_true: \prg_return_false: }
13093 }

```

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

197.15 Messages

\fp_overflow_msg: A generic overflow message, used whenever there is a possible overflow.

```

13094 \msg_kernel_new:nnnn { fpu } { overflow }
13095 { Number~too~big. }
13096 {
13097   The~input~given~is~too~big~for~the~LaTeX~floating~point~unit. \\
13098   Further~errors~may~well~occur!
13099 }
13100 \cs_new_protected_nopar:Npn \fp_overflow_msg:
13101 { \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.

```

13102 \msg_kernel_new:nnnn { fpu } { exponent-overflow }
13103 { Number~too~big~for~exponent~unit. }
13104 {
13105   The~exponent~of~the~input~given~is~too~big~for~the~floating~point~
13106   unit:~the~maximum~input~value~for~an~exponent~is~230.
13107 }
13108 \cs_new_protected_nopar:Npn \fp_exp_overflow_msg:
13109 { \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

```

13110 \msg_kernel_new:nnnn { fpu } { logarithm-input-error }
13111 { Invalid~input~to~ln~function. }
13112 { Logarithms~can~only~be~calculated~for~positive~numbers. }
13113 \cs_new_protected_nopar:Npn \fp_ln_error_msg: {
13114   \msg_kernel_error:nn { fpu } { logarithm-input-error }
13115 }

```

(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.
13116 \msg_kernel_new:nnnn { fpu } { trigonometric-overflow }
13117 { Number~too~big~for~trigonometry~unit. }
13118 {
13119   The~trigonometry~code~can~only~work~with~numbers~smaller~
13120   than~1000000000.
13121 }
13122 \cs_new_protected_nopar:Npn \fp_trig_overflow_msg:
13123 { \msg_kernel_error:nn { fpu } { trigonometric-overflow } }
      (End definition for \fp_trig_overflow_msg:. This function is documented on page ??.)
13124 </initex | package>

```

198 l3luatex implementation

```

13125 <*initex | package>

      Announce and ensure that the required packages are loaded.
13126 <*package>
13127 \ProvidesExplPackage
13128   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
13129 \package_check_loaded_expl:
13130 </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
13131 \luatex_if_engine:TF
13132 {
13133   \cs_new_eq:NN \lua_now:x \luatex_directlua:D
13134   \cs_new_eq:NN \lua_shipout_x:n \luatex_latelua:D
13135 }
13136 {
13137   \cs_new:Npn \lua_now:x #1
13138   {
13139     \msg_expandable_error:n
13140       { LuaTeX~ engine~ not~ in~ use!~ Ignoring~ \lua_now:x. }
13141   }
13142   \cs_new_protected:Npn \lua_shipout_x:n #1
13143   {
13144     \msg_expandable_error:n
13145       { LuaTeX~ engine~ not~ in~ use!~ Ignoring~ \lua_shipout_x:n. }
13146   }
13147 }
13148 \cs_new:Npn \lua_now:n #1
13149 { \lua_now:x { \exp_not:n {#1} } }
13150 \cs_generate_variant:Nn \lua_shipout_x:n { x }
13151 \cs_new_protected:Npn \lua_shipout:n #1
13152 { \lua_shipout_x:n { \exp_not:n {#1} } }

```

```

13153 \cs_generate_variant:Nn \lua_shipout:n { x }
      (End definition for \lua_now:n and \lua_now:x. These functions are documented on page ??.)

```

198.1 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`

```

13154 \int_new:N \g_cctab_allocate_int
13155 \int_set:Nn \g_cctab_allocate_int { -1 }
13156 \int_new:N \g_cctab_stack_int
13157 \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.

```

13158 \cs_new_protected_nopar:Npn \cctab_new:N #1
13159 {
13160   \cs_if_free:NTF #1
13161   {
13162     \int_gadd:Nn \g_cctab_allocate_int { 2 }
13163     \int_compare:nNnTF
13164       { \g_cctab_allocate_int } < { \c_max_register_int + 1 }
13165     {
13166       \tex_global:D \tex_mathchardef:D #1 \g_cctab_allocate_int
13167       \luatex_initcatcodetable:D #1
13168     }
13169     { \msg_kernel_fatal:nxx { alloc } { out-of-registers } { cctab } }
13170   }
13171   {
13172     \msg_kernel_error:nxx { code } { variable-already-defined }
13173     { \token_to_str:N #1 }
13174   }
13175 }
13176 \luatex_if_engine:F
13177 { \cs_set_protected_nopar:Npn \cctab_new:N #1 { \lua_wrong_engine: } }
13178 <*package>
13179 \luatex_if_engine:T
13180 {
13181   \cs_set_protected_nopar:Npn \cctab_new:N #1
13182   {
13183     \newcatcodetable #1
13184     \luatex_initcatcodetable:D #1
13185   }
13186 }
13187 </package>

```

(End definition for `\cctab_new:N`. This function is documented on page 173.)

`\cctab_begin:N` The aim here is to ensure that the saved tables are read-only. This is done by using a stack of tables which are not read only, and actually having them as “in use” copies.

```

\l_cctab_tmp_tl 13188 \cs_new_protected_nopar:Npn \cctab_begin:N #1
13189 {
13190   \seq_gpush:Nx \g_cctab_stack_seq { \tex_the:D \luatex_catcodetable:D }
13191   \luatex_catcodetable:D #1
13192   \int_gadd:Nn \g_cctab_stack_int { 2 }
13193   \int_compare:nNnT { \g_cctab_stack_int } > { 268 435 453 }
13194     { \msg_kernel_error:nn { code } { cctab-stack-full } }
13195   \luatex_savecatcodetable:D \g_cctab_stack_int
13196   \luatex_catcodetable:D \g_cctab_stack_int
13197 }
13198 \cs_new_protected_nopar:Npn \cctab_end:
13199 {
13200   \int_gsub:Nn \g_cctab_stack_int { 2 }
13201   \seq_gpop:NN \g_cctab_stack_seq \l_cctab_tmp_tl
13202   \quark_if_no_value:NT \l_cctab_tmp_tl
13203     { \tl_set:Nn \l_cctab_tmp_tl { 0 } }
13204   \luatex_catcodetable:D \l_cctab_tmp_tl \scan_stop:
13205 }
13206 \luatex_if_engine:F
13207 {
13208   \cs_set_protected_nopar:Npn \cctab_begin:N #1 { \lua_wrong_engine: }
13209   \cs_set_protected_nopar:Npn \cctab_end: { \lua_wrong_engine: }
13210 }
13211 <*package>
13212 \luatex_if_engine:T
13213 {
13214   \cs_set_protected_nopar:Npn \cctab_begin:N #1 { \BeginCatcodeRegime #1 }
13215   \cs_set_protected_nopar:Npn \cctab_end: { \EndCatcodeRegime }
13216 }
13217 </package>
13218 \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.

```

13219 \cs_new_protected:Npn \cctab_gset:Nn #1#2
13220 {
13221   \group_begin:
13222     #2
13223     \luatex_savecatcodetable:D #1
13224   \group_end:
13225 }
13226 \luatex_if_engine:F
13227 { \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 173.)

`\c_code_cctab` Creating category code tables is easy using the function above. The `other` and `string`
`\c_document_cctab` ones are done by completely ignoring the existing codes as this makes life a lot less
`\c_initex_cctab` complex. The table for `expl3` category codes is always needed, whereas when in package
`\c_other_cctab` mode the rest can be copied from the existing L^AT_EX 2_ε package `luatex`.
`\c_string_cctab`

```

13228 \luatex_if_engine:T
13229 {
13230   \cctab_new:N \c_code_cctab
13231   \cctab_gset:Nn \c_code_cctab { }
13232 }
13233 <*package>
13234 \luatex_if_engine:T
13235 {
13236   \cs_new_eq:NN \c_document_cctab \CatcodeTableLaTeX
13237   \cs_new_eq:NN \c_initex_cctab \CatcodeTableIniTeX
13238   \cs_new_eq:NN \c_other_cctab \CatcodeTableOther
13239   \cs_new_eq:NN \c_string_cctab \CatcodeTableString
13240 }
13241 </package>
13242 <*initex>
13243 \luatex_if_engine:T
13244 {
13245   \cctab_new:N \c_document_cctab
13246   \cctab_new:N \c_other_cctab
13247   \cctab_new:N \c_string_cctab
13248   \cctab_gset:Nn \c_document_cctab
13249   {
13250     \char_set_catcode_space:n { 9 }
13251     \char_set_catcode_space:n { 32 }
13252     \char_set_catcode_other:n { 58 }
13253     \char_set_catcode_math_subscript:n { 95 }
13254     \char_set_catcode_active:n { 126 }
13255   }
13256   \cctab_gset:Nn \c_other_cctab
13257   {
13258     \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13259     { \char_set_catcode_other:n {#1} }
13260   }
13261   \cctab_gset:Nn \c_string_cctab
13262   {
13263     \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13264     { \char_set_catcode_other:n {#1} }
13265     \char_set_catcode_space:n { 32 }
13266   }
13267 }
13268 </initex>
13269 </initex | package>

```

(End definition for `\c_code_cctab`. This function is documented on page 174.)

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	
\!	8549
\#	5, 8252
\%	8254
\	2264, 2265, 2278, 2279, 8549, 8550
*	2572, 2574, 2578, 2585, 8209, 8210
\,	9008, 9010
\-	348
\.	8491, 8496
\.bool_gset:N	152, 9350
\.bool_gset_inverse:N	152, 9354
\.bool_set:N	151, 9350
\.bool_set_inverse:N	152, 9354
\.choice:	152, 9358
\.choice_code:n	152, 9366
\.choice_code:x	152, 9366
\.choices:nn	152, 9360
\.clist_gset:N	152, 9370
\.clist_gset:c	152, 9370
\.clist_set:N	152, 9370
\.clist_set:c	152, 9370
\.code:n	153, 9362
\.code:x	153, 9362
\.default:V	153, 9378
\.default:n	153, 9378
\.dim_gset:N	153, 9382
\.dim_gset:c	153, 9382
\.dim_set:N	153, 9382
\.dim_set:c	153, 9382
\.fp_gset:N	153, 9390
\.fp_gset:c	153, 9390
\.fp_set:N	153, 9390
\.fp_set:c	153, 9390
\.generate_choices:n	154, 9398
\.int_gset:N	154, 9400
\.int_gset:c	154, 9400
\.int_set:N	154, 9400
\.int_set:c	154, 9400
\.meta:n	154, 9408
\.meta:x	154, 9408
\.multichoice:	154, 9412
\.multichoice:nn	154
\.multichoices:nn	9412
\.skip_gset:N	154, 9416
\.skip_gset:c	154, 9416
\.skip_set:N	154, 9416
\.skip_set:c	154, 9416
\.tl_gset:N	155, 9424
\.tl_gset:c	155, 9424
\.tl_gset_x:N	155, 9424
\.tl_gset_x:c	155, 9424
\.tl_set:N	155, 9424
\.tl_set:c	155, 9424
\.tl_set_x:N	155, 9424
\.tl_set_x:c	155, 9424
\.value_forbidden:	155, 9440
\.value_required:	155, 9440
\/	347
\:	1067, 2664, 2797
\::	30,
	1520, 1521, 1522, 1522–1525, 1527,
	1529, 1536, 1541, 1547, 1668–1694,
	1696, 1701, 1703, 1708, 1734–1736
\::N	30, 1524,
	1524, 1677, 1683, 1684, 1688, 1689
\::V	30, 1541, 1541, 1674
\::V_unbraced	1695, 1703
\::c	30, 1525, 1525,
	1669, 1675, 1678, 1685, 1692, 1693
\::f	30, 1529, 1529, 1670–1672, 1736
\::f_unbraced	1695, 1696
\::n	30, 1523, 1523,
	1669, 1672–1674, 1679, 1683, 1685,
	1686, 1688, 1690, 1691, 1693, 1734
\::o	30, 1527, 1527, 1670, 1673,
	1675, 1676, 1680, 1681, 1683, 1684,
	1686, 1687, 1689, 1691, 1694, 1735
\::o_unbraced	1695, 1701, 1734–1736
\::v	30, 1541, 1547
\::v_unbraced	1695, 1708
\::x	30, 1536,
	1536, 1668, 1677–1682, 1688–1694
\;	2664, 2796, 2797
\=	9007, 9009
\?	1818, 2704
\@	1067, 1068, 4303, 4304
\@end	763
\@hyph	766

`\@input` 767
`\@italiccorr` 768
`\@underline` 769
`\@addtofilelist` 9731
`\@currname` 9659
`\@filelist` 9756
`\@ifpackageloaded` 216
`\@namedef` 198
`\@nil` 193, 201
`\@popfilename` 163, 181, 183
`\@pushfilename` 163, 164, 179
`\@tempa` 54, 56, 64
`\@tempboxa` 6425
`\` 1818, 2704,
 8167, 8174, 8257, 8267, 8315, 8336,
 8453, 8471, 8473, 8478, 8479, 8516,
 8518, 8524, 8526, 8590, 8666, 8674,
 8821, 8843, 8863, 8871, 8878, 8885,
 8894, 8902, 8903, 8947, 9581, 9596,
 9597, 9603, 9616, 9629, 9635, 13097
`\{` 3, 5312, 5866, 6229, 6231, 8251
`\}` ... 4, 5312, 5866, 6229, 6231, 8253, 8548
`\^` 6, 9, 249, 3059, 3066, 8216, 8952
`_` 2278
`\|` 3874
`\~` 8255

Numbers

`\8` 9009
`\9` 9010
`_` . 346, 1059, 8210, 8258, 8548, 8904, 8953

A

`\A` 2663, 2705, 4303, 4305
`\above` 467
`\abovedisplayshortskip` 480
`\abovedisplayskip` 481
`\abovewithdelims` 468
`\accent` 518
`\adjdemerits` 555
`\advance` 362
`\afterassignment` 372
`\aftergroup` 373
`\alloc_reg:nNN` 7963, 7966
`\alloc_setup_type:nnn` 7961, 7964
`\AtBeginDocument` 9754
`\atop` 469
`\atopwithdelims` 470

B

`\B` 4304, 4306
`\badness` 617
`\baselineskip` 545
`\batchmode` 438
`\begin` 8512
`\BeginCatcodeRegime` 13214
`\begingroup` . 12, 61, 100, 228, 232, 338, 376
`\beginL` 730
`\beginR` 732
`\belowdisplayshortskip` 482
`\belowdisplayskip` 483
`\binoppenalty` 506
`\bool(:w` 1917
`\bool_)_0:w` 1917
`\bool_)_1:w` 1917
`\bool_8_0:w` 1917
`\bool_8_1:w` 1917
`\bool_:w` 1917
`\bool_choose:NN` 1917, 2006, 2014
`\bool_cleanup:N` .. 1917, 1999, 2002, 2004
`\bool_do_until:cn` 2062
`\bool_do_until:Nn` 2062, 2064, 2065, 2067
`\bool_do_until:nn` 2068, 2089, 2092
`\bool_do_while:cn` 2062
`\bool_do_while:Nn` 2062, 2062, 2063, 2066
`\bool_do_while:nn` 2068, 2076, 2079
`\bool_eval_skip_to_end:Nw` 1917,
 2026, 2027, 2029, 2031, 2033, 2047
`\bool_eval_skip_to_end_aux:Nw`
 1917, 2035, 2039
`\bool_eval_skip_to_end_aux_ii:Nw` ...
 1917, 2043, 2045
`\bool_get_next:N` ... 1917, 1928, 1930,
 1950, 1979, 1999, 2001, 2016–2019
`\bool_get_next:NN` 1950, 1958
`\bool_get_not_next:N` ... 1940, 1951, 1978
`\bool_get_not_next:NN` 1951, 1971
`\bool_gset:cn` 1897
`\bool_gset:Nn` 36, 1897, 1899, 1902
`\bool_gset_eq:cc` 1889, 1896
`\bool_gset_eq:cN` 1889, 1895
`\bool_gset_eq:Nc` 1889, 1894
`\bool_gset_eq:NN` 35, 1889, 1893
`\bool_gset_false:c` 1877
`\bool_gset_false:N` .. 35, 1877, 1883, 1888
`\bool_gset_true:c` 1877
`\bool_gset_true:N` .. 35, 1877, 1881, 1887
`\bool_I_0:w` 1917
`\bool_I_1:w` 1917

\bool_if:c	1903	\botmark	453
\bool_if:N	1903, 1903	\botmarks	679
\bool_if:n	1917, 1917	\box	661
\bool_if:NF	294, 1913, 2059, 2065, 3704, 7771, 9543	\box_clear:c	6331
\bool_if:nF	2083, 2092	\box_clear:N	121, 6331, 6331, 6335, 6880, 6936, 6981
\bool_if:NT	1912, 2057, 2063, 3704, 3705, 7137, 9508, 10602	\box_clear_new:c	6337
\bool_if:nT	2070, 2079	\box_clear_new:N	121, 6337, 6337, 6349
\bool_if:NTF	36, 1914, 3690, 8298, 9169, 10615	\box_dp:c	6363
\bool_if:nTF	37, 2929, 3079, 4109, 9483, 9493	\box_dp:N	123, 6363, 6364, 6367, 6370, 6547, 6660, 6707, 6728, 6750, 7042, 7043, 7091, 7093, 7110, 7124, 7286, 7304, 7452, 7841, 7855
\bool_if_p:N	1911	\box_gclear:c	6331
\bool_if_p:n	1898, 1900, 1919, 1925, 2049, 2052	\box_gclear:N	121, 6331, 6333, 6336
\bool_new:c	1875	\box_gclear_new:c	6337
\bool_new:N	35, 1875, 1875, 1876, 1915, 1916, 6837, 8207, 9117, 9186, 9201, 9825	\box_gclear_new:N	121, 6337, 6343, 6350
\bool_Not:N	1960, 1980	\box_gset_eq:cc	6351
\bool_Not:w	1917, 1955, 1978	\box_gset_eq:cN	6351
\bool_not_choose:NN	2011, 2015	\box_gset_eq:Nc	6351
\bool_not_cleanup:N	2001, 2003, 2009	\box_gset_eq:NN	121, 6334, 6346, 6351, 6353, 6356
\bool_not_Not:N	1973, 1989	\box_gset_eq_clear:cc	6357
\bool_not_Not:w	1968, 1979	\box_gset_eq_clear:cN	6357
\bool_not_p:n	37, 2049, 2049	\box_gset_eq_clear:Nc	6357
\bool_p:w	1917, 1982, 1991	\box_gset_eq_clear:NN	122, 6357, 6359, 6362
\bool_S_0:w	1917	\box_gset_to_last:c	6412
\bool_S_1:w	1917	\box_gset_to_last:N	125, 6412, 6414, 6417
\bool_set:cn	1897	\box_ht:c	6363
\bool_set:Nn	35, 1897, 1897, 1901	\box_ht:N	123, 6363, 6363, 6366, 6372, 6546, 6659, 6706, 6727, 6749, 6932, 6976, 7041, 7043, 7087, 7089, 7110, 7117, 7285, 7303, 7449, 7451, 7839, 7854
\bool_set_eq:cc	1889, 1892	\box_if_empty:c	6405
\bool_set_eq:cN	1889, 1891	\box_if_empty:N	6405, 6405
\bool_set_eq:Nc	1889, 1890	\box_if_empty:NF	6409
\bool_set_eq:NN	35, 1889, 1889	\box_if_empty:NT	6408
\bool_set_false:c	1877	\box_if_empty:NTF	125, 6410
\bool_set_false:N	35, 309, 1877, 1879, 1886, 7133, 7769, 8300, 9138, 9475, 10592, 10621	\box_if_empty_p:N	6407
\bool_set_true:c	1877	\box_if_horizontal:c	6393
\bool_set_true:N	35, 323, 1877, 1877, 1885, 7151, 7166, 7199, 8249, 8339, 9133, 9470, 10629	\box_if_horizontal:N	6393, 6393
\bool_until_do:cn	2056	\box_if_horizontal:NF	6399
\bool_until_do:Nn	37, 2056, 2058, 2059, 2061	\box_if_horizontal:NT	6398
\bool_until_do:nn	38, 2068, 2081, 2086	\box_if_horizontal:NTF	125, 6400
\bool_while_do:cn	2056	\box_if_horizontal_p:N	6397
\bool_while_do:Nn	37, 2056, 2056, 2057, 2060	\box_if_vertical:c	6393
\bool_while_do:nn	38, 2068, 2068, 2073	\box_if_vertical:N	6393, 6395
\bool_xor_p:nn	37, 2050, 2050	\box_if_vertical:NF	6403
		\box_if_vertical:NT	6402

- \box_if_vertical:NTF 125, 6404
 - \box_if_vertical_p:N 6401
 - \box_move_down:nn . 122, 6382, 6388, 7432
 - \box_move_left:nn 122, 6382, 6382
 - \box_move_right:nn 122, 6382, 6384
 - \box_move_up:nn 122, 6382, 6386, 7324, 7836
 - \box_new:c 6323
 - \box_new:N
 - 121, 6323, 6324, 6330, 6341, 6347,
 - 6422, 6428, 6430, 6521, 6811, 6887
 - \box_resize:cnn 6654
 - \box_resize:Nnn 124, 6654, 6654, 6680, 7547
 - \box_resize_aux:Nnn
 - 6654, 6674, 6676, 6681, 6717, 6737
 - \box_resize_common:N 6699, 6778, 6780, 6780
 - \box_resize_to_ht_plus_dp:cn 6701
 - \box_resize_to_ht_plus_dp:Nn
 - 124, 6701, 6701, 6721
 - \box_resize_to_wd:cn 6701
 - \box_resize_to_wd:Nn 124, 6701, 6722, 6741
 - \box_rotate:Nn 124, 6527, 6527, 7427
 - \box_rotate_aux:N 6527, 6538, 6540, 6544
 - \box_rotate_quadrant_four:
 - 6527, 6559, 6641
 - \box_rotate_quadrant_one: 6527, 6553, 6608
 - \box_rotate_quadrant_three:
 - 6527, 6558, 6630
 - \box_rotate_quadrant_two: 6527, 6554, 6619
 - \box_rotate_set_sin_cos: 6527, 6533, 6578
 - \box_rotate_x:nnN
 - 6527, 6586, 6614, 6616,
 - 6625, 6627, 6636, 6638, 6647, 6649
 - \box_rotate_y:nnN
 - 6527, 6597, 6610, 6612,
 - 6621, 6623, 6632, 6634, 6643, 6645
 - \box_scale:cnn 6742
 - \box_scale:Nnn 124, 6742, 6742, 6763, 7574
 - \box_scale_aux:Nnn 6742, 6757, 6759, 6764
 - \box_set_dp:cn 6369
 - \box_set_dp:Nn . 123, 6369, 6369, 6376,
 - 6573, 6789, 7286, 7304, 7437, 7840
 - \box_set_eq:cc 6351
 - \box_set_eq:cN 6351
 - \box_set_eq:Nc 6351
 - \box_set_eq:NN . 121, 6332, 6340, 6351,
 - 6351, 6354, 6355, 6991, 7306, 7844
 - \box_set_eq_clear:cc 6357
 - \box_set_eq_clear:cN 6357
 - \box_set_eq_clear:Nc 6357
 - \box_set_eq_clear:NN
 - 122, 6357, 6357, 6360, 6361
 - \box_set_ht:cn 6369
 - \box_set_ht:Nn . 123, 6369, 6371, 6375,
 - 6572, 6788, 7285, 7303, 7435, 7838
 - \box_set_to_last:c 6412
 - \box_set_to_last:N
 - 125, 6412, 6412, 6415, 6416
 - \box_set_wd:cn 6369
 - \box_set_wd:Nn . 123, 6369, 6373, 6377,
 - 6574, 6800, 7287, 7305, 7438, 7842
 - \box_show:c 6431
 - \box_show:N 126, 6431, 6431, 6432
 - \box_use:c 6378
 - \box_use:N 122, 6378, 6379, 6381,
 - 6537, 6561, 6568, 6576, 6673, 6716,
 - 6736, 6756, 6785, 6795, 6801, 7321,
 - 7324, 7402, 7433, 7763, 7833, 7836
 - \box_use_clear:c 6378
 - \box_use_clear:N .. 122, 6378, 6378, 6380
 - \box_wd:c 6363
 - \box_wd:N 123, 6363,
 - 6365, 6368, 6374, 6548, 6661, 6708,
 - 6729, 6751, 7044, 7089, 7093, 7099,
 - 7104, 7254, 7287, 7305, 7322, 7451,
 - 7456, 7616, 7623, 7834, 7843, 7856
 - \boxmaxdepth 623
 - \brokenpenalty 580
- C
- \C 1824, 2705
 - \c_active_char_token 3128, 3129
 - \c_alignment_tab_token 3122, 3123
 - \c_alignment_token
 - 50, 2569, 2575, 2605, 3123
 - \c_catcode_active_tl
 - 50, 2584, 2586, 2643, 3129
 - \c_catcode_letter_token
 - 50, 2569, 2581, 2633, 3125
 - \c_catcode_other_space_tl
 - 139, 8208, 8211, 8221, 8223, 8225, 8258
 - \c_catcode_other_token
 - 50, 2569, 2582, 2638, 3126
 - \c_code_cctab ... 173, 13228, 13230, 13231
 - \c_coffin_corners_prop
 - 6815, 6815–6819, 6891, 7013
 - \c_coffin_poles_prop 6820, 6820,
 - 6822–6824, 6826–6831, 6893, 7015
 - \c_document_cctab
 - 174, 13228, 13236, 13245, 13248

<code>\c_e_fp</code>	170 , 9784 , 9784	<code>\c_fp_exp_30_tl</code>	11926
<code>\c_eight</code>	67 , 2497 , 2529 , 3766 , 3829 , 3834 , 7940 , 10593	<code>\c_fp_exp_3_tl</code>	11926
<code>\c_eleven</code>	67 , 2503 , 2535 , 3829 , 3837 , 7943	<code>\c_fp_exp_40_tl</code>	11926
<code>\c_empty_box</code>	125 , 6332 , 6334 , 6340 , 6346 , 6418 , 6419 , 6422 , 7401	<code>\c_fp_exp_4_tl</code>	11926
<code>\c_empty_coffin</code> ..	6996 , 6996 , 6997 , 7399	<code>\c_fp_exp_50_tl</code>	11926
<code>\c_empty_prop</code> ..	119 , 6003 , 6003 – 6009 , 6122	<code>\c_fp_exp_5_tl</code>	11926
<code>\c_empty_tl</code>	93 , 3579 , 4201 , 4217 , 4219 , 4443 , 4783 , 4783 , 5204 , 5318	<code>\c_fp_exp_60_tl</code>	11926
<code>\c_false_bool</code>	20 , 986 , 1015 , 1055 , 1056 , 1084 , 1436 , 1438 , 1447 , 1459 , 1875 , 1880 , 1884 , 1984 , 1995 , 2020 , 2023 , 2024 , 2026 , 2029 , 2053 , 2749 , 3679 , 3684 , 3693 , 4742	<code>\c_fp_exp_6_tl</code>	11926
<code>\c_fifteen</code>	67 , 2511 , 2543 , 3829 , 3840 , 7947	<code>\c_fp_exp_70_tl</code>	11926
<code>\c_five</code>	67 , 2491 , 2523 , 3829 , 3833 , 7937 , 10246 , 11503 , 11801	<code>\c_fp_exp_7_tl</code>	11926
<code>\c_five_hundred_million</code> ..	9767 , 9770 , 10283 , 12192 , 12348 , 12543 , 12545	<code>\c_fp_exp_80_tl</code>	11926
<code>\c_forty_four</code>	9767 , 9767 , 10411	<code>\c_fp_exp_8_tl</code>	11926
<code>\c_four</code>	67 , 2489 , 2521 , 3829 , 3832 , 7936 , 8343 , 8349 , 10392 , 10628 , 11440 , 11441	<code>\c_fp_exp_90_tl</code>	11926
<code>\c_fourteen</code>	67 , 2509 , 2541 , 3829 , 3839 , 7946	<code>\c_fp_exp_9_tl</code>	11926
<code>\c_fp_exp_100_tl</code>	11946	<code>\c_fp_ln_10_1_tl</code>	12272
<code>\c_fp_exp_10_tl</code>	11946	<code>\c_fp_ln_10_2_tl</code>	12272
<code>\c_fp_exp_1_tl</code>	11946	<code>\c_fp_ln_10_3_tl</code>	12272
<code>\c_fp_exp_200_tl</code>	11946	<code>\c_fp_ln_10_4_tl</code>	12272
<code>\c_fp_exp_20_tl</code>	11946	<code>\c_fp_ln_10_5_tl</code>	12272
<code>\c_fp_exp_2_tl</code>	11946	<code>\c_fp_ln_10_6_tl</code>	12272
<code>\c_fp_exp_30_tl</code>	11946	<code>\c_fp_ln_10_7_tl</code>	12272
<code>\c_fp_exp_3_tl</code>	11946	<code>\c_fp_ln_10_8_tl</code>	12272
<code>\c_fp_exp_40_tl</code>	11946	<code>\c_fp_ln_10_9_tl</code>	12272
<code>\c_fp_exp_4_tl</code>	11946	<code>\c_fp_ln_2_1_tl</code>	12281
<code>\c_fp_exp_50_tl</code>	11946	<code>\c_fp_ln_2_2_tl</code>	12281
<code>\c_fp_exp_5_tl</code>	11946	<code>\c_fp_ln_2_3_tl</code>	12281
<code>\c_fp_exp_60_tl</code>	11946	<code>\c_fp_pi_by_four_decimal_int</code>	
<code>\c_fp_exp_6_tl</code>	11946	9772 , 9772 , 9773 , 11431 , 11442 , 11454 , 11461 , 11465
<code>\c_fp_exp_70_tl</code>	11946	<code>\c_fp_pi_by_four_extended_int</code>	9772 , 9774 , 9775 , 11431 , 11443 , 11454 , 11466
<code>\c_fp_exp_7_tl</code>	11946	<code>\c_fp_pi_decimal_int</code>	
<code>\c_fp_exp_80_tl</code>	11946	9772 , 9776 , 9777 , 11371
<code>\c_fp_exp_8_tl</code>	11946	<code>\c_fp_pi_extended_int</code> ..	9772 , 9778 , 9779
<code>\c_fp_exp_90_tl</code>	11946	<code>\c_fp_two_pi_decimal_int</code>	
<code>\c_fp_exp_9_tl</code>	11946	9772 , 9780 , 9781 , 11367 , 11373
<code>\c_fp_exp_100_tl</code>	11926	<code>\c_fp_two_pi_extended_int</code>	
<code>\c_fp_exp_10_tl</code>	11926	9772 , 9782 , 9783 , 11367 , 11373
<code>\c_fp_exp_1_tl</code>	11926	<code>\c_group_begin_token</code>	
<code>\c_fp_exp_200_tl</code>	11926	50 , 2569 , 2569 , 2590 , 2931 , 3081 , 4690 , 4724 , 6445 , 6491
<code>\c_fp_exp_20_tl</code>	11926	<code>\c_group_end_token</code> ..	50 , 2569 , 2570 , 2595 , 2932 , 3082 , 6450 , 6451 , 6496 , 6497
<code>\c_fp_exp_2_tl</code>	11926	<code>\c_initex_cctab</code>	174 , 13228 , 13237
		<code>\c_int_from_roman_C_int</code>	3767
		<code>\c_int_from_roman_c_int</code>	3767
		<code>\c_int_from_roman_D_int</code>	3767
		<code>\c_int_from_roman_d_int</code>	3767
		<code>\c_int_from_roman_I_int</code>	3767
		<code>\c_int_from_roman_i_int</code>	3767

- \c_int_from_roman_L_int 3767
- \c_int_from_roman_l_int 3767
- \c_int_from_roman_M_int 3767
- \c_int_from_roman_m_int 3767
- \c_int_from_roman_V_int 3767
- \c_int_from_roman_v_int 3767
- \c_int_from_roman_X_int 3767
- \c_int_from_roman_x_int 3767
- \c_ior_log_stream 7926, 7929
- \c_ior_streams_tl 7930, 7949, 7986
- \c_ior_term_stream 7926, 7927
- \c_iow_log_stream 7926, 7928, 8186, 8187
- \c_iow_streams_tl 7930, 7930, 7949, 7999
- \c_iow_term_stream 7926, 7926, 8188, 8189
- \c_iow_wrap_end_marker_tl ... 8213, 8272
- \c_iow_wrap_indent_marker_tl 8213, 8236
- \c_iow_wrap_marker_tl
 - 8213, 8215, 8222, 8282, 8329
- \c_iow_wrap_newline_marker_tl 8213, 8257
- \c_iow_wrap_unindent_marker_tl
 - 8213, 8238
- \c_job_name_tl 93, 4771, 4782
- \c_keys_code_root_tl 9108, 9108, 9280,
 - 9285, 9549, 9551, 9563, 9569, 9574
- \c_keys_props_root_tl
 - 9110, 9110, 9143, 9173,
 - 9180, 9350, 9352, 9354, 9356, 9358,
 - 9360, 9362, 9364, 9366, 9368, 9370,
 - 9372, 9374, 9376, 9378, 9380, 9382,
 - 9384, 9386, 9388, 9390, 9392, 9394,
 - 9396, 9398, 9400, 9402, 9404, 9406,
 - 9408, 9410, 9412, 9414, 9416, 9418,
 - 9420, 9422, 9424, 9426, 9428, 9430,
 - 9432, 9434, 9436, 9438, 9440, 9442
- \c_keys_value_forbidden_tl .. 9111, 9111
- \c_keys_value_required_tl ... 9111, 9112
- \c_keys_vars_root_tl 9108, 9109, 9243,
 - 9262, 9269, 9272, 9274, 9289–9291,
 - 9294, 9337, 9510, 9512, 9515, 9523
- \c_letter_token 3122, 3125
- \c_luatex_is_engine_bool 1504, 1504
- \c_math_shift_token 3122, 3124
- \c_math_subscript_token
 - 50, 2569, 2579, 2623
- \c_math_superscript_token
 - 50, 2569, 2577, 2618
- \c_math_toggle_token
 - 50, 2569, 2573, 2600, 3124
- \c_max_dim 74, 4050, 4052,
 - 4053, 4057, 4132, 7501–7504, 7517
- \c_max_int 67, 3847, 3847
- \c_max_register_int
 - 67, 1162, 1162, 3280, 13164
- \c_max_skip 77, 4131, 4132
- \c_minus_one 67, 1150, 1151, 1154, 1155,
 - 1164, 3278, 3322, 3829, 4321, 4322,
 - 4348, 4349, 4361, 4362, 7928, 7929,
 - 8104, 8117, 8214, 8250, 9842, 9950,
 - 10379, 10633, 10634, 10771, 10806,
 - 11049, 11097, 11101, 11266, 11390,
 - 11394, 11659, 11674, 11762, 11766,
 - 11839, 11847, 12255, 12259, 12383
- \c_msg_coding_error_text_tl
 - 7883, 7894, 8450, 8450,
 - 8861, 8870, 8892, 8900, 8909, 8916,
 - 8923, 8930, 9587, 9594, 9615, 9622
- \c_msg_continue_text_tl 8450, 8455, 8518
- \c_msg_critical_text_tl 8450, 8457, 8637
- \c_msg_fatal_text_tl 8450, 8459, 8626, 8768
- \c_msg_help_text_tl 8450, 8461, 8526
- \c_msg_hide_tl 8491, 8493–8495, 8563
- \c_msg_hide_tl<dots> 8491
- \c_msg_kernel_bug_more_text_tl
 - 8934, 8941, 8945
- \c_msg_kernel_bug_text_tl 8934, 8936, 8943
- \c_msg_more_text_prefix_tl 8419, 8420,
 - 8436, 8445, 8642, 8650, 8781, 8791
- \c_msg_no_info_text_tl . 8450, 8463, 8516
- \c_msg_on_line_text_tl 8468, 8487
- \c_msg_on_line_tl 8450
- \c_msg_return_text_tl 143, 8450, 8466,
 - 8469, 8865, 8873, 8880, 8887, 8949
- \c_msg_text_prefix_tl
 - 8419, 8419, 8423, 8434, 8443, 8623,
 - 8634, 8647, 8656, 8667, 8675, 8681,
 - 8711, 8764, 8786, 8799, 8822, 8844
- \c_msg_trouble_text_tl . 143, 8450, 8476
- \c_nine 67, 2499, 2531, 3829,
 - 3835, 7941, 9955, 11303, 11535,
 - 11621, 11867, 11876, 12095, 12387
- \c_one 67, 2483, 2515, 3320,
 - 3829, 3829, 7933, 9844, 9855, 9914,
 - 9926, 9974, 9980, 10022, 10047,
 - 10244, 10603, 10607, 10609, 10616,
 - 10756, 10785, 11045, 11082, 11087,
 - 11108, 11231, 11299, 11342, 11356,
 - 11407, 11422, 11457, 11469, 11530,
 - 11616, 11664, 11671, 11686, 11712,
 - 11734, 11739, 11747, 11753, 11841,
 - 11845, 12047, 12057, 12158, 12183,

- 12194, 12198, 12220, 12240, 12246,
12350, 12354, 12385, 12402, 12454,
12477, 12483, 12484, 12528, 12546,
12584, 12605, 12611, 12765, 12767
- \c_one_fp . 170, 6536, 6670, 6672, 6715,
6735, 6753, 6755, 9785, 9785, 12664
- \c_one_hundred
... 67, 3844, 3844, 9977, 9978, 12056
- \c_one_hundred_million
..... 9767, 9769, 10966, 11898
- \c_one_million 9767, 9768, 11229
- \c_one_thousand 67,
3844, 3845, 10876, 11132, 11176, 11227
- \c_one_thousand_million
..... 9767, 9771, 9917,
9939, 9956, 9966, 10002, 10013,
10027, 10038, 10079, 10121, 10395,
10414, 10533, 10569, 10595, 10646,
10671, 10737, 10754, 10757, 10772,
10781, 10858, 10897, 10905, 10914,
11001, 11014, 11050, 11080, 11083,
11085, 11088, 11098, 11102, 11109,
11110, 11230, 11232, 11244, 11256,
11271, 11304, 11315, 11333, 11391,
11395, 11411, 11415, 11499, 11554,
11596, 11640, 11691, 11697, 11745,
11749, 11751, 11755, 11763, 11767,
11797, 11921, 11991, 12166, 12176,
12195, 12213, 12238, 12242, 12244,
12248, 12256, 12260, 12330, 12351,
12373, 12425, 12492, 12495, 12564,
12603, 12607, 12609, 12613, 12757
- \c_other_cctab
.... 174, 13228, 13238, 13246, 13256
- \c_other_char_token 3122, 3126
- \c_parameter_token
..... 50, 2569, 2576, 2609, 2612
- \c_pdftex_is_engine_bool 1504, 1505
- \c_pi_fp 170, 6582, 7415, 9786, 9786
- \c_seven
67, 1150, 1160, 2495, 2527, 3829, 7939
- \c_six 67, 1150, 1159,
2493, 2525, 3829, 7938, 11367, 11373
- \c_sixteen 67, 1150, 1157,
1166, 3764, 3829, 7926, 7927, 7961,
7964, 7985, 7987, 7998, 8000, 8033,
8052, 8070, 8089, 8106, 8119, 8381
- \c_space_tl 93, 4784, 4784, 4846,
5311, 5865, 6228, 6230, 7875, 7876,
8145, 8146, 8273, 8345, 8488, 9735
- \c_space_token 50, 2569, 2580,
2628, 2933, 2952, 3083, 4691, 4725
- \c_string_cctab
.... 174, 13228, 13239, 13247, 13261
- \c_ten 67,
2501, 2533, 3604, 3829, 3836, 7942,
9967, 10014, 10039, 10191, 10255,
10311, 10413, 10418, 10530, 10566,
10605, 10608, 10618, 11013, 11067,
11243, 11292, 11334, 11346, 11424,
11826, 12447, 12680, 12714, 12719
- \c_ten_thousand 67, 3844, 3846
- \c_thirteen 67, 2507, 2539, 3829, 3838, 7945
- \c_thirty_two 67, 3841, 3841
- \c_three 67, 2487, 2519,
3829, 3831, 7935, 11365, 11683, 12016
- \c_tl_act_lowercase_tl . 4911, 4916, 4924
- \c_tl_act_uppercase_tl . 4911, 4911, 4922
- \c_tl_rescan_marker_tl
..... 4302, 4310, 4320, 4332, 4360
- \c_token_A_int 2794, 2829
- \c_true_bool 20,
986, 1015, 1055, 1055, 1088, 1437,
1448, 1458, 1878, 1882, 1983, 1986,
1992, 1993, 2021, 2022, 2025, 2027,
2031, 2054, 3679, 3684, 3697, 4744
- \c_twelve 67, 1150, 1161, 2265,
2279, 2505, 2537, 2706, 3829, 7944
- \c_two 67, 2485, 2517, 3762,
3829, 3830, 7934, 10382, 11370,
11658, 11662, 11681, 11715, 11838,
11844, 12497–12499, 12512, 12588
- \c_two_hundred_fifty_five 67, 3842, 3842
- \c_two_hundred_fifty_six . 67, 3842, 3843
- \c_undefined:D 1277, 1285
- \c_undefined_fp 170, 9787, 9787, 10944,
11854, 12647, 12697, 12704, 12824
- \c_xetex_is_engine_bool 1504, 1506
- \c_zero 67, 985, 993,
1001, 1008, 1014, 1022, 1030, 1037,
1150, 1158, 1465, 1470, 1562, 1571,
2094, 2185–2195, 2198, 2201, 2259,
2261, 2408, 2415, 2446, 2455, 2481,
2513, 2677, 3209, 3239, 3243, 3244,
3250, 3297, 3298, 3577, 3829, 4101,
4111, 4112, 4756, 4793, 4838, 5400,
5413, 5904, 5919, 5939, 7932, 7961,
7964, 8403, 8405, 8967, 9864, 9875,
9913, 9925, 9927, 9938, 9981–9983,
10085, 10127, 10170, 10173, 10235,

- 10302, 10437–10445, 10476–10484,
 10539, 10575, 10619, 10674, 10712,
 10728, 10770, 10774, 10776, 10842,
 10846, 10871, 10940, 10950, 10963,
 10964, 10985, 10989, 11010, 11019,
 11020, 11048, 11096, 11100, 11104,
 11105, 11124, 11168, 11242, 11270,
 11281, 11289, 11347, 11350, 11357–
 11359, 11389, 11393, 11397, 11402,
 11425, 11426, 11431, 11450, 11454,
 11465, 11490, 11531, 11545, 11587,
 11617, 11631, 11657, 11670, 11672,
 11684, 11685, 11687, 11688, 11690,
 11692, 11695, 11706–11708, 11740,
 11741, 11761, 11765, 11788, 11837,
 11851, 11852, 11882, 11883, 11895,
 11896, 11912, 11979, 11982, 12018,
 12044, 12048–12050, 12058–12060,
 12072, 12105, 12123, 12124, 12156,
 12157, 12162, 12163, 12168, 12197,
 12233, 12234, 12254, 12258, 12297,
 12301, 12353, 12364, 12381, 12391–
 12394, 12410, 12436, 12444, 12458,
 12459, 12461, 12464, 12510, 12511,
 12514, 12519, 12521, 12533, 12550,
 12557, 12563, 12573, 12581, 12596,
 12639, 12643, 12660, 12675, 12679,
 12686, 12711, 12716, 12718, 12768,
 12769, 12797, 12815–12818, 12905,
 12908, 12912, 12923, 12924, 12946
 \c_zero_dim 74, 3913, 4050,
 4051, 4056, 4131, 6473, 6683, 6685,
 6686, 7147, 7150, 7153, 7162, 7165,
 7168, 7177, 7184, 7250, 7255, 7262
 \c_zero_fp 170, 6534, 6550, 6552, 6557,
 6766, 6775, 6790, 7563, 7578, 7581,
 9788, 9788, 10053, 10063, 10065,
 10954, 11830, 11885, 12008, 12035,
 12075, 12307, 12315, 12653, 12832
 \c_zero_muskip 4147
 \c_zero_skip
 77, 4072, 4131, 4131, 4182, 4183, 6460
 \catcode 3–
 6, 9, 70–78, 84–91, 101, 281–288, 665
 \catcodetable 755
 \CatcodeTableIniTeX 13237
 \CatcodeTableLaTeX 13236
 \CatcodeTableOther 13238
 \CatcodeTableString 13239
 \cctab_begin:N
 173, 13188, 13188, 13208, 13214
 \cctab_end 173
 \cctab_end: ... 13188, 13198, 13209, 13215
 \cctab_gset:Nn 173, 13219, 13219,
 13227, 13231, 13248, 13256, 13261
 \cctab_new:N 173, 13158, 13158,
 13177, 13181, 13230, 13245–13247
 \char 519
 \char_active_gset:Npn 57, 3072
 \char_active_gset:Npx 3073
 \char_active_gset_eq:NN . 58, 3058, 3075
 \char_active_set:Npn 57, 3058, 3070
 \char_active_set:Npx 3058, 3071
 \char_active_set_eq:NN .. 58, 3058, 3074
 \char_make_active:N 3131, 3147
 \char_make_active:n 3131, 3165
 \char_make_alignment:N 3131
 \char_make_alignment:n 3131
 \char_make_alignment_tab:N 3136
 \char_make_alignment_tab:n 3154
 \char_make_begin_group:N 3133
 \char_make_begin_group:n 3151
 \char_make_comment:N 3131, 3148
 \char_make_comment:n 3131, 3166
 \char_make_end_group:N 3134
 \char_make_end_group:n 3152
 \char_make_end_line:N 3131, 3137
 \char_make_end_line:n 3131, 3155
 \char_make_escape:N 3131, 3132
 \char_make_escape:n 3131, 3150
 \char_make_group_begin:N 3131
 \char_make_group_begin:n 3131
 \char_make_group_end:N 3131
 \char_make_group_end:n 3131
 \char_make_ignore:N 3131, 3143
 \char_make_ignore:n 3131, 3161
 \char_make_invalid:N 3131, 3149
 \char_make_invalid:n 3131, 3167
 \char_make_letter:N 3131, 3145
 \char_make_letter:n 3131, 3163
 \char_make_math_shift:N 3135
 \char_make_math_shift:n 3153
 \char_make_math_subscript:N . 3131, 3141
 \char_make_math_subscript:n . 3131, 3159
 \char_make_math_superscript:N 3131, 3139
 \char_make_math_superscript:n 3131, 3157
 \char_make_math_toggle:N 3131
 \char_make_math_toggle:n 3131
 \char_make_other:N 3131, 3146

- \char_make_other:n 3131, 3164
- \char_make_parameter:N 3131, 3138
- \char_make_parameter:n 3131, 3156
- \char_make_space:N 3131, 3144
- \char_make_space:n 3131, 3162
- \char_set_catcode:nn 48, 298–306, 2474, 2474, 2481, 2483, 2485, 2487, 2489, 2491, 2493, 2495, 2497, 2499, 2501, 2503, 2505, 2507, 2509, 2511, 2513, 2515, 2517, 2519, 2521, 2523, 2525, 2527, 2529, 2531, 2533, 2535, 2537, 2539, 2541, 2543, 2706
- \char_set_catcode:w 3094, 3095, 3102, 3104
- \char_set_catcode_active:N 47, 2480, 2506, 2585, 3059, 3147, 8550
- \char_set_catcode_active:n . 47, 2512, 2538, 3064, 3165, 9007, 9008, 13254
- \char_set_catcode_alignment:N 47, 2480, 2488, 2574, 3136
- \char_set_catcode_alignment:n 47, 316, 2512, 2520, 3154
- \char_set_catcode_comment:N 47, 2480, 2508, 3148
- \char_set_catcode_comment:n 47, 2512, 2540, 3166
- \char_set_catcode_end_line:N 47, 2480, 2490, 3137
- \char_set_catcode_end_line:n 47, 2512, 2522, 3155
- \char_set_catcode_escape:N 47, 2480, 2480, 3132
- \char_set_catcode_escape:n 47, 2512, 2512, 3150
- \char_set_catcode_group_begin:N 47, 2480, 2482, 3133
- \char_set_catcode_group_begin:n 47, 2512, 2514, 3151
- \char_set_catcode_group_end:N 47, 2480, 2484, 3134
- \char_set_catcode_group_end:n 47, 2512, 2516, 3152
- \char_set_catcode_ignore:N 47, 2480, 2498, 3143
- \char_set_catcode_ignore:n 47, 313, 314, 2512, 2530, 3161, 9853
- \char_set_catcode_invalid:N 47, 2480, 2510, 3149
- \char_set_catcode_invalid:n 47, 2512, 2542, 3167
- \char_set_catcode_letter:N 47, 2480, 2502, 3145, 8491
- \char_set_catcode_letter:n 47, 317, 319, 2512, 2534, 3163
- \char_set_catcode_math_subscript:N . 47, 2480, 2496, 2578, 3142
- \char_set_catcode_math_subscript:n . 47, 2512, 2528, 3160, 13253
- \char_set_catcode_math_superscript:N 47, 2480, 2494, 3140, 8952
- \char_set_catcode_math_superscript:n 47, 318, 2512, 2526, 3158
- \char_set_catcode_math_toggle:N 47, 2480, 2486, 2572, 3135
- \char_set_catcode_math_toggle:n 47, 2512, 2518, 3153
- \char_set_catcode_other:N 47, 2480, 2504, 2662, 2663, 2796, 3146, 8209, 8496
- \char_set_catcode_other:n 47, 315, 320, 2512, 2536, 3164, 13252, 13259, 13264
- \char_set_catcode_parameter:N 47, 2480, 2492, 3138
- \char_set_catcode_parameter:n 47, 2512, 2524, 3156
- \char_set_catcode_space:N 47, 2480, 2500, 3144
- \char_set_catcode_space:n . 47, 321, 2512, 2532, 3162, 13250, 13251, 13265
- \char_set_lccode:nn 48, 2544, 2550, 2664–2666, 2699–2704, 2797–2799, 3066, 8210, 8548, 8549, 8953–8956, 9009, 9010
- \char_set_lccode:w 3094, 3097, 3108, 3110
- \char_set_mathcode:nn ... 49, 2544, 2544
- \char_set_mathcode:w 3094, 3096, 3105, 3107
- \char_set_sfcode:nn 49, 2544, 2562
- \char_set_sfcode:w 3094, 3099, 3114, 3116
- \char_set_uccode:nn 49, 2544, 2556
- \char_set_uccode:w 3094, 3098, 3111, 3113
- \char_show_value_catcode:n 48, 2474, 2478
- \char_show_value_catcode:w . 3101, 3103
- \char_show_value_lccode:n 48, 2544, 2554
- \char_show_value_lccode:w ... 3101, 3109
- \char_show_value_mathcode:n 49, 2544, 2548
- \char_show_value_mathcode:w . 3101, 3106
- \char_show_value_sfcode:n 50, 2544, 2566
- \char_show_value_sfcode:w ... 3101, 3115
- \char_show_value_uccode:n 49, 2544, 2560

\char_show_value_uccode:w . . .	3101 , 3112	\clist_gpush:No	5671 , 5681
\char_tmp:NN	3060 , 3070 – 3075	\clist_gpush:Nv	5671 , 5680
\char_value_catcode:n	48 , 298 – 306 , 2474 , 2476	\clist_gpush:Nx	5671 , 5682
\char_value_catcode:w	3101 , 3102	\clist_gput_left:cn	5622 , 5683
\char_value_lccode:n	48 , 2544 , 2552	\clist_gput_left:co	5622 , 5685
\char_value_lccode:w	3101 , 3108	\clist_gput_left:cV	5622 , 5684
\char_value_mathcode:n . . .	49 , 2544 , 2546	\clist_gput_left:cx	5622 , 5686
\char_value_mathcode:w	3101 , 3105	\clist_gput_left:Nn	107 , 5622 , 5624 , 5638 , 5639 , 5679
\char_value_sfcode:n	49 , 2544 , 2564	\clist_gput_left:No	5622 , 5681
\char_value_sfcode:w	3101 , 3114	\clist_gput_left:Nv	5622 , 5680
\char_value_uccode:n	49 , 2544 , 2558	\clist_gput_left:Nx	5622 , 5682
\char_value_uccode:w	3101 , 3111	\clist_gput_right:cn	5640
\chardef	80 , 93 , 96 , 328 , 354	\clist_gput_right:co	5640
\chk_if_exist_cs:c	1208 , 1216	\clist_gput_right:cV	5640
\chk_if_exist_cs:N	23 , 1208 , 1208 , 1217 , 1766	\clist_gput_right:cx	5640
\chk_if_free_cs:c	1185 , 1206	\clist_gput_right:Nn	107 , 5640 , 5642 , 5646 , 5647
\chk_if_free_cs:N	23 , 1185 , 1185 , 1195 , 1207 , 1222 , 1266 , 3271 , 3286 , 3908 , 4067 , 4141 , 4200 , 4206 , 4211 , 6326 , 7976	\clist_gput_right:No	5640
\cleaders	537	\clist_gput_right:Nv	5640
\clist_clear:c	5543 , 5544	\clist_gput_right:Nx	5640
\clist_clear:N	105 , 5543 , 5543 , 5696 , 5955 , 9459	\clist_gremove_all:cn	5706
\clist_clear_new:c	5547 , 5548	\clist_gremove_all:Nn	108 , 5706 , 5708 , 5736 , 5985
\clist_clear_new:N	105 , 5547 , 5547	\clist_gremove_duplicates:c	5690
\clist_concat:ccc	5559	\clist_gremove_duplicates:N	108 , 5690 , 5692 , 5705
\clist_concat:NNN	106 , 5559 , 5559 , 5572	\clist_gremove_element:Nn	5983 , 5985
\clist_concat_aux:NNNN	5559 , 5560 , 5562 , 5563	\clist_gset:cn	5616
\clist_display:c	5987 , 5989	\clist_gset:co	5616
\clist_display:N	5987 , 5988	\clist_gset:cV	5616
\clist_gclear:c	5543 , 5546	\clist_gset:cx	5616
\clist_gclear:N	105 , 5543 , 5545 , 5957	\clist_gset:Nn	106 , 5616 , 5618 , 5621
\clist_gclear_new:c	5547 , 5550	\clist_gset:No	5616 , 5992
\clist_gclear_new:N	106 , 5547 , 5549	\clist_gset:Nv	5616
\clist_gconcat:ccc	5559	\clist_gset:Nx	5616
\clist_gconcat:NNN	106 , 5559 , 5561 , 5573	\clist_gset_eq:cc	5551 , 5558
\clist_get:cN	5648 , 5981	\clist_gset_eq:cN	5551 , 5557
\clist_get:NN	111 , 5648 , 5648 , 5652 , 5980	\clist_gset_eq:Nc	5551 , 5556
\clist_get_aux:wN	5648 , 5649 , 5650	\clist_gset_eq:NN	106 , 5551 , 5555 , 5693
\clist_gpop:cN	5653	\clist_gset_from_seq:cc	5954
\clist_gpop:NN	112 , 5653 , 5655 , 5670	\clist_gset_from_seq:cN	5954
\clist_gpush:cn	5671 , 5683	\clist_gset_from_seq:Nc	5954
\clist_gpush:co	5671 , 5685	\clist_gset_from_seq:NN	113 , 5954 , 5956 , 5977 , 5978
\clist_gpush:cV	5671 , 5684	\clist_gtrim_spaces:c	5991
\clist_gpush:cx	5671 , 5686	\clist_gtrim_spaces:N	5991 , 5992 , 5994
\clist_gpush:Nn	112 , 5671 , 5679	\clist_if_empty:c	5738 , 5739
		\clist_if_empty:N	5738 , 5738
		\clist_if_empty:Nf	5568 , 5723 , 5771 , 5808

\clist_if_empty:NTF	108 , 5629 , 5846	\clist_map_function_n_aux:Nn	
\clist_if_eq:cc	5740 , 5743		5830 , 5832 , 5835 , 5839
\clist_if_eq:cN	5740 , 5742	\clist_map_inline:cn	5785
\clist_if_eq:Nc	5740 , 5741	\clist_map_inline:Nn	
\clist_if_eq:NN	5740 , 5740		109 , 5697 , 5785 , 5785 , 5805
\clist_if_eq:NNTF	109	\clist_map_inline:nn	5785 , 5795 , 9228 , 9322	
\clist_if_in:cn	5744	\clist_map_variable:cNn	5806
\clist_if_in:co	5744	\clist_map_variable:NNn	
\clist_if_in:cV	5744		110 , 5806 , 5806 , 5826 , 5828
\clist_if_in:Nn	5744 , 5744	\clist_map_variable:nNn	5806 , 5823
\clist_if_in:nn	5744 , 5748	\clist_map_variable_aux:Nnw	
\clist_if_in:NnF	5699 , 5762 , 5763		5806 , 5811 , 5816 , 5821
\clist_if_in:nnF	5767	\clist_new:c	5541 , 5542
\clist_if_in:NnT	5760 , 5761	\clist_new:N	
\clist_if_in:nnT	5766			105 , 5541 , 5541 , 5689 , 5737 , 5870 – 5873
\clist_if_in:NnTF	109 , 5764 , 5765	\clist_pop:cn	5653
\clist_if_in:nnTF	5768	\clist_pop:NN	111 , 5653 , 5653 , 5669
\clist_if_in:No	5744	\clist_pop_aux:NNN	5653 , 5654 , 5656 , 5657	
\clist_if_in:no	5744	\clist_pop_aux:NwNNN	5653
\clist_if_in:NV	5744	\clist_pop_aux:w	5666 , 5668
\clist_if_in:nV	5744	\clist_pop_aux:wNN	5653
\clist_if_in_return:nn		\clist_pop_aux:wNNN	5659 , 5661
	5744 , 5746 , 5751 , 5753	\clist_push:cn	5671 , 5675
\clist_item:cn	5894	\clist_push:co	5671 , 5677
\clist_item:Nn	113 , 5894 , 5894 , 5923	\clist_push:cV	5671 , 5676
\clist_item:nn	5924 , 5924	\clist_push:cx	5671 , 5678
\clist_item_aux:nnNn	5894 , 5896 , 5902 , 5926		\clist_push:Nn	112 , 5671 , 5671
\clist_item_n_aux:nw	... 5924 , 5929 , 5932		\clist_push:No	5671 , 5673
\clist_item_n_end:n 5924 , 5940 , 5948		\clist_push:NV	5671 , 5672
\clist_item_N_loop:nw		\clist_push:Nx	5671 , 5674
	5894 , 5899 , 5917 , 5921	\clist_put_aux:NNnnNn	
\clist_item_n_loop:nw	5623 , 5625 , 5626 , 5641 , 5643
	5924 , 5933 , 5934 , 5937 , 5942	\clist_put_left:cn	5622 , 5675
\clist_item_n_strip:w	.. 5924 , 5950 , 5953		\clist_put_left:co	5622 , 5677
\clist_length:c	5874	\clist_put_left:cV	5622 , 5676
\clist_length:N	112 , 5874 , 5874 , 5893 , 5897		\clist_put_left:cx	5622 , 5678
\clist_length:n	5874 , 5883 , 5927	\clist_put_left:Nn	
\clist_length_aux:n 5874 , 5879 , 5882			...	107 , 5622 , 5622 , 5636 , 5637 , 5671
\clist_length_n_aux:n	5888 , 5892	\clist_put_left:No	5622 , 5673
\clist_map_aux_unbrace:Nw	5829 , 5829 , 5838		\clist_put_left:NV	5622 , 5672
\clist_map_break	110	\clist_put_left:Nx	5622 , 5674
\clist_map_break:	5842 , 5842	\clist_put_right:cn	5640
\clist_map_break:n	111 , 5842 , 5843	\clist_put_right:co	5640
\clist_map_function:cN	5769	\clist_put_right:cV	5640
\clist_map_function:NN	109 , 5449 , 5459 , 5769 , 5769 , 5783 , 5791 , 5858 , 5879		\clist_put_right:cx	5640
\clist_map_function:nN	.. 5454 , 5464 , 5769 , 5801 , 5830 , 5830 , 9247 , 9307		\clist_put_right:Nn	
\clist_map_function_aux:Nw	107 , 5640 , 5640 , 5644 , 5645 , 5700
	5769 , 5773 , 5777 , 5781 , 5888	\clist_put_right:No	5640
			\clist_put_right:NV	5640
			\clist_put_right:Nx	5640 , 9540

\clist_remove_all:cn	5706	\clist_trim_spaces_generic:nw	5574 , 5576 , 5601 , 5832 , 5839
\clist_remove_all:Nn	108 , 5706 , 5706 , 5735 , 5984	\clist_trim_spaces_generic_aux:w ...	5574 , 5585 , 5591
\clist_remove_all_aux:	5706 , 5716 , 5720 , 5732	\clist_trim_spaces_generic_aux_ii:nn	5574 , 5592 , 5593
\clist_remove_all_aux:NNn	5706 , 5707 , 5709 , 5710	\clist_use:c	5687 , 5688
\clist_remove_all_aux:w	5706 , 5733 , 5734	\clist_use:N	107 , 5687 , 5687
\clist_remove_duplicates:c	5690	\closein	413
\clist_remove_duplicates:N	108 , 5690 , 5690 , 5704	\closeout	408
\clist_remove_duplicates_aux:NN	5690 , 5691 , 5693 , 5694	\clubpenalties	721
\clist_remove_element:Nn	5983 , 5984	\clubpenalty	548
\clist_set:cn	5616	\coffin_align:NnnNnnnnN	7246 , 7283 , 7301 , 7309 , 7309 , 7399
\clist_set:co	5616	\coffin_attach:cnncnnnn	7281
\clist_set:cV	5616	\coffin_attach:cnNnnnnn	7281
\clist_set:cx	5616	\coffin_attach:Nnncnnnn	7281
\clist_set:Nn	106 , 5616 , 5616 , 5620 , 5750	\coffin_attach:NnnNnnnn	133 , 7281 , 7281 , 7308
\clist_set:No	5616 , 5991	\coffin_attach_mark:NnnNnnnn	7281 , 7299 , 7692 , 7713 , 7729
\clist_set:NV	5616	\coffin_calculate_intersection:Nnn	7129 , 7129 , 7311 , 7314 , 7824
\clist_set:Nx	5616	\coffin_calculate_intersection:nnnnnnnn	7129 , 7135 , 7144 , 7770
\clist_set_eq:cc	5551 , 5554	\coffin_calculate_intersection_aux:nnnnnnN	7129 , 7156 , 7171 , 7180 , 7187 , 7221 , 7230
\clist_set_eq:cN	5551 , 5553	\coffin_clear:c	6876
\clist_set_eq:Nc	5551 , 5552	\coffin_clear:N ...	131 , 6876 , 6876 , 6884
\clist_set_eq:NN	106 , 5551 , 5551 , 5691 , 9464	\coffin_display_attach:Nnnnn	7735 , 7775 , 7797 , 7816 , 7822
\clist_set_from_seq:cc	5954	\coffin_display_handles:cn ...	134 , 7735
\clist_set_from_seq:cN	5954	\coffin_display_handles:Nn	7735 , 7735 , 7821
\clist_set_from_seq:Nc	5954	\coffin_display_handles_aux:nnnn	7735 , 7803 , 7808 , 7814
\clist_set_from_seq:NN	113 , 5954 , 5954 , 5975 , 5976	\coffin_display_handles_aux:nnnnnn	7735 , 7761 , 7765
\clist_set_from_seq_aux:NNNN	5955 , 5957 , 5958	\coffin_end_user_dimensions:	7031 , 7046 , 7063 , 7076 , 7570
\clist_set_from_seq_aux:w ...	5966 , 5974	\coffin_find_bounding_shift:	7426 , 7515 , 7515
\clist_show:c	5844 , 5989	\coffin_find_bounding_shift_aux:nn	7515 , 7519 , 7521
\clist_show:N	112 , 5844 , 5844 , 5869 , 5988	\coffin_find_corner_maxima:N	7425 , 7499 , 7499
\clist_show_aux:n	5844 , 5858 , 5863	\coffin_find_corner_maxima_aux:nn	7499 , 7506 , 7508
\clist_show_aux:w	5844 , 5860 , 5868		
\clist_tmp:w	5540 , 5540 , 5574 , 5590 , 5712 , 5733 , 5755 , 5757		
\clist_top:cN	5979 , 5981		
\clist_top:NN	5979 , 5980		
\clist_trim_spaces:c	5991		
\clist_trim_spaces:N ...	5991 , 5991 , 5993		
\clist_trim_spaces:n	113 , 5594 , 5594 , 5617 , 5619 , 5628 , 5825		
\clist_trim_spaces_aux:n	5594 , 5596 , 5599 , 5609 , 5613		
\clist_trim_spaces_aux_ii:nn	5594 , 5602 , 5604		

\coffin_get_pole:NnN	\coffin_scale:cnn	7572
... 7000 , 7000 , 7131 , 7132 , 7364 ,	\coffin_scale:Nnn .	133 , 7572 , 7572 , 7587
7365 , 7368 , 7369 , 7749 , 7750 , 7753	\coffin_scale_corner:Nnnn	7560 , 7597 , 7597
\coffin_gset_eq_structure:NN	\coffin_scale_pole:Nnnnn	7562 , 7597 , 7603
\coffin_if_exist:NT	\coffin_scale_vector:nnNN	7588 , 7588 , 7599 , 7605
..... 6860 , 6860 , 6878 , 6898 ,	\coffin_set_bounding:N .	7422 , 7446 , 7446
6915 , 6942 , 6959 , 6989 , 7055 , 7068	\coffin_set_eq:cc	6987
\coffin_join:cnncnnnn	\coffin_set_eq:cN	6987
\coffin_join:cnnNnnnn	\coffin_set_eq:Nc	6987
\coffin_join:Nnncnnnn	\coffin_set_eq:NN	131 , 6987 ,
\coffin_join:NnnNnnnn 133 , 7244 , 7244 , 7280 6987 , 6995 , 7278 , 7297 , 7326 , 7756	
\coffin_mark_handle:cnnn	\coffin_set_eq_structure:NN	6992 , 7017 , 7017
\coffin_mark_handle:Nnnn	\coffin_set_horizontal_pole:cnn ..	7053
..... 134 , 7680 , 7680 , 7734	\coffin_set_horizontal_pole:Nnn ..	132 , 7053 , 7053 , 7081
\coffin_mark_handle_aux:nnnnNnn ..	\coffin_set_pole:Nnn ..	7053 , 7079 , 7083
..... 7680 , 7718 , 7723 , 7727	\coffin_set_pole:Nnx	6929 , 6972 , 7053 , 7058 , 7071 , 7340 ,
\coffin_new:c 7377 , 7381 , 7389 , 7393 , 7475 , 7606	
\coffin_new:N	\coffin_set_user_dimensions:N	7031 , 7031 , 7057 , 7070 , 7546 , 7575
..... 131 , 6885 , 6885 ,	\coffin_set_vertical_pole:cnn ..	7053
6895 , 6996 , 6998 , 6999 , 7627 - 7629	\coffin_set_vertical_pole:Nnn	132 , 7053 , 7066 , 7082
\coffin_offset_corner:Nnnnn 7441 , 7523 , 7523	
7350 , 7352	\coffin_shift_pole:Nnnnn	7443 , 7523 , 7531
\coffin_offset_corners:Nnn	\coffin_show_aux:n	7846
.. 7266 , 7267 , 7273 , 7274 , 7347 , 7347	\coffin_show_aux:nn	7863 , 7873
\coffin_offset_corners:Nnnnn ..	\coffin_show_aux:w	7846 , 7866 , 7878
7347	\coffin_show_structure:c	7846
\coffin_offset_pole:Nnnnnnn	\coffin_show_structure:N	134 , 7846 , 7846 , 7879
..... 7328 , 7331 , 7333	\coffin_typeset:cnnn	7397
\coffin_offset_poles:Nnn .	\coffin_typeset:Nnnnn 133 , 7397 , 7397 , 7404	
7264 , 7265 ,	\coffin_update_B:nnnnnnnnN	7362 , 7370 , 7385
7270 , 7271 , 7293 , 7294 , 7328 , 7328 6909 , 6927 , 6951 , 6970 , 7084 , 7084	
\coffin_reset_structure:N	\coffin_update_poles:N ..	6908 , 6926 ,
..... 6881 , 6907 , 6925 , 6950 , 6969 , 7095 , 7095 , 7261 , 7292	
6949 , 6968 , 7010 , 7010 , 7258 , 7288	\coffin_update_T:nnnnnnnnN	7362 , 7366 , 7373
\coffin_resize:cnn 7362 , 7366 , 7373	
7544	\coffin_update_vertical_poles:NNN ..	7277 , 7296 , 7362 , 7362
\coffin_resize:Nnn 7277 , 7296 , 7362 , 7362	
132 , 7544 , 7544 , 7556	\coffin_x_shift_corner:Nnnn	7566 , 7612 , 7612
\coffin_resize_common:Nnn 7566 , 7612 , 7612	
..... 7554 , 7557 , 7557 , 7584	\coffin_x_shift_pole:Nnnnn	7568 , 7612 , 7619
\coffin_rotate:cn		
7411		
\coffin_rotate:Nn .		
132 , 7411 , 7411 , 7445		
\coffin_rotate_bounding:nnn		
..... 7424 , 7458 , 7458		
\coffin_rotate_corner:Nnnn		
..... 7419 , 7458 , 7464		
\coffin_rotate_pole:Nnnnn		
..... 7421 , 7470 , 7470		
\coffin_rotate_vector:nnNN		
.. 7460 , 7466 , 7472 , 7473 , 7482 , 7482		
\coffin_saved_Depth: 6856 , 6856 , 7034 , 7049		
\coffin_saved_Height:		
..... 6856 , 6857 , 7033 , 7048		
\coffin_saved_TotalHeight:		
..... 6856 , 6858 , 7035 , 7050		
\coffin_saved_Width: 6856 , 6859 , 7036 , 7051		

- `\color` 7688, 7700, 7743, 7784
- `\color_ensure_current` 135
- `\color_ensure_current:` 6903,
6921, 6944, 6963, 7911, 7912, 7916
- `\color_group_begin` 135
- `\color_group_begin:`
.. 6902, 6920, 6944, 6963, 7905, 7905
- `\color_group_end` 135
- `\color_group_end:`
.. 6905, 6923, 6947, 6966, 7905, 7906
- `\copy` 605
- `\count` 656
- `\countdef` 355
- `\cr` 380
- `\crrcr` 381
- `\cs:w` 16, 803,
805, 818, 850, 1114, 1142, 1340,
1372, 1526, 1565, 1579, 1581, 1583,
1587–1589, 1624, 1630, 1650, 1652,
1657, 1664, 1665, 1723, 1763, 2164,
2166, 3337, 4571, 4959, 12145, 12505
- `\cs_end` 16
- `\cs_end:` 803, 806, 818,
850, 1108, 1114, 1136, 1142, 1280,
1340, 1372, 1526, 1565, 1579, 1581,
1583, 1587–1589, 1624, 1630, 1650,
1652, 1657, 1664, 1665, 1723, 1763,
2161, 2167–2170, 2172, 2174, 2176,
2178, 2180, 2182, 2184, 3337, 4570,
4571, 4959, 11512, 11600, 11810,
11995, 12005, 12145, 12334, 12505
- `\cs_generate_from_arg_count:cNnn` ...
.. 1012, 1020, 1028, 1036, 1328, 1371
- `\cs_generate_from_arg_count:NNnn` ...
..... 14, 1305, 1305, 1329, 1339
- `\cs_generate_from_arg_count_aux:nwn`
..... 1305, 1308–1317, 1326
- `\cs_generate_from_arg_count_error_msg:Nn`
..... 1305, 1319, 1330
- `\cs_generate_internal_variant:n` ...
..... 31, 1805, 1844, 1844
- `\cs_generate_internal_variant_aux:N`
..... 1844, 1849, 1852, 1858
- `\cs_generate_variant:Nn` 25, 1764, 1764,
1860–1867, 1876, 1885–1888, 1901,
1902, 1911–1914, 2060, 2061, 2066,
2067, 2132, 2155, 2421, 2422, 2439–
2442, 2461–2464, 3275, 3296, 3299,
3300, 3302, 3303, 3305, 3306, 3315–
3318, 3327–3330, 3334, 3335, 3709,
3912, 3915, 3916, 3920, 3921, 3923,
3924, 3926, 3927, 3936–3939, 3943,
3944, 3948, 3949, 4047, 4049, 4071,
4074, 4075, 4079, 4080, 4082, 4083,
4085, 4086, 4090, 4091, 4095, 4096,
4120, 4127, 4128, 4130, 4145, 4149,
4150, 4154, 4155, 4157, 4158, 4160,
4161, 4165, 4166, 4170, 4171, 4175,
4177, 4203, 4214, 4215, 4220, 4221,
4226, 4227, 4248–4253, 4270–4277,
4294–4301, 4336–4339, 4355, 4356,
4379–4382, 4423, 4424, 4429, 4430,
4433–4440, 4449–4452, 4461–4464,
4484–4487, 4506–4508, 4515–4517,
4530, 4551, 4562, 4566, 4587, 4588,
4640, 4641, 4649, 4650, 4678–4681,
4769, 4891, 4894, 4952, 4953, 4992,
5031, 5032, 5037–5040, 5045–5048,
5064, 5065, 5090, 5091, 5113–5118,
5127, 5143, 5144, 5168, 5195, 5196,
5221, 5250, 5261, 5262, 5315, 5338–
5343, 5372–5383, 5393, 5417, 5419,
5444, 5445, 5467–5472, 5572, 5573,
5620, 5621, 5636–5639, 5644–5647,
5652, 5669, 5670, 5704, 5705, 5735,
5736, 5760–5768, 5783, 5805, 5828,
5869, 5893, 5923, 5975–5978, 5993,
5994, 6012, 6048–6051, 6060, 6061,
6079–6082, 6097, 6099, 6101, 6103,
6118, 6119, 6128–6131, 6153–6160,
6172–6177, 6191, 6192, 6203, 6234,
6253–6258, 6272, 6286, 6296, 6297,
6303–6305, 6308, 6330, 6335, 6336,
6349, 6350, 6355, 6356, 6361, 6362,
6366–6368, 6375–6377, 6380, 6381,
6397–6404, 6407–6410, 6416, 6417,
6432, 6436, 6437, 6442, 6443, 6448,
6449, 6467, 6468, 6476, 6477, 6482,
6483, 6488, 6489, 6494, 6495, 6506,
6507, 6680, 6721, 6741, 6763, 6884,
6895, 6912, 6939, 6956, 6986, 6995,
7081–7083, 7280, 7308, 7404, 7445,
7556, 7587, 7734, 7821, 7879, 7972,
7973, 7980, 7981, 8008, 8009, 8126,
8127, 8179, 8182, 8614–8617, 8744,
9130, 9295, 9348, 9349, 9452, 9453,
9466, 9467, 10055, 10061, 10066,
10067, 10100, 10101, 10148, 10149,
10164, 10226, 10229, 10296, 10521,
10524, 10556, 10559, 10640, 10641,

- 10665, 10666, 10691, 10692, 10794,
 10795, 10812, 10813, 10925, 10926,
 11477, 11478, 11574, 11575, 11775,
 11776, 11968, 11969, 12271, 12286,
 12287, 12620, 12621, 13150, 13153
 \cs_generate_variant_aux:N
 1767, 1816, 1829
 \cs_generate_variant_aux:NNn
 1764, 1778, 1800
 \cs_generate_variant_aux:nnNNn
 1764, 1768, 1771
 \cs_generate_variant_aux:Nnnw
 1764, 1772, 1773, 1798
 \cs_generate_variant_aux:w
 1816, 1831, 1838
 \cs_get_arg_count_from_signature:c .
 1303, 1373
 \cs_get_arg_count_from_signature:N .
 19, 931,
 940, 947, 957, 1287, 1287, 1304, 1341
 \cs_get_arg_count_from_signature_aux:nnN
 1287, 1288, 1289
 \cs_get_arg_count_from_signature_auxii:w
 1287, 1297, 1302
 \cs_get_function_name:N . 19, 1090, 1090
 \cs_get_function_signature:N
 19, 1090, 1092
 \cs_gnew:cpn 1483
 \cs_gnew:cpx 1487
 \cs_gnew:Npn 1475
 \cs_gnew:Npx 1479
 \cs_gnew_eq:cc 1495
 \cs_gnew_eq:cN 1493
 \cs_gnew_eq:Nc 1494
 \cs_gnew_eq:NN 1492
 \cs_gnew_nopar:cpn 1482
 \cs_gnew_nopar:cpx 1486
 \cs_gnew_nopar:Npn 1474
 \cs_gnew_nopar:Npx 1478
 \cs_gnew_protected:cpn 1485
 \cs_gnew_protected:cpx 1489
 \cs_gnew_protected:Npn 1477
 \cs_gnew_protected:Npx 1481
 \cs_gnew_protected_nopar:cpn 1484
 \cs_gnew_protected_nopar:cpx 1488
 \cs_gnew_protected_nopar:Npn 1476
 \cs_gnew_protected_nopar:Npx 1480
 \cs_gset:cn 1376
 \cs_gset:cpn 1242, 1244, 4534,
 4544, 5788, 5798, 6197, 8443, 8445
 \cs_gset:cpx 1242, 1245
 \cs_gset:cx 1376
 \cs_gset:Nn 13, 1335
 \cs_gset:Npn
 . 11, 836, 838, 1228, 1244, 3072, 5225
 \cs_gset:Npx 836, 840, 1229, 1245, 3073, 5230
 \cs_gset:Nx 1335
 \cs_gset_eq:cc ... 1272, 1275, 1896, 4235
 \cs_gset_eq:cN
 ... 1272, 1274, 1285, 1895, 4233,
 5234, 6009, 8037, 8074, 9095, 9097
 \cs_gset_eq:Nc 1272, 1273, 1894,
 4234, 5241, 8029, 8045, 8066, 8082
 \cs_gset_eq:NN
 . 15, 1272, 1272–1275, 1277, 1882,
 1884, 1893, 3075, 4201, 4232, 6008
 \cs_gset_nopar:cn 1376
 \cs_gset_nopar:cpn 1234, 1238
 \cs_gset_nopar:cpx 1234, 1239, 2970
 \cs_gset_nopar:cx 1376
 \cs_gset_nopar:Nn 14, 1335
 \cs_gset_nopar:Npn 12, 836,
 836, 839, 843, 847, 1226, 1238, 2229
 \cs_gset_nopar:Npx 836, 837, 841,
 845, 849, 1227, 1239, 2235, 4207,
 4212, 4243, 4245, 4247, 4263, 4265,
 4267, 4269, 4287, 4289, 4291, 4293
 \cs_gset_nopar:Nx 1335
 \cs_gset_protected:cn 1376
 \cs_gset_protected:cpn 1254, 1256
 \cs_gset_protected:cpx 1254, 1257
 \cs_gset_protected:cx 1376
 \cs_gset_protected:Nn 14, 1335
 \cs_gset_protected:Npn
 12, 836, 846, 1232, 1256
 \cs_gset_protected:Npx 836, 848, 1233, 1257
 \cs_gset_protected:Nx 1335
 \cs_gset_protected_nopar:cn 1376
 \cs_gset_protected_nopar:cpn 1248, 1250
 \cs_gset_protected_nopar:cpx 1248, 1251
 \cs_gset_protected_nopar:cx 1376
 \cs_gset_protected_nopar:Nn ... 14, 1335
 \cs_gset_protected_nopar:Npn
 12, 836, 842, 1230, 1250
 \cs_gset_protected_nopar:Npx
 836, 844, 1231, 1251
 \cs_gset_protected_nopar:Nx 1335
 \cs_gundefine:c 1499
 \cs_gundefine:N 1498
 \cs_if_eq:cc 1400

<code>\cs_if_eq:ccF</code>	1416	1289, 1302, 1330, 1475, 1520–1525,
<code>\cs_if_eq:ccT</code>	1415	1527, 1529, 1541, 1547, 1566, 1573,
<code>\cs_if_eq:ccTF</code>	1414	1574, 1576, 1578, 1580, 1582, 1584,
<code>\cs_if_eq:cN</code>	<u>1400</u>	1591, 1593, 1598, 1603, 1609, 1615,
<code>\cs_if_eq:cNF</code>	1408	1621, 1627, 1640, 1647, 1654, 1661,
<code>\cs_if_eq:cNT</code>	1407	1695, 1696, 1701, 1703, 1708, 1713,
<code>\cs_if_eq:cNTF</code>	1406	1715, 1717, 1718, 1720, 1726, 1732,
<code>\cs_if_eq:Nc</code>	<u>1400</u>	1737, 1744, 1746, 1748, 1750, 1751,
<code>\cs_if_eq:NcF</code>	1412	1753, 1758, 1763, 1771, 1773, 1800,
<code>\cs_if_eq:NcT</code>	1411	1838, 1852, 1897, 1899, 1925, 1930,
<code>\cs_if_eq:NcTF</code>	1410	1940, 1950, 1951, 1978–1980, 1989,
<code>\cs_if_eq:NN</code>	<u>1400</u> , 1400	2004, 2009, 2033, 2039, 2045, 2049,
<code>\cs_if_eq:NNF</code>	1408, 1412, 1416	2050, 2056, 2058, 2062, 2064, 2068,
<code>\cs_if_eq:NNT</code>	1407, 1411, 1415	2076, 2081, 2089, 2094, 2095, 2100,
<code>\cs_if_eq:NNTF</code> ..	<u>21</u> , 1406, 1410, 1414, 8609	2102, 2108, 2113, 2115, 2121, 2126,
<code>\cs_if_eq_p:cc</code>	1413	2133, 2138, 2144, 2149, 2196, 2207,
<code>\cs_if_eq_p:cN</code>	1405	2216, 2391, 2397, 2405, 2412, 2819,
<code>\cs_if_eq_p:Nc</code>	1409	2845, 2860, 2941, 3031, 3040, 3049,
<code>\cs_if_eq_p:NN</code>	1405, 1409, 1413	3062, 3204, 3206, 3214, 3225, 3236,
<code>\cs_if_exist:c</code>	<u>1094</u> , 1106	3261, 3262, 3340, 3350, 3432, 3440,
<code>\cs_if_exist:cF</code> ..	3797, 3804, 3806, 9268	3448, 3454, 3460, 3468, 3476, 3482,
<code>\cs_if_exist:cT</code>	8423, 9512	3489, 3503, 3509, 3541, 3573, 3575,
<code>\cs_if_exist:cTF</code> ...	6864, 7848, 8028,	3581, 3593, 3601, 3634, 3636, 3638,
	8054, 8065, 8091, 8326, 8701, 8711,	3676, 3681, 3686, 3710, 3718, 3720,
	9143, 9242, 9549, 9563, 9569, 12868	3729, 3731, 3740, 3742, 3752, 3761,
<code>\cs_if_exist:N</code>	<u>1094</u> , 1094	3763, 3765, 3865, 3880, 3969, 4406,
<code>\cs_if_exist:Nf</code> ..	1210, 9186, 9201, 9342	4407, 4417, 4465, 4518, 4525, 4576,
<code>\cs_if_exist:NT</code>		4586, 4589, 4591, 4599, 4613, 4621,
	.. 1439, 1450, 8102, 8115, 9692, 9710	4626, 4632, 4642–4644, 4646, 4648,
<code>\cs_if_exist:NTF</code>		4651, 4659, 4705, 4713, 4762, 4789,
 <u>21</u> , 1419, 2691, 4223, 4225,	4799–4802, 4811, 4812, 4818, 4829,
	6011, 6014, 6339, 6345, 6862, 8379	4838, 4839, 4846, 4852, 4854, 4856,
<code>\cs_if_free:c</code>	<u>1122</u> , 1134	4861, 4870, 4872, 4874, 4880, 4889,
<code>\cs_if_free:cT</code>	1846	4895, 4907–4909, 4921, 4923, 4925,
<code>\cs_if_free:N</code>	<u>1122</u> , 1122	4932, 4933, 4941, 4991, 4993, 5002,
<code>\cs_if_free:Nf</code>	1187, 1197	5163, 5197, 5198, 5209, 5215, 5309,
<code>\cs_if_free:NTF</code> <u>21</u> , 1802, 8191, 8193, 13160		5314, 5384, 5392, 5437, 5466, 5486,
<code>\cs_meaning:c</code>	819, 819	5516, 5522, 5576, 5591, 5593, 5594,
<code>\cs_meaning:N</code>	<u>15</u> , <u>803</u> , 807, 819	5599, 5604, 5732, 5734, 5777, 5829,
<code>\cs_new:cn</code>	<u>1392</u>	5830, 5835, 5863, 5868, 5874, 5882,
<code>\cs_new:cpn</code> ...	<u>1242</u> , 1246, 1483, 1952,	5883, 5892, 5894, 5902, 5917, 5924,
	1965, 1998, 2000, 2002, 2003, 2168–	5932, 5934, 5948, 5953, 5974, 6034,
	2171, 2173, 2175, 2177, 2179, 2181,	6139, 6145, 6183, 6226, 6233, 6259,
	2183, 2185–2195, 3352, 3360, 3368,	6264, 6273, 6280, 7873, 8143, 8361,
	3376, 3384, 3392, 3400, 3981–3987	8368, 8613, 8972, 9058, 13137, 13148
<code>\cs_new:cpx</code>	<u>1242</u> , 1247, 1487, 1848	<code>\cs_new:Npx</code>
<code>\cs_new:cx</code>	<u>1392</u>	.. <u>1218</u> , 1229, 1247, 1479, 8234, 8959
<code>\cs_new:Nn</code>	<u>12</u> , <u>1360</u>	<code>\cs_new:Nx</code>
<code>\cs_new:Npn</code>	<u>10</u> ,	<code>\cs_new_eq:cc</code>
	906, 938, <u>1218</u> , 1228, 1246, 1287,	962, <u>1264</u> , 1271, 1495
		<code>\cs_new_eq:cN</code>
		<u>1264</u> ,

- 1269, 1493, 6005, 11829, 11853, 11884
 \cs_new_eq:Nc [1264](#), 1270, 1494
 \cs_new_eq:NN [14](#), [1264](#), 1264, 1269–1271,
 1427–1438, 1474–1489, 1492–1495,
 1498, 1499, 1502, 1504–1506, 1535,
 1875, 1889–1896, 2568–2570, 2853–
 2855, 3095–3099, 3119, 3120, 3123–
 3126, 3129, 3132–3139, 3141, 3143–
 3157, 3159, 3161–3167, 3170–3185,
 3194–3199, 3336, 3827, 3828, 3855–
 3857, 3902–3904, 4046, 4048, 4056,
 4057, 4119, 4121, 4124, 4129, 4131,
 4132, 4174, 4176, 4228–4235, 4368,
 4369, 4563, 4564, 4567, 4770, 4962–
 4969, 4972–4979, 4982–4986, 4989,
 4990, 5009–5026, 5199–5201, 5263–
 5288, 5525, 5526, 5529, 5530, 5541–
 5558, 5671–5688, 5842, 5843, 5980,
 5981, 5984, 5985, 5988, 5989, 6004,
 6015–6023, 6204, 6205, 6288, 6289,
 6300, 6363–6365, 6378, 6379, 6390–
 6392, 6411, 6419, 6425, 6431, 6450–
 6457, 6465, 6466, 6496–6505, 7905,
 7925–7929, 7949, 7959, 7979, 8163,
 8178, 8195, 8408, 8409, 8976, 8979–
 8983, 9537, 9639–9641, 10152–
 10161, 13133, 13134, 13236–13239
 \cs_new_nopar:cn [1392](#)
 \cs_new_nopar:cpn
 [1234](#), 1240, 1482, 2016–2028,
 2030, 10203, 10204, 10206, 10208,
 10210, 10212, 10214, 10216, 10218,
 10264, 10266, 10268, 10270, 10272,
 10274, 10276, 10278, 10280, 10326,
 10331, 10336, 10341, 10346, 10351,
 10356, 10361, 10366, 10371, 10376
 \cs_new_nopar:cpx [1234](#), 1241, 1486
 \cs_new_nopar:cx [1392](#)
 \cs_new_nopar:Nn [12](#), [1360](#)
 \cs_new_nopar:Npn
 [10](#), [1218](#), 1226, 1240, 1303,
 1328, 1405–1417, 1426, 1461, 1474,
 1519, 1553, 1564, 1633, 1669–1676,
 1683–1687, 1734–1736, 1829, 2014,
 2015, 2156, 2163, 2165, 2167, 2258,
 2260, 2269, 2274, 2283, 2288, 2476,
 2478, 2546, 2548, 2552, 2554, 2558,
 2560, 2564, 2566, 2586, 2675, 2715,
 2722, 2733, 2744, 2755, 2766, 2774,
 2782, 2790, 2811, 2818, 2827, 2836,
 2856–2859, 2910, 2919, 2927, 3028,
 3102, 3103, 3105, 3106, 3108, 3109,
 3111, 3112, 3114, 3115, 3309, 3337,
 3488, 3640, 3645, 3650, 3655, 3660–
 3675, 3781, 3790, 3824, 3826, 3858,
 3950, 3952, 4044, 4117, 4122, 4125,
 4172, 4178, 4412, 4520, 4565, 4568,
 4581, 4657, 4665, 5112, 5316, 5394,
 5410, 5418, 5420, 5429, 5769, 6178,
 6490, 6856–6859, 7878, 8194, 8359,
 8482–8484, 8584–8589, 9184, 9199,
 9306, 9527, 9529, 9538, 9547, 9556,
 9573, 10162, 10165, 10182, 10183,
 10189, 10198, 10219, 10225, 10227,
 10230, 10242, 10253, 10262, 10281,
 10289, 10294, 10297, 10309, 10318,
 10320, 10377, 10390, 10409, 10429,
 10435, 10474, 10513–10515, 10517
 \cs_new_nopar:Npx
 [1218](#), 1227, 1241, 1478, 1835, 4329
 \cs_new_nopar:Nx [1360](#)
 \cs_new_protected:cn [1392](#)
 \cs_new_protected:cpn [1254](#), 1258,
 1485, 9360, 9362, 9364, 9366, 9368,
 9378, 9380, 9398, 9408, 9410, 9414
 \cs_new_protected:cpx [1254](#), 1259, 1489
 \cs_new_protected:cx [1392](#)
 \cs_new_protected:Nn [12](#), [1360](#)
 \cs_new_protected:Npn
 [10](#), 921, 955, [1218](#),
 1232, 1258, 1260, 1264, 1305, 1477,
 1536, 1764, 1844, 2226, 2232, 2243,
 2374, 2375, 2865, 2871, 2888, 2890,
 2892, 2906, 2908, 4204, 4209, 4236,
 4238, 4240, 4242, 4244, 4246, 4254,
 4256, 4258, 4260, 4262, 4264, 4266,
 4268, 4278, 4280, 4282, 4284, 4286,
 4288, 4290, 4292, 4317, 4357, 4383,
 4531, 4541, 4552, 4556, 4636, 4638,
 4768, 4947, 5033, 5035, 5041, 5043,
 5050, 5052, 5054, 5066, 5068, 5070,
 5125, 5138, 5222, 5227, 5232, 5244,
 5251, 5446, 5451, 5456, 5461, 5540,
 5616, 5618, 5626, 5640, 5650, 5661,
 5668, 5690, 5692, 5694, 5706, 5708,
 5710, 5753, 5785, 5795, 5806, 5816,
 5823, 5954, 5956, 5958, 5991, 5992,
 6004–6010, 6013, 6026, 6035, 6042,
 6044, 6046, 6052, 6058, 6062, 6068,
 6074, 6083–6085, 6089, 6109, 6167,

- 6194, 6247, 6292, 6294, 6324, 6382,
6384, 6386, 6388, 6434, 6438, 6458,
6460, 6461, 6463, 6471, 6473, 6474,
6478, 6484, 6654, 6681, 6860, 6896,
6913, 8233, 8240, 8279, 8402, 8404,
8421, 8430, 8432, 8439, 8441, 8448,
8498, 8513, 8521, 8529, 8531, 8554,
8569, 8576, 8688, 8733, 8743, 8759,
8771, 8773, 8775, 8777, 8779, 8806,
8815, 8828, 8830, 8832, 8834, 8837,
8850, 8852, 8854, 8856, 8986–8992,
9014, 9028, 9040, 9060, 9065, 9086,
9092, 9122, 9124, 9136, 9141, 9167,
9182, 9224, 9240, 9266, 9277, 9282,
9293, 9318, 9444, 9446, 9454, 9456,
9473, 9478, 9505, 13142, 13151, 13219
\cs_new_protected:Npx
..... [1218](#), 1233, 1259, 1481
\cs_new_protected:Nx [1360](#)
\cs_new_protected_nopar:cn [1392](#)
\cs_new_protected_nopar:cpn
..... [1248](#), 1252, 1484, 9350,
9352, 9354, 9356, 9358, 9370, 9372,
9374, 9376, 9382, 9384, 9386, 9388,
9390, 9392, 9394, 9396, 9400, 9402,
9404, 9406, 9412, 9416, 9418, 9420,
9422, 9424, 9426, 9428, 9430, 9432,
9434, 9436, 9438, 9440, 9442, 12875,
12901, 12936, 12954, 12986, 13018
\cs_new_protected_nopar:cpx
..... [1248](#), 1253, 1488
\cs_new_protected_nopar:cx [1392](#)
\cs_new_protected_nopar:Nn [13](#), [1360](#)
\cs_new_protected_nopar:Npn
..... [11](#), [1218](#), 1230,
1235, 1252, 1261–1263, 1269–1276,
1278, 1325, 1476, 1668, 1677–1682,
1688–1694, 1875, 1877, 1879, 1881,
1883, 2262, 2295, 2384, 2474, 2480,
2482, 2484, 2486, 2488, 2490, 2492,
2494, 2496, 2498, 2500, 2502, 2504,
2506, 2508, 2510, 2512, 2514, 2516,
2518, 2520, 2522, 2524, 2526, 2528,
2530, 2532, 2534, 2536, 2538, 2540,
2542, 2544, 2550, 2556, 2562, 2568,
2861, 2863, 2950, 2959, 3077, 3088,
3090, 3092, 3269, 3276, 3297, 3298,
3301, 3304, 3307, 3311, 3313, 3319,
3321, 3323, 3325, 3331, 3333, 3906,
3913, 3914, 3917, 3919, 3922, 3925,
3928, 3930, 3932, 3934, 3940, 3942,
3945, 3947, 4065, 4072, 4073, 4076,
4078, 4081, 4084, 4087, 4089, 4092,
4094, 4139, 4146, 4148, 4151, 4153,
4156, 4159, 4162, 4164, 4167, 4169,
4198, 4216, 4218, 4222, 4224, 4313,
4315, 4340, 4342, 4344, 4371, 4373,
4375, 4377, 4419, 4421, 4425, 4427,
4503–4505, 4554, 4892, 4956, 4958,
5027, 5029, 5119, 5128, 5130, 5132,
5145, 5151, 5169, 5171, 5173, 5179,
5202, 5238, 5290, 5473, 5475, 5477,
5479, 5492, 5494, 5496, 5559, 5561,
5563, 5622, 5624, 5642, 5648, 5653,
5655, 5657, 5844, 6105, 6107, 6207,
6331, 6333, 6337, 6343, 6351, 6353,
6357, 6359, 6369, 6371, 6373, 6412,
6414, 6433, 6435, 6440, 6444, 6446,
6469, 6470, 6475, 6480, 6486, 6492,
6508, 6527, 6544, 6578, 6586, 6597,
6608, 6619, 6630, 6641, 6701, 6722,
6742, 6764, 6780, 6876, 6885, 6940,
6955, 6957, 6985, 6987, 7000, 7010,
7017, 7024, 7031, 7046, 7053, 7066,
7079, 7084, 7095, 7129, 7144, 7230,
7244, 7281, 7299, 7309, 7328, 7333,
7347, 7352, 7362, 7373, 7385, 7397,
7411, 7446, 7458, 7464, 7470, 7482,
7499, 7508, 7515, 7521, 7523, 7531,
7544, 7557, 7572, 7588, 7597, 7603,
7612, 7619, 7680, 7727, 7735, 7765,
7814, 7822, 7846, 7906, 7912, 7916,
7962, 7965, 7974, 7982, 7995, 8010,
8018, 8026, 8049, 8063, 8086, 8100,
8113, 8128, 8148, 8180, 8183, 8184,
8187, 8189, 8190, 8192, 8286, 8296,
8312, 8324, 8333, 8347, 8353, 8393,
8395, 8397, 8399, 8593, 8715, 8723,
8742, 8745, 8747, 8749, 8751, 8753,
8755, 8757, 9131, 9150, 9157, 9214,
9254, 9287, 9296, 9301, 9308, 9334,
9340, 9346, 9468, 9678, 9688, 9722,
9739, 9744, 9746, 9837, 9839, 9850,
9860, 9885, 9893, 9895, 9897, 9903,
9905, 9922, 9934, 9989–9991, 9999,
10009, 10024, 10034, 10049, 10050,
10056, 10062, 10064, 10068–10070,
10102, 10104, 10106, 10519, 10522,
10525, 10554, 10557, 10560, 10590,
10599, 10612, 10638, 10639, 10642,

- 10663, 10664, 10667, 10689, 10690,
10693, 10706, 10743, 10760, 10792,
10793, 10796, 10810, 10811, 10814,
10865, 10895, 10907, 10912, 10918,
10923, 10924, 10927, 10962, 11008,
11026, 11043, 11054, 11055, 11060,
11069, 11075, 11091, 11114, 11158,
11218, 11235, 11240, 11250, 11260,
11268, 11278, 11301, 11311, 11325,
11331, 11344, 11363, 11381, 11420,
11438, 11448, 11475, 11476, 11479,
11526, 11559, 11572, 11573, 11576,
11612, 11645, 11668, 11704, 11719,
11773, 11774, 11777, 11824, 11834,
11863, 11893, 11966, 11967, 11970,
12014, 12042, 12054, 12089, 12121,
12141, 12147, 12154, 12218, 12266,
12284, 12285, 12288, 12321, 12345,
12379, 12398, 12408, 12422, 12434,
12456, 12481, 12489, 12501, 12507,
12561, 12571, 12586, 12618, 12619,
12622, 12673, 12709, 12763, 12775,
12866, 13052, 13058, 13064, 13070,
13076, 13082, 13088, 13100, 13108,
13113, 13122, 13158, 13188, 13198
\cs_new_protected_nopar:Npx
.. 1218, 1231, 1253, 1480, 1833, 8341
\cs_new_protected_nopar:Nx 1360
\cs_set:cn 1376
\cs_set:cpn .. 1242, 1242, 8434, 8436, 9280
\cs_set:cpx 1242, 1243,
2296, 2300, 2304, 2308, 2312, 2321,
2330, 2339, 2348, 2350, 2352, 2354,
2356, 2358, 2360, 2362, 2364, 9285
\cs_set:cx 1376
\cs_set:Nn 13, 1335
\cs_set:Npn 11,
822, 824, 850, 859–888, 898, 929,
963, 964, 1051–1054, 1072, 1077,
1087, 1090, 1092, 1218, 1234, 1242,
1335, 1368, 1507–1510, 2371, 2372,
3060, 3070, 3201, 3988, 3996, 4004,
4010, 4016, 4024, 4032, 4038, 4511,
4597, 5574, 5712, 5755, 8217, 8267
\cs_set:Npx 822, 826, 854, 1243, 3071, 4391
\cs_set:Nx 1335
\cs_set_eq:cc . 960, 1260, 1263, 1892, 4231
\cs_set_eq:cN 1260, 1261, 1891, 4229, 6007
\cs_set_eq:Nc 1260, 1262, 1890, 4230
\cs_set_eq:NN 15, 1260, 1260–1263, 1267,
1272, 1441–1448, 1452–1459, 1841,
1878, 1880, 1889, 2609, 2869, 2873,
2894, 2896, 2955, 2973, 3074, 4228,
5481, 5482, 5484, 6006, 7033–7040,
7048–7051, 7969, 7970, 8257–8259,
9460, 9462, 10965, 11058, 11897, 12819
\cs_set_eq:NwN 1511, 1511
\cs_set_nopar:cn 1376
\cs_set_nopar:cpn 1234, 1236
\cs_set_nopar:cpx 1234, 1237
\cs_set_nopar:cx 1376
\cs_set_nopar:Nn 13, 1335
\cs_set_nopar:Npn 11, 822, 822, 824–826,
828–830, 833, 889, 891, 1057, 1064,
1183, 1236, 2962, 2968, 8485, 10144
\cs_set_nopar:Npx
... 822, 823, 827, 831, 835, 1237,
1538, 2875, 2880, 2897, 2898, 4237,
4239, 4241, 4255, 4257, 4259, 4261,
4279, 4281, 4283, 4285, 8251–8255
\cs_set_nopar:Nx 1335
\cs_set_protected:cn 1376
\cs_set_protected:cpn .. 1254, 1254, 8596
\cs_set_protected:cpx ... 1254, 1255,
1337, 1370, 8598, 8600, 8602, 8604
\cs_set_protected:cx 1376
\cs_set_protected:Nn 13, 1335
\cs_set_protected:Npn
. 11, 822, 832, 851, 893, 901, 909,
913, 916, 924, 933, 943, 946, 950,
959, 961, 965, 973, 981, 989, 997,
1005, 1010, 1018, 1026, 1034, 1039,
1041, 1254, 4331, 5101, 6024, 6028,
6036, 6964, 8808, 8810, 8812, 8934
\cs_set_protected:Npx 822, 834, 1255, 8695
\cs_set_protected:Nx 1335
\cs_set_protected_nopar:cn 1376
\cs_set_protected_nopar:cpn . 1248, 1248
\cs_set_protected_nopar:cpx . 1248, 1249
\cs_set_protected_nopar:cx 1376
\cs_set_protected_nopar:Nn 13, 1335
\cs_set_protected_nopar:Npn
..... 11, 310, 822, 828,
832, 834, 838, 840, 842, 844, 846,
848, 1163, 1165, 1167, 1179, 1181,
1185, 1195, 1206, 1208, 1216, 1220,
1248, 6945, 8186, 8188, 9936, 9945,
9953, 9962, 10898, 13177, 13181,
13208, 13209, 13214, 13215, 13227

- \cs_set_protected_nopar:Npx 296, [822](#), 830, 1249,
8564, 8690, 10080, 10122, 10142,
10534, 10570, 10647, 10723, 10834,
10941, 10951, 10977, 11504, 11517,
11604, 11802, 11815, 11999, 12304,
12312, 12338, 12530, 12644, 12650,
12661, 12694, 12701, 12747, 12782
 - \cs_set_protected_nopar:Nx [1335](#)
 - \cs_show:c [819](#), [821](#), [9574](#)
 - \cs_show:N [15](#), [803](#), 808, [821](#), [4768](#)
 - \cs_split_function:NN [19](#), [897](#), [905](#), [912](#),
[920](#), [928](#), [937](#), [945](#), [954](#), [1047](#), [1048](#),
[1066](#), [1072](#), [1091](#), [1093](#), [1288](#), [1768](#)
 - \cs_split_function_aux:w [1066](#), [1074](#), [1077](#)
 - \cs_split_function_auxii:w
..... [1066](#), [1085](#), [1087](#)
 - \cs_tmp:w [851](#), [854](#), [857](#), [859](#),
[1218](#), [1226–1234](#), [1236–1259](#), [1335](#),
[1344–1368](#), [1376–1399](#), [1804](#), [1841](#)
 - \cs_to_str:N
.. [4](#), [17](#), [1057](#), [1057](#), [1075](#), [2272](#), [8195](#)
 - \cs_to_str_aux:w [1057](#), [1060](#), [1064](#)
 - \cs_undefine:c [1276](#), [1278](#), [1499](#)
 - \cs_undefine:N [15](#), [1276](#), [1276](#), [1498](#)
 - \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](#)
 - \currentifttype [693](#)
- D**
- \d [1817](#), [2703](#)
 - \dagger [3870](#), [3876](#)
 - \day [651](#)
 - \ddagger [3871](#), [3877](#)
 - \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 [2374](#), [2375](#), [8986–8992](#)
 - \Depth [7031](#), [7034](#), [7038](#), [7042](#), [7049](#)
 - \detokenize [32](#), [35](#), [80](#), [93](#), [96](#),
[167](#), [170](#), [176](#), [185](#), [190](#), [192](#), [198](#),
[201](#), [204](#), [214](#), [269](#), [271](#), [276](#), [278](#), [683](#)
 - \dim_add:cn [3940](#)
 - \dim_add:Nn [70](#), [3940](#), [3940](#), [3942](#), [3943](#)
 - \dim_compare:n [3959](#), [3959](#)
 - \dim_compare:nF [3998](#), [4013](#)
 - \dim_compare:nNn [3954](#), [3954](#)
 - \dim_compare:nNnF [4026](#), [4041](#)
 - \dim_compare:nNnT [3929](#), [3931](#),
[3933](#), [3935](#), [4018](#), [4035](#), [7250](#), [7255](#)
 - \dim_compare:nNnTF ... [72](#), [2117](#), [6683](#),
[6686](#), [7147](#), [7150](#), [7153](#), [7162](#), [7165](#),
[7168](#), [7177](#), [7184](#), [7262](#), [7375](#), [7387](#)
 - \dim_compare:nT [3990](#), [4007](#)
 - \dim_compare:nTF [72](#)
 - \dim_compare_<:nw [3959](#)
 - \dim_compare_=:nw [3959](#)
 - \dim_compare_>:nw [3959](#)
 - \dim_compare_aux:wNN ... [3959](#), [3961](#), [3969](#)
 - \dim_compare_p:nNn [3954](#)
 - \dim_do_until:nn ... [73](#), [3988](#), [4010](#), [4014](#)
 - \dim_do_until:nNnn .. [73](#), [4016](#), [4038](#), [4042](#)
 - \dim_do_while:nn [73](#), [3988](#), [4004](#)
 - \dim_do_while:nNnn
..... [73](#), [4008](#), [4016](#), [4032](#), [4036](#)
 - \dim_eval:n [74](#), [2111](#), [4044](#),
[4044](#), [6932](#), [6976](#), [7060](#), [7073](#), [7091](#),
[7093](#), [7099](#), [7110](#), [7124](#), [7358](#), [7359](#),
[7527](#), [7528](#), [7535](#), [7536](#), [7616](#), [7623](#)
 - \dim_eval:w
.. [80](#), [3902](#), [3903](#), [3918](#), [3941](#), [3946](#),
[3953](#), [3956](#), [3961](#), [3981–3987](#), [4045](#),
[6370](#), [6372](#), [6374](#), [6383](#), [6385](#), [6387](#),
[6389](#), [6439](#), [6459](#), [6472](#), [6485](#), [6509](#)
 - \dim_eval_end [80](#)
 - \dim_eval_end: [3902](#), [3904](#), [3918](#),
[3941](#), [3946](#), [3953](#), [3956](#), [3962](#), [4045](#),
[6370](#), [6372](#), [6374](#), [6383](#), [6385](#), [6387](#),
[6389](#), [6439](#), [6459](#), [6472](#), [6485](#), [6509](#)
 - \dim_gadd:cn [3940](#)
 - \dim_gadd:Nn [70](#), [3940](#), [3942](#), [3944](#)
 - \dim_gset:cn [3917](#)
 - \dim_gset:Nn [71](#), [3917](#), [3919](#), [3921](#), [3931](#), [3935](#)
 - \dim_gset_eq:cc [3922](#)
 - \dim_gset_eq:cN [3922](#)
 - \dim_gset_eq:Nc [3922](#)
 - \dim_gset_eq:NN [71](#), [3922](#), [3925–3927](#)

- \dim_gset_max:cn [3928](#)
 - \dim_gset_max:Nn [71](#), [3928](#), [3930](#), [3937](#)
 - \dim_gset_min:cn [3928](#)
 - \dim_gset_min:Nn [71](#), [3928](#), [3934](#), [3939](#)
 - \dim_gsub:cn [3940](#)
 - \dim_gsub:Nn [71](#), [3940](#), [3947](#), [3949](#)
 - \dim_gzero:c [3913](#)
 - \dim_gzero:N [70](#), [3913](#), [3914](#), [3916](#)
 - \dim_new:c [3905](#)
 - \dim_new:N [70](#),
[3905](#), [3906](#), [3912](#), [4051](#), [4052](#), [4059](#)–
[4063](#), [6513](#)–[6520](#), [6812](#), [6838](#), [6839](#),
[6844](#)–[6847](#), [6852](#)–[6855](#), [7406](#)–[7410](#),
[7542](#), [7543](#), [7667](#), [7669](#), [7670](#), [10150](#)
 - \dim_ratio:nn [72](#), [3950](#), [3950](#)
 - \dim_ratio_aux:n [3950](#), [3951](#), [3952](#)
 - \dim_set:cn [3917](#)
 - \dim_set:Nn [70](#), [3917](#), [3917](#),
[3919](#), [3920](#), [3929](#), [3933](#), [4053](#), [6546](#)–
[6548](#), [6595](#), [6606](#), [6659](#)–[6661](#), [6684](#),
[6685](#), [6688](#), [6690](#), [6694](#), [6696](#), [6706](#)–
[6708](#), [6727](#)–[6729](#), [6749](#)–[6751](#), [6768](#),
[6769](#), [6772](#), [6773](#), [6919](#), [6962](#), [7041](#)–
[7044](#), [7149](#), [7154](#), [7164](#), [7169](#), [7179](#),
[7186](#), [7219](#), [7242](#), [7253](#), [7312](#), [7313](#),
[7315](#), [7317](#), [7335](#), [7336](#), [7452](#), [7496](#),
[7497](#), [7501](#)–[7504](#), [7517](#), [7592](#), [7595](#),
[7668](#), [7773](#), [7774](#), [7825](#)–[7827](#), [7829](#)
 - \dim_set_eq:cc [3922](#)
 - \dim_set_eq:cN [3922](#)
 - \dim_set_eq:Nc [3922](#)
 - \dim_set_eq:NN [71](#), [3922](#), [3922](#)–[3924](#)
 - \dim_set_max:cn [3928](#)
 - \dim_set_max:Nn
.... [71](#), [3928](#), [3928](#), [3936](#), [7511](#), [7513](#)
 - \dim_set_min:cn [3928](#)
 - \dim_set_min:Nn
[71](#), [3928](#), [3932](#), [3938](#), [7510](#), [7512](#), [7522](#)
 - \dim_show:c [4048](#)
 - \dim_show:N [74](#), [4048](#), [4048](#), [4049](#)
 - \dim_sub:cn [3940](#)
 - \dim_sub:Nn [71](#), [3940](#), [3945](#), [3947](#), [3948](#)
 - \dim_until_do:nn [73](#), [3988](#), [3996](#), [4001](#)
 - \dim_until_do:nNnn .. [73](#), [4016](#), [4024](#), [4029](#)
 - \dim_use:c [4046](#)
 - \dim_use:N [74](#),
[3961](#), [4045](#), [4046](#), [4046](#), [4047](#), [7087](#),
[7089](#), [7093](#), [7104](#), [7117](#), [7343](#), [7449](#),
[7451](#), [7454](#), [7456](#), [7462](#), [7468](#), [7477](#)–
[7479](#), [7601](#), [7608](#), [7854](#)–[7856](#), [10114](#)
 - \dim_while_do:nn [74](#), [3988](#), [3988](#), [3993](#)
 - \dim_while_do:nNnn .. [73](#), [4016](#), [4016](#), [4021](#)
 - \dim_zero:c [3913](#)
 - \dim_zero:N . [70](#), [3913](#), [3913](#)–[3915](#), [6549](#),
[6662](#), [6709](#), [6730](#), [6752](#), [7140](#), [7141](#)
 - \dimen [657](#)
 - \dimendef [356](#)
 - \dimexpr [710](#)
 - \directlua [15](#), [756](#)
 - \discretionary [520](#)
 - \displayindent [485](#)
 - \displaylimits [495](#)
 - \displaystyle [473](#)
 - \displaywidowpenalties [723](#)
 - \displaywidowpenalty [484](#)
 - \displaywidth [486](#)
 - \divide [363](#)
 - \doublehyphendemerits [553](#)
 - \dp [664](#)
 - \driver_box_rotate_begin: [6567](#)
 - \driver_box_rotate_end: [6569](#)
 - \driver_box_scale_begin: [6784](#)
 - \driver_box_scale_end: [6786](#)
 - \driver_color_ensure_current: ... [7913](#)
 - \dump [647](#)
- E**
- \E [1823](#), [2705](#)
 - \edef [33](#),
[68](#), [82](#), [164](#), [166](#), [181](#), [201](#), [266](#), [273](#), [351](#)
 - \else [14](#), [22](#), [63](#), [117](#), [137](#), [172](#), [187](#), [207](#), [404](#)
 - \else: [782](#), [785](#), [1081](#), [1098](#),
[1101](#), [1110](#), [1116](#), [1126](#), [1129](#), [1138](#),
[1144](#), [1282](#), [1293](#), [1318](#), [1403](#), [1466](#),
[1471](#), [1508](#), [1559](#), [1907](#), [1921](#), [1935](#),
[1945](#), [1956](#), [1959](#), [1969](#), [1972](#), [1985](#),
[1994](#), [2251](#), [2253](#), [2255](#), [2257](#), [2401](#),
[2417](#), [2427](#), [2435](#), [2448](#), [2457](#), [2591](#),
[2596](#), [2601](#), [2606](#), [2613](#), [2619](#), [2624](#),
[2629](#), [2634](#), [2639](#), [2644](#), [2649](#), [2654](#),
[2659](#), [2679](#), [2687](#), [2694](#), [2728](#), [2739](#),
[2750](#), [2761](#), [2823](#), [2832](#), [2840](#), [2849](#),
[2915](#), [2923](#), [2945](#), [3220](#), [3231](#), [3241](#),
[3246](#), [3249](#), [3252](#), [3345](#), [3356](#), [3364](#),
[3372](#), [3380](#), [3388](#), [3396](#), [3404](#), [3412](#),
[3420](#), [3428](#), [3631](#), [3957](#), [3964](#), [4103](#),
[4445](#), [4457](#), [4470](#), [4480](#), [4496](#), [4572](#),
[4674](#), [4694](#), [4709](#), [4717](#), [4727](#), [4746](#),
[4758](#), [4795](#), [6124](#), [6149](#), [6394](#), [6396](#),
[6406](#), [8372](#), [8383](#), [8386](#), [9843](#), [9872](#),

9918, 9929, 9979, 9985, 9995, 10087,	
10129, 10172, 10176, 10193, 10237,	
10245, 10248, 10257, 10285, 10304,	
10313, 10381, 10384, 10401, 10404,	
10420, 10423, 10446, 10449, 10452,	
10455, 10458, 10461, 10464, 10467,	
10470, 10485, 10488, 10491, 10494,	
10497, 10500, 10503, 10506, 10509,	
10541, 10577, 10606, 10620, 10625,	
10676, 10714, 10730, 10755, 10778,	
10788, 10848, 10851, 10946, 10956,	
10991, 10994, 11030, 11032, 11035,	
11081, 11086, 11107, 11283, 11355,	
11369, 11404, 11429, 11444, 11459,	
11492, 11509, 11513, 11532, 11547,	
11589, 11601, 11618, 11633, 11661,	
11663, 11673, 11689, 11694, 11709,	
11746, 11752, 11757, 11790, 11807,	
11811, 11828, 11840, 11843, 11846,	
11855, 11859, 11886, 11889, 11914,	
11984, 11996, 12007, 12023, 12028,	
12033, 12038, 12046, 12062, 12076,	
12082, 12107, 12126, 12161, 12169,	
12193, 12196, 12239, 12245, 12250,	
12303, 12311, 12335, 12349, 12352,	
12366, 12384, 12390, 12430, 12438,	
12466, 12544, 12552, 12575, 12604,	
12610, 12649, 12656, 12666, 12678,	
12692, 12700, 12713, 12727, 12736,	
12742, 12826, 12834, 12884, 12888,	
12892, 12896, 12911, 12915, 12920,	
12927, 12931, 12941, 12945, 12948,	
12959, 12963, 12967, 12972, 12977,	
12991, 12995, 12999, 13004, 13009	
\emergencystretch	568
\end	442
\EndCatcodeRegime	13215
\endcsname 13, 32, 35, 62, 80, 93, 96, 167,	
170, 176, 185, 190, 192, 201, 204,	
214, 229, 233, 269, 271, 276, 278, 444	
\endgroup	12, 61, 111, 120, 228, 232, 377
\endinput	264, 416
\endL	731
\endlinechar	79, 92, 289, 458
\endR	733
\eqno	478
\errhelp	250, 424
\errmessage	418
\ERROR	2371, 2372
\errorcontextlines	425
\errorstopmode	439
\escapechar	457
\etex_beginL:D	730
\etex_beginR:D	732
\etex_botmarks:D	679
\etex_clubpenalties:D	721
\etex_currentgrouplevel:D	695
\etex_currentgrouptype:D	696
\etex_currentifbranch:D	692
\etex_currentiflevel:D	691
\etex_currentifttype:D	693
\etex_detokenize:D	683, 4564, 4565
\etex_dimexpr:D	710, 3903
\etex_displaywidowpenalties:D	723
\etex_endL:D	731
\etex_endR:D	733
\etex_eTeXrevision:D	675
\etex_eTeXversion:D	674
\etex_everyeof:D	735, 4320, 4347, 4360
\etex_firstmarks:D	678
\etex_fontcharhp:D	703
\etex_fontcharht:D	702
\etex_fontcharic:D	705
\etex_fontcharwd:D	704
\etex_glueexpr:D	711, 4077,
4088, 4093, 4118, 4123, 4126, 10109	
\etex_glueshrink:D	714, 4188
\etex_glueshrinkorder:D	716, 4112
\etex_gluestretch:D	713, 4187
\etex_gluestretchorder:D	715, 4111
\etex_gluetomu:D	717
\etex_ifcsname:D	672, 799
\etex_ifdefined:D	671, 798
\etex_iffontchar:D	701
\etex_interactionmode:D	699
\etex_interlinepenalties:D	720
\etex_lastlinefit:D	719
\etex_lastnodetype:D	700
\etex_marks:D	676
\etex_middle:D	724
\etex_muexpr:D	712, 4152, 4163, 4168, 4173
\etex_mutoglu:D	718
\etex_numexpr:D	709, 3195
\etex_pagediscards:D	727
\etex_parshapedimen:D	708
\etex_parshapeindent:D	706
\etex_parshapelength:D	707
\etex_predisplaydirection:D	734
\etex_protected:D	736, 817
\etex_readline:D	686, 8398, 8400

<code>\etex_savinghyphcodes:D</code>	725	1971, 1973, 1978, 1979, 1982, 1991,
<code>\etex_savingvdiscards:D</code>	726	1999, 2001–2003, 2006, 2011, 2159,
<code>\etex_scantokens:D</code>	684, 4325, 4351, 4364	2285, 2394, 2400, 2402, 2409, 2416,
<code>\etex_showgroups:D</code>	697	2418, 2672, 2693, 2712, 2719, 2729,
<code>\etex_showifs:D</code>	698	2740, 2751, 2762, 2771, 2779, 2787,
<code>\etex_showtokens:D</code>		2807, 2830, 2831, 2833, 2839, 2842,
	685, 4770, 5305, 5859, 7865, 8139, 8159	2914, 2916, 2922, 2924, 2937, 2944,
<code>\etex_splitbotmarks:D</code>	681	2946, 3035, 3044, 3053, 3339, 3342,
<code>\etex_splitdiscards:D</code>	728	3603, 3631, 3642, 3652, 3785, 3961,
<code>\etex_splitfirstmarks:D</code>	680	4325, 4364, 4401, 4409, 4414, 4455,
<code>\etex_TeXETstate:D</code>	729	4467, 4468, 4522, 4523, 4565, 4571,
<code>\etex_topmarks:D</code>	677	4633, 4637, 4639, 4653, 4661, 4671,
<code>\etex_tracingassigns:D</code>	687	4687, 4707, 4716, 4719, 4737–4739,
<code>\etex_tracinggroups:D</code>	694	4766, 4772, 4832, 4859, 4943, 4944,
<code>\etex_tracingifs:D</code>	690	5105, 5122, 5135, 5154, 5155, 5183,
<code>\etex_tracingnesting:D</code>	689	5184, 5206, 5211, 5305, 5306, 5320,
<code>\etex_tracingscantokens:D</code>	688	5326, 5347, 5354, 5422–5424, 5431,
<code>\etex_unexpanded:D</code>	682,	5432, 5649, 5659, 5720, 5728, 5733,
	802, 1750, 1752, 1755, 1760, 4601, 4893	5859, 5860, 5950, 6031, 6186, 6222,
<code>\etex_unless:D</code>	673, 787	6223, 6267, 7865, 7866, 8139, 8140,
<code>\etex_widowpenalties:D</code>	722	8159, 8160, 8364, 8371, 8374, 8545,
<code>\eTeXrevision</code>	675	8964–8967, 9022, 9024, 9063, 9171,
<code>\eTeXversion</code>	674	9299, 9652, 9664, 9735, 9838, 9858,
<code>\everycr</code>	386	9863, 9867, 9871, 9874, 9878, 9881,
<code>\everydisplay</code>	487	9900, 9919, 9928, 9930, 9940, 9942,
<code>\everyeof</code>	735	9957, 9959, 9971, 9986, 9994, 9996,
<code>\everyhbox</code>	626	10003, 10006, 10018, 10028, 10031,
<code>\everyjob</code>	31, 655	10043, 10092, 10113, 10134, 10163,
<code>\everymath</code>	511	10171, 10174, 10177, 10192, 10194,
<code>\everypar</code>	574	10228, 10236, 10238, 10256, 10258,
<code>\everyvbox</code>	627	10295, 10303, 10305, 10312, 10314,
<code>\exhyphenpenalty</code>	550	10380, 10383, 10385, 10397, 10416,
<code>\exp_after:wN</code>	29, 800,	10531, 10546, 10567, 10582, 10596,
	800, 818, 855, 890, 892, 976, 1044,	10635, 10655, 10681, 10686, 10687,
	1062, 1074, 1080, 1082, 1109, 1111,	10713, 10715, 10735, 10856, 10904,
	1114, 1137, 1139, 1142, 1281, 1283,	10915, 10957, 10999, 11015, 11021,
	1292, 1294, 1297, 1340, 1372, 1519,	11029, 11036–11038, 11245, 11247,
	1526, 1528, 1531, 1532, 1539, 1543,	11257, 11272, 11275, 11305, 11308,
	1544, 1549, 1550, 1555, 1560, 1562,	11319, 11322, 11338, 11354, 11360,
	1565, 1573, 1575, 1577, 1579, 1581,	11368, 11374–11376, 11445, 11458,
	1583, 1586–1588, 1592, 1595, 1600,	11470, 11497, 11514, 11552, 11563,
	1605–1607, 1611–1613, 1617–1619,	11565, 11567, 11569, 11594, 11602,
	1623–1625, 1629–1631, 1635–1638,	11638, 11649, 11651, 11653, 11655,
	1642–1645, 1649–1651, 1656–1659,	11701, 11716, 11770, 11795, 11812,
	1663–1666, 1698, 1699, 1702, 1705,	11827, 11831, 11856, 11860, 11887,
	1706, 1710, 1711, 1714, 1716, 1717,	11890, 11919, 11989, 11997, 12020–
	1719, 1722, 1723, 1728, 1729, 1733,	12022, 12024–12026, 12030–12032,
	1739–1741, 1745, 1747, 1749, 1750,	12039, 12045, 12051, 12061, 12065,
	1752, 1755, 1760, 1763, 1776, 1782,	12078–12080, 12084, 12085, 12098,
	1831, 1856, 1955, 1958, 1960, 1968,	12143, 12211, 12263, 12302, 12309,

- 12317, 12328, 12336, 12371, 12388,
 12426, 12429, 12465, 12467, 12478,
 12486, 12503, 12551, 12553, 12565,
 12568, 12615, 12667, 12677, 12689–
 12691, 12712, 12721–12725, 12729–
 12734, 12738–12740, 12743, 12755
 \exp_arg_last_unbraced:nn
 .. 1695, 1695, 1698, 1702, 1705, 1710
 \exp_arg_next:nnn 1520,
 1520, 1528, 1531, 1539, 1543, 1549
 \exp_arg_next_nobrace:nnn 1520, 1521, 1526
 \exp_args:cc 1578, 1578
 \exp_args:Nc 26, 818,
 818–821, 983, 991, 999, 1007, 1207,
 1217, 1235, 1261, 1269, 1274, 1304,
 1329, 1405–1408, 1426, 1578, 4536
 \exp_args:Ncc 1263,
 1271, 1275, 1413–1416, 1578, 1582
 \exp_args:Nccc 1578, 1584
 \exp_args:Ncco 1640, 1661
 \exp_args:Nccx 1683, 1692
 \exp_args:Ncf 1603, 1627
 \exp_args:NcNc 1640, 1647
 \exp_args:NcNo 1640, 1654
 \exp_args:Ncnx 1683, 1693
 \exp_args:Nco 1603, 1621
 \exp_args:Ncx 1669, 1678
 \exp_args:Nf 27, 1591,
 1591, 2098, 2111, 2202, 2203, 2212,
 2221, 3505, 3574, 3586, 3595, 3688,
 3701, 3715, 3725, 3736, 3747, 4858,
 4935, 5415, 5908, 5921, 5926, 5942
 \exp_args:Nff 1669, 1671
 \exp_args:Nfo 1669, 1670, 5896
 \exp_args:NNc 1047,
 1048, 1262, 1270, 1273, 1409–1412,
 1578, 1580, 1778, 2228, 2234, 5791
 \exp_args:Nnc 1669, 1669, 5801
 \exp_args:NNf 1603, 1603
 \exp_args:Nnf 926, 935, 944, 952, 1669, 1672
 \exp_args:Nnnc 1683, 1685
 \exp_args:NNNo . 28, 1573, 1576, 4326, 4352
 \exp_args:NNno 1683, 1683
 \exp_args:Nnno 1683, 1686
 \exp_args:NNNV 1640, 1640
 \exp_args:NNnx 28, 1683, 1688
 \exp_args:Nnnx 1683, 1690
 \exp_args:NNno 27, 1573,
 1574, 3493, 4937, 6025, 8276, 9298
 \exp_args:Nno 27, 1669,
 1673, 3027, 3968, 5810, 6040, 9551
 \exp_args:NNoo 28, 1683, 1684
 \exp_args:NNox 1683, 1689
 \exp_args:Nnox 1683, 1691
 \exp_args:NNV 1603, 1615
 \exp_args:NNv 1603, 1609
 \exp_args:NnV 1669, 1674
 \exp_args:NNx 28, 1669, 1677
 \exp_args:Nnx 1669, 1679
 \exp_args:No 26, 1573, 1573,
 3493, 3578, 4320, 4360, 4365, 4503–
 4505, 4555, 4799–4802, 5519, 5592,
 5727, 5746, 5751, 5936, 5940, 8360
 \exp_args:Noc 1669, 1675
 \exp_args:Noo 1669, 1676
 \exp_args:Nooo 1683, 1687
 \exp_args:Noox 1683, 1694
 \exp_args:Nox 1669, 1680
 \exp_args:NV 27, 1591, 1598
 \exp_args:Nv 27, 1591, 1593
 \exp_args:NVV 1603, 1633
 \exp_args:Nx 27, 1591, 1668, 1668
 \exp_args:Nxo 1669, 1681
 \exp_args:Nxx 1669, 1682
 \exp_eval_error_msg:w .. 1553, 1557, 1566
 \exp_eval_register:c 1550, 1553,
 1564, 1596, 1613, 1711, 1716, 1761
 \exp_eval_register:N
 30, 1544, 1553, 1553,
 1565, 1601, 1619, 1637, 1638, 1645,
 1706, 1714, 1724, 1730, 1742, 1756
 \exp_last_two_unbraced:Noo
 29, 1746, 1746, 7134, 7366, 7370
 \exp_last_two_unbraced_aux:noN
 1747, 1748
 \exp_last_unbraced:NcV . 1713, 1720, 4546
 \exp_last_unbraced:Nf
 28, 1713, 1718, 2271, 3584, 5970
 \exp_last_unbraced:Nfo . 1713, 1736, 5396
 \exp_last_unbraced:NNNo 1713, 1744
 \exp_last_unbraced:NNNV 1713, 1737
 \exp_last_unbraced:NNno
 .. 1713, 1732, 4845, 5773, 6180, 7340
 \exp_last_unbraced:Nno . 1713, 1734, 6261
 \exp_last_unbraced:NNV 1713, 1726
 \exp_last_unbraced:No
 .. 1713, 1717, 7718, 7723, 7802, 7808
 \exp_last_unbraced:Noo
 1713, 1735, 6134, 6275

- `\exp_last_unbraced:Nv` [1713](#), [1713](#)
`\exp_last_unbraced:Nv` [1713](#), [1715](#)
`\exp_not:c` [29](#), [1763](#),
[1763](#), [1804](#), [1854](#), [2297](#), [2301](#), [2306](#),
[2310](#), [2313](#), [2315–2317](#), [2322](#), [2324–](#)
[2326](#), [2331](#), [2333–2335](#), [2340](#), [2342–](#)
[2344](#), [2349](#), [2351](#), [2353](#), [2355](#), [2357](#),
[2359](#), [2361](#), [2363](#), [2365–2367](#), [2974](#),
[8599](#), [8601](#), [8603](#), [8605](#), [8969](#), [9047](#),
[9069](#), [9189](#), [9191](#), [9204](#), [9206](#), [9344](#)
`\exp_not:f` [30](#), [1750](#), [1751](#)
`\exp_not:N` .. [29](#), [800](#), [801](#), [1339–1341](#),
[1371–1373](#), [1519](#), [1555](#), [1763](#), [1804](#),
[2239](#), [2302](#), [2305](#), [2309](#), [2313](#), [2322](#),
[2331](#), [2340](#), [2408](#), [2415](#), [2446](#), [2455](#),
[2586](#), [2590](#), [2595](#), [2600](#), [2605](#), [2612](#),
[2618](#), [2623](#), [2628](#), [2633](#), [2638](#), [2643](#),
[2653](#), [2658](#), [2686](#), [2693](#), [2877](#), [2882](#),
[2900](#), [2913](#), [2943](#), [4331](#), [4333](#), [4347](#),
[4394](#), [4395](#), [4669](#), [4671](#), [4685](#), [4687](#),
[4715](#), [4722](#), [5255](#), [5518](#), [5520](#), [8274](#),
[8344](#), [8692](#), [8700](#), [8701](#), [8703](#), [8969](#),
[8970](#), [9189](#), [9191](#), [9204](#), [9206](#), [9232](#),
[9233](#), [9258](#), [9259](#), [9304](#), [9326](#), [9327](#),
[9344](#), [10083](#), [10125](#), [10144](#), [10537](#),
[10573](#), [10650](#), [10837](#), [10944](#), [10954](#),
[11507](#), [11520](#), [11607](#), [11805](#), [11818](#),
[12002](#), [12307](#), [12315](#), [12341](#), [12534](#),
[12536](#), [12538](#), [12785](#), [12787](#), [12789](#),
[12791](#), [12793](#), [13022](#), [13024](#), [13026](#),
[13028](#), [13030](#), [13032](#), [13034](#), [13036](#)
`\exp_not:n` [29](#), [800](#), [802](#),
[1464](#), [1519](#), [2240](#), [2298](#), [2408](#), [2415](#),
[2446](#), [2455](#), [2878](#), [2883](#), [2897](#), [2901](#),
[2973](#), [2975](#), [4207](#), [4237](#), [4243](#), [4255](#),
[4263](#), [4279](#), [4287](#), [4396](#), [4792](#), [5084](#),
[5181](#), [5256](#), [5312](#), [5466](#), [5490](#), [5523](#),
[5612](#), [5734](#), [5866](#), [5967](#), [5968](#), [6093](#),
[6094](#), [6115](#), [6231](#), [7876](#), [8146](#), [8181](#),
[8185](#), [8403](#), [8693](#), [8697](#), [8704](#), [8961](#),
[9235](#), [9329](#), [9367](#), [13048](#), [13149](#), [13152](#)
`\exp_not:o` [30](#), [1750](#), [1750](#), [4239](#),
[4245](#), [4255](#), [4257](#), [4259](#), [4261](#), [4263](#),
[4265](#), [4267](#), [4269](#), [4279](#), [4281](#), [4283](#),
[4285](#), [4287](#), [4289](#), [4291](#), [4293](#), [4406](#),
[4418](#), [4764](#), [4792](#), [5028](#), [5030](#), [5080](#),
[5567](#), [5569](#), [5633](#), [5727](#), [8565](#), [9049](#),
[9071](#), [9074](#), [9081](#), [9090](#), [9542](#), [9544](#)
`\exp_not:V`
[29](#), [1750](#), [1753](#), [4257](#), [4265](#), [4281](#), [4289](#)
`\exp_not:v` [30](#), [1750](#), [1758](#), [9261](#)
`\exp_stop_f` [30](#)
`\exp_stop_f:` . [1529](#), [1535](#), [2272](#), [4938](#), [5666](#)
`\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](#), [779](#), [1516](#),
[1872](#), [2381](#), [2471](#), [3191](#), [3899](#), [4195](#),
[4999](#), [5536](#), [5999](#), [6320](#), [6808](#), [7902](#),
[7922](#), [8415](#), [8998](#), [9647](#), [9764](#), [13128](#)
`\ExplFileDescription`
..... [113](#), [130](#), [334](#), [779](#), [1516](#),
[1872](#), [2381](#), [2471](#), [3191](#), [3899](#), [4195](#),
[4999](#), [5536](#), [5999](#), [6320](#), [6808](#), [7902](#),
[7922](#), [8415](#), [8998](#), [9647](#), [9764](#), [13128](#)
`\ExplFileName` [114](#), [128](#), [334](#), [779](#), [1516](#),
[1872](#), [2381](#), [2471](#), [3191](#), [3899](#), [4195](#),
[4999](#), [5536](#), [5999](#), [6320](#), [6808](#), [7902](#),
[7922](#), [8415](#), [8998](#), [9647](#), [9764](#), [13128](#)
`\ExplFileVersion`
..... [49](#), [115](#), [129](#), [334](#), [779](#), [1516](#),
[1872](#), [2381](#), [2471](#), [3191](#), [3899](#), [4195](#),
[4999](#), [5536](#), [5999](#), [6320](#), [6808](#), [7902](#),
[7922](#), [8415](#), [8998](#), [9647](#), [9764](#), [13128](#)
`\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` [2666](#), [2700](#), [2799](#)
`\fam` [366](#)
`\fi` [22](#), [44](#), [65](#),
[123](#), [139](#), [174](#), [194](#), [209](#), [231](#), [265](#), [405](#)
`\fi:` [782](#), [786](#), [977](#), [1045](#),
[1061](#), [1065](#), [1083](#), [1103](#), [1104](#), [1112](#),
[1118](#), [1131](#), [1132](#), [1140](#), [1146](#), [1204](#),
[1284](#), [1295](#), [1321](#), [1326](#), [1327](#), [1403](#),
[1466](#), [1471](#), [1507–1510](#), [1558](#), [1561](#),
[1568](#), [1569](#), [1777](#), [1857](#), [1909](#), [1923](#),
[1935](#), [1945](#), [1961](#), [1962](#), [1974](#), [1975](#),
[1987](#), [1996](#), [2251](#), [2253](#), [2255](#), [2257](#),
[2259](#), [2261](#), [2395](#), [2403](#), [2410](#), [2419](#),
[2429](#), [2437](#), [2450](#), [2459](#), [2591](#), [2596](#),

- 2601, 2606, 2613, 2619, 2624, 2629,
 2634, 2639, 2644, 2649, 2654, 2659,
 2681, 2687, 2694, 2731, 2742, 2753,
 2764, 2825, 2834, 2843, 2851, 2917,
 2925, 2947, 3211, 3222, 3233, 3248,
 3254, 3255, 3257, 3347, 3351, 3358,
 3366, 3374, 3382, 3390, 3398, 3406,
 3414, 3422, 3430, 3632, 3957, 3966,
 3976, 4105, 4447, 4459, 4472, 4482,
 4499, 4574, 4665, 4676, 4696, 4711,
 4720, 4729, 4748, 4760, 4765, 4766,
 4797, 4833, 5076, 5079, 5106, 5182,
 5186, 5207, 5321, 6126, 6151, 6187,
 6268, 6394, 6396, 6406, 8375, 8388,
 8389, 9845, 9882, 9883, 9915, 9920,
 9931, 9943, 9960, 9984, 9987, 9997,
 10007, 10032, 10089, 10131, 10169,
 10178, 10179, 10195, 10234, 10239,
 10250, 10251, 10259, 10287, 10301,
 10306, 10315, 10386, 10387, 10403,
 10407, 10422, 10427, 10448, 10451,
 10454, 10457, 10460, 10463, 10466,
 10469, 10472, 10487, 10490, 10493,
 10496, 10499, 10502, 10505, 10508,
 10511, 10532, 10543, 10568, 10579,
 10610, 10622, 10630–10632, 10636,
 10678, 10716, 10732, 10758, 10773,
 10787, 10790, 10850, 10853, 10958,
 10959, 10993, 10996, 11023, 11024,
 11039–11041, 11051, 11084, 11089,
 11099, 11103, 11111, 11112, 11233,
 11248, 11276, 11291, 11309, 11353,
 11361, 11377–11379, 11392, 11396,
 11417, 11418, 11427, 11436, 11446,
 11472, 11473, 11494, 11516, 11523,
 11536, 11549, 11570, 11591, 11603,
 11622, 11635, 11656, 11660, 11665,
 11666, 11693, 11698, 11702, 11717,
 11750, 11756, 11764, 11768, 11769,
 11771, 11792, 11814, 11821, 11832,
 11842, 11848, 11849, 11858, 11861,
 11888, 11891, 11916, 11986, 11998,
 12009, 12027, 12036, 12037, 12040,
 12052, 12081, 12086, 12087, 12109,
 12118, 12129, 12138, 12178, 12179,
 12191, 12199, 12200, 12243, 12249,
 12257, 12261, 12262, 12264, 12310,
 12318, 12337, 12355, 12356, 12368,
 12386, 12395, 12419, 12432, 12446,
 12468, 12474, 12479, 12487, 12493,
 12496, 12547, 12554, 12569, 12583,
 12608, 12614, 12616, 12655, 12669,
 12670, 12699, 12706, 12707, 12735,
 12741, 12745, 12746, 12828, 12836,
 12887, 12891, 12895, 12899, 12918,
 12919, 12930, 12933, 12934, 12950–
 12952, 12980–12984, 13012–13016
 \file_add_path:nN
 161, 9678, 9678, 9717, 9724
 \file_add_path_search:nN 9678, 9682, 9688
 \file_if_exist:n 9715, 9715
 \file_if_exist:nTF 161
 \file_input:n 162, 9722, 9722
 \file_list 162
 \file_list: 9746, 9746
 \file_path_include:n ... 162, 9739, 9739
 \file_path_remove:n 162, 9739, 9744
 \finalhyphendemerits 554
 \firstmark 452
 \firstmarks 678
 \floatingpenalty 599
 \font 365
 \fontcharhp 703
 \fontcharht 702
 \fontcharic 705
 \fontcharwd 704
 \fontdimen 632
 \fontname 456
 \fp_abs:c 10638
 \fp_abs:N 167, 10638, 10638, 10640
 \fp_abs_aux:NN 10638, 10638, 10639, 10642
 \fp_add:cn 10689
 \fp_add:Nn 168, 6605,
 7208, 7241, 7495, 10689, 10689, 10691
 \fp_add:NNNNNNNN
 .. 11075, 11075, 12413, 12465, 12551
 \fp_add_aux:NNn 10689, 10689, 10690, 10693
 \fp_add_core: . 10689, 10703, 10706, 10807
 \fp_add_difference: . 10689, 10715, 10760
 \fp_add_sum: 10689, 10713, 10743
 \fp_compare:n 13042, 13042
 \fp_compare:NNN 12838, 12855
 \fp_compare:nNn 12838, 12838
 \fp_compare:NNNT 7563
 \fp_compare:NNNTF
 6534, 6536, 6550, 6552,
 6557, 6670, 6672, 6715, 6735, 6753,
 6755, 6766, 6775, 6790, 7578, 7581

\fp_compare:nNnTF
..... [166](#), [7197](#), [13056](#), [13062](#),
[13068](#), [13074](#), [13080](#), [13086](#), [13092](#)
\fp_compare:nTF [167](#)
\fp_compare_<: [12838](#)
\fp_compare_<_aux: [12838](#)
\fp_compare_>: [12838](#)
\fp_compare_absolute_a<b: [12838](#)
\fp_compare_absolute_a>b: [12838](#)
\fp_compare_aux:N
..... [12838](#), [12853](#), [12864](#), [12866](#)
\fp_compare_aux_i:w . [13042](#), [13048](#), [13052](#)
\fp_compare_aux_ii:w [13042](#), [13055](#), [13058](#)
\fp_compare_aux_iii:w [13042](#), [13061](#), [13064](#)
\fp_compare_aux_iv:w [13042](#), [13067](#), [13070](#)
\fp_compare_aux_v:w . [13042](#), [13073](#), [13076](#)
\fp_compare_aux_vi:w [13042](#), [13079](#), [13082](#)
\fp_compare_aux_vii:w [13042](#), [13085](#), [13088](#)
\fp_const:cn [10056](#)
\fp_const:Nn [163](#), [10056](#), [10056](#), [10061](#)
\fp_cos:cn [11572](#)
\fp_cos:Nn
. [169](#), [6584](#), [7417](#), [11572](#), [11572](#), [11574](#)
\fp_cos_aux:Nn [11572](#), [11572](#), [11573](#), [11576](#)
\fp_cos_aux_i: [11572](#), [11602](#), [11612](#)
\fp_cos_aux_ii: [11572](#), [11615](#), [11645](#), [11850](#)
\fp_div:cn [10923](#)
\fp_div:Nn [168](#), [6581](#), [6665](#), [6669](#),
[6713](#), [6733](#), [7195](#), [7196](#), [7217](#), [7239](#),
[7414](#), [7550](#), [7553](#), [10923](#), [10923](#), [10925](#)
\fp_div_aux:Nn [10923](#), [10923](#), [10924](#), [10927](#)
\fp_div_divide: [10923](#), [11011](#), [11026](#), [11052](#)
\fp_div_divide_aux:
..... [10923](#), [11029](#), [11038](#), [11043](#)
\fp_div_integer:NNNN . [11218](#), [11218](#),
[11679](#), [11730](#), [11735](#), [12226](#), [12597](#)
\fp_div_internal:
. [10923](#), [10957](#), [10962](#), [12515](#), [12773](#)
\fp_div_loop:
. [10923](#), [10967](#), [11008](#), [11022](#), [11899](#)
\fp_div_loop_step:w [11015](#), [11069](#)
\fp_div_store: [10923](#),
[10965](#), [11012](#), [11054](#), [11058](#), [11897](#)
\fp_div_store_decimal: [10923](#), [11058](#), [11060](#)
\fp_div_store_integer:
..... [10923](#), [10965](#), [11055](#), [11897](#)
\fp_exp:cn [11966](#)
\fp_exp:Nn [169](#), [11966](#), [11966](#), [11968](#)
\fp_exp_aux: . [11966](#), [12022](#), [12032](#), [12042](#)
\fp_exp_aux:Nn [11966](#), [11966](#), [11967](#), [11970](#)
\fp_exp_const:cx [11966](#), [12034](#), [12074](#), [12206](#)
\fp_exp_const:Nx [11966](#), [12266](#), [12271](#), [12819](#)
\fp_exp_decimal: [11966](#), [12051](#), [12139](#), [12154](#)
\fp_exp_integer: . . . [11966](#), [12045](#), [12054](#)
\fp_exp_integer_const:n
..... [11966](#), [12077](#), [12083](#),
[12106](#), [12108](#), [12125](#), [12127](#), [12141](#)
\fp_exp_integer_const:nnnn
..... [11966](#), [12144](#), [12147](#), [12504](#)
\fp_exp_integer_tens:
. [11966](#), [12061](#), [12080](#), [12085](#), [12089](#)
\fp_exp_integer_units: [11966](#), [12119](#), [12121](#)
\fp_exp_internal:
..... [11966](#), [11997](#), [12014](#), [12820](#)
\fp_exp_overflow_msg:
..... [12026](#), [12039](#), [13102](#), [13108](#)
\fp_exp_Taylor: [11966](#), [12184](#), [12218](#), [12263](#)
\fp_extended_normalise: . . . [11235](#),
[11235](#), [11348](#), [12017](#), [12682](#), [12717](#)
\fp_extended_normalise_aux:NNNNNNNN
..... [11235](#)
\fp_extended_normalise_aux_i:
..... [11235](#), [11237](#), [11240](#), [11247](#)
\fp_extended_normalise_aux_i:w
..... [11235](#), [11245](#), [11250](#)
\fp_extended_normalise_aux_ii:
..... [11235](#), [11238](#), [11268](#), [11275](#)
\fp_extended_normalise_aux_ii:w
..... [11235](#), [11257](#), [11260](#)
\fp_extended_normalise_ii_aux:NNNNNNNN
..... [11273](#), [11278](#)
\fp_extended_normalise_output: [11301](#),
[11301](#), [11308](#), [12117](#), [12137](#), [12809](#)
\fp_extended_normalise_output_aux:N
..... [11301](#), [11329](#), [11331](#)
\fp_extended_normalise_output_aux_i:NNNNNNNN
..... [11301](#), [11306](#), [11311](#)
\fp_extended_normalise_output_aux_ii:NNNNNNNN
..... [11301](#), [11322](#), [11325](#)
\fp_gabs:c [10638](#)
\fp_gabs:N [167](#), [10638](#), [10639](#), [10641](#)
\fp_gadd:cn [10689](#)
\fp_gadd:Nn [168](#), [10689](#), [10690](#), [10692](#)
\fp_gcos:cn [11572](#)
\fp_gcos:Nn [170](#), [11572](#), [11573](#), [11575](#)
\fp_gdiv:cn [10923](#)
\fp_gdiv:Nn [168](#), [10923](#), [10924](#), [10926](#)
\fp_gexp:cn [11966](#)
\fp_gexp:Nn [169](#), [11966](#), [11967](#), [11969](#)
\fp_gln:cn [12284](#)

- \fp_gln:Nn [169](#), [12284](#), [12285](#), [12287](#)
- \fp_gmul:cn [10810](#)
- \fp_gmul:Nn [168](#), [10810](#), [10811](#), [10813](#)
- \fp_gneg:c [10663](#)
- \fp_gneg:N [167](#), [10663](#), [10664](#), [10666](#)
- \fp_gpow:cn [12618](#)
- \fp_gpow:Nn [169](#), [12618](#), [12619](#), [12621](#)
- \fp_ground_figures:cn [10519](#)
- \fp_ground_figures:Nn
 [165](#), [10519](#), [10522](#), [10524](#)
- \fp_ground_places:cn [10554](#)
- \fp_ground_places:Nn
 [166](#), [10554](#), [10557](#), [10559](#)
- \fp_gset:cn [10068](#)
- \fp_gset:Nn [164](#), [10059](#), [10068](#), [10069](#), [10101](#)
- \fp_gset_eq:cc [10152](#), [10159](#)
- \fp_gset_eq:cN [10152](#), [10157](#)
- \fp_gset_eq:Nc [10152](#), [10158](#)
- \fp_gset_eq:NN [164](#), [10152](#), [10156](#)
- \fp_gset_from_dim:cn [10102](#)
- \fp_gset_from_dim:Nn
 [164](#), [10102](#), [10104](#), [10149](#)
- \fp_gsin:cn [11475](#)
- \fp_gsin:Nn [169](#), [11475](#), [11476](#), [11478](#)
- \fp_gsub:cn [10792](#)
- \fp_gsub:Nn [168](#), [10792](#), [10793](#), [10795](#)
- \fp_gtan:cn [11773](#)
- \fp_gtan:Nn [170](#), [11773](#), [11774](#), [11776](#)
- \fp_gzero:c [10062](#)
- \fp_gzero:N [164](#), [10062](#), [10064](#), [10067](#)
- \fp_if_undefined:N [12822](#), [12822](#)
- \fp_if_undefined:NTF [166](#)
- \fp_if_zero:N [12830](#), [12830](#)
- \fp_if_zero:NTF [166](#)
- \fp_level_input_exponents:
 [9991](#), [9991](#), [10708](#)
- \fp_level_input_exponents_a:
 [9991](#), [9994](#), [9999](#), [10006](#)
- \fp_level_input_exponents_a:NNNNNNNN
 [9991](#), [10004](#), [10009](#)
- \fp_level_input_exponents_b:
 [9991](#), [9996](#), [10024](#), [10031](#)
- \fp_level_input_exponents_b:NNNNNNNN
 [9991](#), [10029](#), [10034](#)
- \fp_ln:cn [12284](#)
- \fp_ln:Nn [169](#), [12284](#), [12284](#), [12286](#)
- \fp_ln_aux: [12284](#), [12302](#), [12321](#)
- \fp_ln_aux:NNn [12284](#), [12284](#), [12285](#), [12288](#)
- \fp_ln_const:nn [12401](#), [12411](#), [12462](#), [12501](#)
- \fp_ln_error_msg:
 [12309](#), [12317](#), [13110](#), [13113](#)
- \fp_ln_exponent: [12284](#), [12336](#), [12345](#)
- \fp_ln_exponent_tens: [12284](#)
- \fp_ln_exponent_tens:NN [12388](#), [12398](#)
- \fp_ln_exponent_units: [12284](#), [12396](#), [12408](#)
- \fp_ln_fixed: [12284](#), [12516](#), [12561](#), [12568](#)
- \fp_ln_fixed_aux:NNNNNNNN
 [12284](#), [12566](#), [12571](#)
- \fp_ln_integer_const:nn [12284](#)
- \fp_ln_internal: [12284](#), [12347](#), [12379](#), [12781](#)
- \fp_ln_mantissa: [12284](#), [12420](#), [12456](#)
- \fp_ln_mantissa_aux:
 [12284](#), [12460](#), [12481](#), [12486](#)
- \fp_ln_mantissa_divide_two:
 [12284](#), [12485](#), [12489](#)
- \fp_ln_normalise: [12284](#),
 [12412](#), [12422](#), [12429](#), [12463](#), [12549](#)
- \fp_ln_normalise_aux:NNNNNNNN
 [12427](#), [12434](#)
- \fp_ln_nornalise_aux:NNNNNNNN [12284](#)
- \fp_ln_Taylor: [12284](#), [12478](#), [12507](#)
- \fp_ln_Taylor_aux:
 [12284](#), [12529](#), [12586](#), [12615](#)
- \fp_mul:cn [10810](#)
- \fp_mul:Nn [168](#),
 [6582](#), [6592](#), [6593](#), [6603](#), [6604](#), [7206](#),
 [7211](#), [7240](#), [7415](#), [7488](#), [7489](#), [7493](#),
 [7494](#), [7591](#), [7594](#), [10810](#), [10810](#), [10812](#)
- \fp_mul:NNNNNN [11114](#), [11114](#), [11675](#),
 [11722](#), [11726](#), [12222](#), [12524](#), [12589](#)
- \fp_mul:NNNNNNNN [11158](#),
 [11158](#), [12110](#), [12130](#), [12185](#), [12798](#)
- \fp_mul_aux:NNn [10810](#), [10810](#), [10811](#), [10814](#)
- \fp_mul_end_level:
 [10810](#), [10881](#), [10885](#), [10888](#),
 [10892](#), [10912](#), [11138](#), [11143](#), [11147](#),
 [11152](#), [11154](#), [11155](#), [11184](#), [11191](#),
 [11197](#), [11204](#), [11208](#), [11211](#), [11215](#)
- \fp_mul_end_level:NNNNNNNN
 [10810](#), [10916](#), [10918](#)
- \fp_mul_internal: [10810](#), [10824](#), [10865](#)
- \fp_mul_product:NN [10873](#)–
 [10875](#), [10877](#)–[10880](#), [10882](#)–[10884](#),
 [10886](#), [10887](#), [10891](#), [10907](#), [11126](#)–
 [11131](#), [11133](#)–[11137](#), [11139](#)–[11142](#),
 [11144](#)–[11146](#), [11150](#), [11151](#), [11153](#),
 [11170](#)–[11175](#), [11177](#)–[11183](#), [11185](#)–
 [11190](#), [11192](#)–[11196](#), [11200](#)–[11203](#),
 [11205](#)–[11207](#), [11209](#), [11210](#), [11214](#)

- \fp_mul_split:NNNN [10810](#), [10867](#),
[10869](#), [10895](#), [11116](#), [11118](#), [11120](#),
[11122](#), [11160](#), [11162](#), [11164](#), [11166](#)
- \fp_mul_split:w [10810](#)
- \fp_mul_split_aux:w [10898](#), [10904](#)
- \fp_neg:c [10663](#)
- \fp_neg:N [167](#), [10663](#), [10663](#), [10665](#)
- \fp_neg:NN [10663](#)
- \fp_neg_aux:NN [10663](#), [10664](#), [10667](#)
- \fp_new:c [10050](#)
- \fp_new:N [163](#), [6510](#)–
[6512](#), [6522](#)–[6526](#), [6652](#), [6653](#), [6813](#),
[6832](#)–[6836](#), [6842](#), [6843](#), [6848](#)–[6851](#),
[7540](#), [7541](#), [10050](#), [10050](#), [10055](#), [10058](#)
- \fp_overflow_msg: [9919](#), [9986](#), [13094](#), [13100](#)
- \fp_pow:cn [12618](#)
- \fp_pow:Nn [168](#), [12618](#), [12618](#), [12620](#)
- \fp_pow_aux:NNn [12618](#), [12618](#), [12619](#), [12622](#)
- \fp_pow_aux_i: [12618](#), [12668](#), [12673](#)
- \fp_pow_aux_ii: [12618](#), [12677](#), [12691](#), [12709](#)
- \fp_pow_aux_iii: ... [12618](#), [12734](#), [12763](#)
- \fp_pow_aux_iv: [12618](#), [12712](#),
[12726](#), [12740](#), [12744](#), [12766](#), [12775](#)
- \fp_pow_negative: [12618](#)
- \fp_pow_positive: [12618](#)
- \fp_read:N [9837](#), [9837](#), [10528](#),
[10563](#), [10645](#), [10670](#), [10696](#), [10799](#),
[10817](#), [10930](#), [12625](#), [12858](#), [12863](#)
- \fp_read_aux:w [9837](#), [9838](#), [9839](#)
- \fp_round: ... [10531](#), [10567](#), [10590](#), [10590](#)
- \fp_round_aux:NNNNNNNN
..... [10590](#), [10597](#), [10599](#)
- \fp_round_figures:cn [10519](#)
- \fp_round_figures:Nn
..... [165](#), [10519](#), [10519](#), [10521](#)
- \fp_round_figures_aux:NNn
..... [10519](#), [10520](#), [10523](#), [10525](#)
- \fp_round_loop:N [10590](#), [10601](#), [10612](#), [10635](#)
- \fp_round_places:cn [10554](#)
- \fp_round_places:Nn
..... [166](#), [10554](#), [10554](#), [10556](#)
- \fp_round_places_aux:NNn
..... [10554](#), [10555](#), [10558](#), [10560](#)
- \fp_set:cn [10068](#)
- \fp_set:Nn [164](#), [6532](#), [6747](#), [6748](#),
[7413](#), [7576](#), [7577](#), [10068](#), [10068](#), [10100](#)
- \fp_set_aux:NNn [10068](#), [10068](#)–[10070](#)
- \fp_set_eq:cc [10152](#), [10155](#)
- \fp_set_eq:cN [10152](#), [10153](#)
- \fp_set_eq:Nc [10152](#), [10154](#)
- \fp_set_eq:NN [163](#), [6580](#),
[6590](#), [6591](#), [6601](#), [6602](#), [6714](#), [6734](#),
[7486](#), [7487](#), [7491](#), [7492](#), [10152](#), [10152](#)
- \fp_set_from_dim:cn [10102](#)
- \fp_set_from_dim:Nn [164](#), [6588](#),
[6589](#), [6599](#), [6600](#), [6663](#), [6664](#), [6666](#),
[6667](#), [6710](#), [6711](#), [6731](#), [6732](#), [7191](#)–
[7194](#), [7201](#), [7202](#), [7205](#), [7210](#), [7233](#)–
[7237](#), [7484](#), [7485](#), [7548](#), [7549](#), [7551](#),
[7552](#), [7590](#), [7593](#), [10102](#), [10102](#), [10148](#)
- \fp_set_from_dim_aux:NNn
..... [10102](#), [10103](#), [10105](#), [10106](#)
- \fp_set_from_dim_aux:w
... [10102](#), [10113](#), [10142](#), [10144](#), [10147](#)
- \fp_show:c [10160](#), [10161](#)
- \fp_show:N [164](#), [10160](#), [10160](#)
- \fp_sin:cn [11475](#)
- \fp_sin:Nn
. [169](#), [6583](#), [7416](#), [11475](#), [11475](#), [11477](#)
- \fp_sin_aux:NNn [11475](#), [11475](#), [11476](#), [11479](#)
- \fp_sin_aux_i: [11475](#), [11515](#), [11526](#)
- \fp_sin_aux_ii: [11475](#), [11529](#), [11559](#), [11873](#)
- \fp_split:Nn [9850](#),
[9850](#), [10073](#), [10111](#), [10697](#), [10800](#),
[10818](#), [10931](#), [11482](#), [11579](#), [11780](#),
[11973](#), [12291](#), [12630](#), [12841](#), [12847](#)
- \fp_split_aux_i:w [9850](#), [9889](#), [9893](#)
- \fp_split_aux_ii:w [9850](#), [9894](#), [9895](#)
- \fp_split_aux_iii:w [9850](#), [9896](#), [9897](#)
- \fp_split_decimal:w [9850](#), [9900](#), [9903](#)
- \fp_split_decimal_aux:w [9850](#), [9904](#), [9905](#)
- \fp_split_exponent: [9850](#)
- \fp_split_exponent:w [9858](#), [9885](#)
- \fp_split_sign: [9850](#), [9856](#), [9860](#), [9871](#), [9881](#)
- \fp_standardise:NNNN [9922](#),
[9922](#), [10074](#), [10116](#), [10698](#), [10718](#),
[10801](#), [10819](#), [10829](#), [10932](#), [10972](#),
[11483](#), [11537](#), [11580](#), [11623](#), [11781](#),
[11868](#), [11877](#), [11904](#), [11974](#), [12201](#),
[12292](#), [12357](#), [12631](#), [12842](#), [12848](#)
- \fp_standardise_aux: [9922](#), [9936](#),
[9942](#), [9952](#), [9953](#), [9959](#), [9976](#), [9989](#)
- \fp_standardise_aux:NNNN [9922](#), [9930](#), [9934](#)
- \fp_standardise_aux:w
... [9922](#), [9940](#), [9946](#), [9958](#), [9963](#), [9990](#)
- \fp_sub:cn [10792](#)
- \fp_sub:Nn [168](#), [6594](#), [7203](#), [7213](#),
[7215](#), [7238](#), [7490](#), [10792](#), [10792](#), [10794](#)
- \fp_sub:NNNNNNNN [11091](#), [11091](#),
[11430](#), [11451](#), [11462](#), [12467](#), [12553](#)

\fp_sub_aux:NNn [10792](#), [10792](#), [10793](#), [10796](#)
\fp_tan:cn [11773](#)
\fp_tan:Nn [170](#), [11773](#), [11773](#), [11775](#)
\fp_tan_aux:NNn [11773](#), [11773](#), [11774](#), [11777](#)
\fp_tan_aux_i: [11773](#), [11813](#), [11824](#)
\fp_tan_aux_ii: [11773](#), [11827](#), [11834](#)
\fp_tan_aux_iii: [11773](#), [11857](#), [11860](#), [11863](#)
\fp_tan_aux_iv: [11773](#), [11887](#), [11890](#), [11893](#)
\fp_tmp:w [10049](#),
[10049](#), [10080](#), [10098](#), [10122](#), [10140](#),
[10534](#), [10552](#), [10570](#), [10588](#), [10647](#),
[10661](#), [10704](#), [10723](#), [10808](#), [10834](#),
[10863](#), [10941](#), [10951](#), [10960](#), [10977](#),
[11504](#), [11517](#), [11524](#), [11604](#), [11610](#),
[11802](#), [11815](#), [11822](#), [11999](#), [12012](#),
[12304](#), [12312](#), [12319](#), [12338](#), [12530](#),
[12541](#), [12644](#), [12650](#), [12661](#), [12671](#),
[12694](#), [12701](#), [12747](#), [12782](#), [12796](#)
\fp_to_dim:c [10225](#)
\fp_to_dim:N
[165](#), [6595](#), [6606](#), [7220](#), [7242](#), [7496](#),
[7497](#), [7592](#), [7595](#), [10225](#), [10225](#), [10226](#)
\fp_to_int:c [10227](#)
\fp_to_int:N [165](#), [10227](#), [10227](#), [10229](#)
\fp_to_int_aux:w ... [10227](#), [10228](#), [10230](#)
\fp_to_int_large:w .. [10227](#), [10238](#), [10253](#)
\fp_to_int_large_aux:nnn
[10227](#), [10265](#), [10267](#), [10269](#), [10271](#),
[10273](#), [10275](#), [10277](#), [10279](#), [10281](#)
\fp_to_int_large_aux_1:w [10227](#)
\fp_to_int_large_aux_2:w [10227](#)
\fp_to_int_large_aux_3:w [10227](#)
\fp_to_int_large_aux_4:w [10227](#)
\fp_to_int_large_aux_5:w [10227](#)
\fp_to_int_large_aux_6:w [10227](#)
\fp_to_int_large_aux_7:w [10227](#)
\fp_to_int_large_aux_8:w [10227](#)
\fp_to_int_large_aux_i:w
..... [10227](#), [10256](#), [10262](#)
\fp_to_int_large_aux_ii:w
..... [10227](#), [10258](#), [10289](#)
\fp_to_int_none:w [10227](#)
\fp_to_int_small:w .. [10227](#), [10236](#), [10242](#)
\fp_to_tl:c [10294](#)
\fp_to_tl:N [165](#), [10294](#), [10294](#), [10296](#)
\fp_to_tl_aux:w [10294](#), [10295](#), [10297](#)
\fp_to_tl_large:w .. [10294](#), [10305](#), [10309](#)
\fp_to_tl_large_0:w [10294](#)
\fp_to_tl_large_1:w [10294](#)
\fp_to_tl_large_2:w [10294](#)
\fp_to_tl_large_3:w [10294](#)
\fp_to_tl_large_4:w [10294](#)
\fp_to_tl_large_5:w [10294](#)
\fp_to_tl_large_6:w [10294](#)
\fp_to_tl_large_7:w [10294](#)
\fp_to_tl_large_8:w [10294](#)
\fp_to_tl_large_8_aux:w [10294](#)
\fp_to_tl_large_9:w [10294](#)
\fp_to_tl_large_aux_i:w
..... [10294](#), [10312](#), [10318](#)
\fp_to_tl_large_aux_ii:w
..... [10294](#), [10314](#), [10320](#)
\fp_to_tl_large_zeros:NNNNNNNN
..... [10294](#), [10323](#), [10329](#),
[10334](#), [10339](#), [10344](#), [10349](#), [10354](#),
[10359](#), [10364](#), [10374](#), [10432](#), [10435](#)
\fp_to_tl_small:w .. [10294](#), [10303](#), [10377](#)
\fp_to_tl_small_aux:w [10294](#), [10385](#), [10429](#)
\fp_to_tl_small_one:w [10294](#), [10380](#), [10390](#)
\fp_to_tl_small_two:w [10294](#), [10383](#), [10409](#)
\fp_to_tl_small_zeros:NNNNNNNN
..... [10294](#),
[10397](#), [10406](#), [10416](#), [10426](#), [10474](#)
\fp_trig_calc_cos: [11565](#),
[11567](#), [11649](#), [11655](#), [11668](#), [11668](#)
\fp_trig_calc_sin: [11563](#),
[11569](#), [11651](#), [11653](#), [11668](#), [11704](#)
\fp_trig_calc_Taylor:
.. [11668](#), [11701](#), [11716](#), [11719](#), [11770](#)
\fp_trig_normalise:
.. [11344](#), [11344](#), [11528](#), [11614](#), [11836](#)
\fp_trig_normalise_aux:
.. [11344](#), [11349](#), [11363](#), [11368](#), [11376](#)
\fp_trig_octant: ... [11354](#), [11420](#), [11420](#)
\fp_trig_octant_aux_i:
.. [11420](#), [11423](#), [11438](#), [11458](#), [11471](#)
\fp_trig_octant_aux_ii:
..... [11420](#), [11445](#), [11448](#)
\fp_trig_overflow_msg:
..... [11360](#), [11831](#), [13116](#), [13122](#)
\fp_trig_sub:NNN [11344](#), [11366](#), [11372](#), [11381](#)
\fp_use:c [10162](#)
\fp_use:N [164](#), [6689](#), [6691](#),
[6695](#), [6697](#), [10162](#), [10162](#), [10164](#), [10225](#)
\fp_use_aux:w [10162](#), [10163](#), [10165](#)
\fp_use_i_to_iix:NNNNNNNN
..... [10294](#), [10394](#), [10399](#), [10517](#)
\fp_use_i_to_vii:NNNNNNNN
..... [10294](#), [10413](#), [10418](#), [10515](#)

- \fp_use_iix_ix:NNNNNNNNN [10294](#), [10411](#), [10513](#)
- \fp_use_ix:NNNNNNNNN [10294](#), [10392](#), [10514](#)
- \fp_use_large:w [10162](#), [10171](#), [10189](#)
- \fp_use_large_aux_1:w [10162](#)
- \fp_use_large_aux_2:w [10162](#)
- \fp_use_large_aux_3:w [10162](#)
- \fp_use_large_aux_4:w [10162](#)
- \fp_use_large_aux_5:w [10162](#)
- \fp_use_large_aux_6:w [10162](#)
- \fp_use_large_aux_7:w [10162](#)
- \fp_use_large_aux_8:w [10162](#)
- \fp_use_large_aux_i:w [10162](#), [10192](#), [10198](#)
- \fp_use_large_aux_ii:w [10162](#), [10194](#), [10219](#)
- \fp_use_none:w [10162](#), [10177](#), [10182](#)
- \fp_use_small:w [10162](#), [10175](#), [10183](#)
- \fp_zero:c [10062](#)
- \fp_zero:N [164](#), [10062](#), [10062](#), [10066](#)
- \frozen@everydisplay [764](#)
- \frozen@everymath [765](#)
- \futurelet [361](#)
- G**
- \G [2705](#)
- \g [2264](#)
- \g_cctab_allocate_int [13154](#),
[13154](#), [13155](#), [13162](#), [13164](#), [13166](#)
- \g_cctab_stack_int [13154](#), [13156](#),
[13192](#), [13193](#), [13195](#), [13196](#), [13200](#)
- \g_cctab_stack_seq
..... [13154](#), [13157](#), [13190](#), [13201](#)
- \g_clist_map_inline_int
..... [5784](#), [5784](#), [5787](#), [5788](#),
[5792](#), [5793](#), [5797](#), [5798](#), [5802](#), [5803](#)
- \g_file_current_name_tl
..... [161](#), [9650](#), [9650](#),
[9655](#), [9659](#), [9667](#), [9733](#), [9734](#), [9736](#)
- \g_file_record_seq [162](#), [9662](#),
[9662](#), [9667](#), [9728](#), [9748](#), [9750](#), [9757](#)
- \g_file_stack_seq
..... [162](#), [9661](#), [9661](#), [9733](#), [9736](#)
- \g_file_test_stream [9678](#),
[9680](#), [9681](#), [9684](#), [9702](#), [9703](#), [9713](#)
- \g_ior_streams_prop [7950](#), [7951](#),
[7956](#), [7991](#), [8093](#), [8108](#), [8130](#), [8138](#)
- \g_ior_tmp_stream [8026](#), [8072](#), [8073](#), [8076](#)
- \g_iow_streams_prop
.... [7950](#), [7950](#), [7953](#)–[7955](#), [8004](#),
[8012](#), [8020](#), [8056](#), [8121](#), [8150](#), [8158](#)
- \g_iow_tmp_stream [8026](#), [8035](#), [8036](#), [8039](#)
- \g_keyval_level_int [9001](#), [9001](#),
[9048](#), [9070](#), [9094](#), [9096](#), [9098](#), [9100](#)
- \g_peek_token [54](#), [2853](#), [2854](#), [2864](#)
- \g_prg_stepwise_level_int
.. [2225](#), [2225](#), [2230](#), [2236](#), [2246](#), [2248](#)
- \g_prop_map_inline_int
.. [6193](#), [6193](#), [6196](#), [6197](#), [6200](#), [6201](#)
- \g_seq_nesting_depth_int
.. [3853](#), [5222](#), [5234](#), [5236](#), [5240](#), [5242](#)
- \g_tl_inline_level_int
..... [3854](#), [4531](#), [4533](#), [4534](#),
[4537](#), [4539](#), [4543](#), [4544](#), [4547](#), [4549](#)
- \g_tmpa_bool [36](#), [1915](#), [1916](#)
- \g_tmpa_clist [112](#), [5872](#)
- \g_tmpa_dim [75](#), [4059](#), [4062](#)
- \g_tmpa_int [67](#), [3848](#), [3851](#)
- \g_tmpa_skip [77](#), [4133](#), [4136](#)
- \g_tmpa_tl [93](#), [4785](#), [4785](#), [5870](#)
- \g_tmpb_clist [112](#), [5873](#)
- \g_tmpb_dim [75](#), [4059](#), [4063](#)
- \g_tmpb_int [67](#), [3848](#), [3852](#)
- \g_tmpb_skip [77](#), [4133](#), [4137](#)
- \g_tmpb_tl [93](#), [4785](#), [4786](#), [5870](#)
- \gdef [352](#)
- \GetIdInfo [6](#), [97](#), [98](#)
- \GetIdInfoAuxCVS [97](#), [136](#), [141](#)
- \GetIdInfoAuxI [97](#), [102](#), [104](#)
- \GetIdInfoAuxII [97](#), [121](#), [126](#)
- \GetIdInfoAuxIII [97](#), [131](#), [133](#)
- \GetIdInfoAuxSVN [97](#), [138](#), [143](#)
- \GetIdInfoFull [97](#)
- \global [336](#), [367](#)
- \globaldefs [371](#)
- \glueexpr [711](#)
- \glueshrink [714](#)
- \glueshrinkorder [716](#)
- \gluestretch [713](#)
- \gluestretchorder [715](#)
- \gluetomu [717](#)
- \group_align_safe_begin [41](#)
- \group_align_safe_begin:
..... [1927](#), [2258](#), [2258](#), [2885](#), [2903](#)
- \group_align_safe_end [41](#)
- \group_align_safe_end: .. [2024](#), [2025](#),
[2258](#), [2260](#), [2867](#), [2877](#), [2882](#), [2900](#)
- \group_begin [9](#)
- \group_begin: [809](#), [810](#), [853](#), [1066](#), [1816](#),
[2263](#), [2277](#), [2571](#), [2584](#), [2608](#), [2661](#),
[2698](#), [2795](#), [2961](#), [3058](#), [3065](#), [4302](#),
[4319](#), [4346](#), [4359](#), [4490](#), [5099](#), [6531](#),

- 6658, 6705, 6726, 6746, 7905, 8208,
8213, 8242, 8500, 8547, 8951, 9006,
9016, 10072, 10108, 10527, 10562,
10644, 10669, 10695, 10798, 10816,
10929, 11481, 11578, 11779, 11972,
12290, 12509, 12624, 12681, 12715,
12777, 12840, 12857, 13044, 13221
\group_end 9
\group_end: 809,
811, 856, 1071, 1828, 2268, 2282,
2583, 2587, 2615, 2669, 2709, 2801,
3026, 3067, 3076, 4309, 4326, 4352,
4365, 4494, 4497, 5110, 6541, 6677,
6718, 6738, 6760, 7909, 8212, 8232,
8276, 8505, 8553, 8974, 9013, 9024,
10082, 10124, 10536, 10572, 10649,
10686, 10725, 10836, 10943, 10953,
10979, 11506, 11519, 11606, 11804,
11817, 12001, 12306, 12314, 12340,
12532, 12646, 12652, 12663, 12687,
12693, 12696, 12703, 12720, 12728,
12737, 12749, 12784, 12871, 12882,
12885, 12889, 12893, 12897, 12909,
12913, 12916, 12925, 12928, 12939,
12943, 12957, 12961, 12965, 12970,
12975, 12978, 12989, 12993, 12997,
13002, 13007, 13010, 13047, 13224
\group_execute_after:N 1502
\group_insert_after:N . 9, 814, 814, 1502
- H**
- \H 2705
\halign 378
\hangafter 556
\hangindent 557
\hbadness 618
\hbox 613
\hbox:n . 126, 6433, 6433, 6565, 7685, 7740
\hbox_gset:cn 6434
\hbox_gset:cw 6444, 6456
\hbox_gset:Nn 126, 6434, 6435, 6437
\hbox_gset:Nw . 127, 6444, 6446, 6449, 6455
\hbox_gset_end 127
\hbox_gset_end: 6444, 6451, 6457
\hbox_gset_inline_begin:c ... 6452, 6456
\hbox_gset_inline_begin:N ... 6452, 6455
\hbox_gset_inline_end: 6452, 6457
\hbox_gset_to_wd:cnn 6438
\hbox_gset_to_wd:Nnn 126, 6438, 6440, 6443
\hbox_overlap_left:n ... 127, 6461, 6461
\hbox_overlap_right:n 127, 6461, 6463, 6785
\hbox_set:cn 6434
\hbox_set:cw 6444, 6453
\hbox_set:Nn
126, 6434, 6434–6436, 6529, 6561,
6562, 6656, 6703, 6724, 6744, 6782,
6900, 6997, 7248, 7319, 7428, 7831
\hbox_set:Nw
127, 6444, 6444, 6447, 6448, 6452, 6944
\hbox_set_end 127
\hbox_set_end: ... 6444, 6450, 6454, 6948
\hbox_set_inline_begin:c 6452, 6453
\hbox_set_inline_begin:N 6452, 6452
\hbox_set_inline_end: 6452, 6454
\hbox_set_to_wd:cnn 6438
\hbox_set_to_wd:Nnn
..... 126, 6438, 6438, 6441, 6442
\hbox_to_wd:nn 126, 6458, 6458, 6792
\hbox_to_zero:n 126, 6458, 6460, 6462, 6464
\hbox_unpack:c 6465
\hbox_unpack:N
... 127, 6465, 6465, 6467, 7252, 7401
\hbox_unpack_clear:c 6465
\hbox_unpack_clear:N 127, 6465, 6466, 6468
\hcoffin_set:cn 6896
\hcoffin_set:cw 6940
\hcoffin_set:Nn 131, 6896,
6896, 6912, 7682, 7694, 7737, 7778
\hcoffin_set:Nw ... 131, 6940, 6940, 6956
\hcoffin_set_end 131
\hcoffin_set_end: 6940, 6945, 6955
\Height 7031, 7033, 7037, 7041, 7048
\hfil 521
\hfill 523
\hfilneg 522
\hfuzz 620
\hoffset 595
\holdinginserts 598
\hrule 534
\hsize 559
\hskip 524
\hss 525
\ht 663
\hyphenation 649
\hyphenchar 633
\hyphenpenalty 551
- I**
- \I 2705
\if 184, 387

- \if:w 22, 782,
788, 975, 1043, 1059, 1775, 2838,
2943, 3344, 9841, 10167, 10232, 10299
- \if_bool:N 23, 782, 789, 1905
- \if_box_empty:N ... 130, 6390, 6392, 6406
- \if_case:w 69,
1307, 3194, 3199, 3604, 11561, 11647
- \if_catcode:w 22, 782,
792, 2590, 2595, 2600, 2605, 2612,
2618, 2623, 2628, 2633, 2638, 2643,
2653, 2686, 2912, 4684, 4722, 8370
- \if_charcode:w
. 22, 782, 791, 2658, 4662, 4668, 4715
- \if_cs_exist:N 23, 798, 798, 1099, 1127, 2847
- \if_cs_exist:w 798,
799, 1108, 1136, 1280, 4570, 11510,
11600, 11808, 11995, 12004, 12334
- \if_dim:w 80, 3902, 3902, 3956, 3981–3987
- \if_eof:w 141, 7925, 7925, 8384
- \if_false 22
- \if_false: 782, 783, 2259,
4765, 4766, 5076, 5079, 5182, 5186
- \if_hbox:N 130, 6390, 6390, 6394
- \if_int_compare:w 68, 812, 812, 1464,
1470, 2259, 2261, 2407, 2414, 2445,
2454, 2677, 3194, 3209, 3217, 3228,
3239, 3243, 3244, 3250, 3354, 3362,
3370, 3378, 3386, 3394, 3402, 3410,
4099, 4752, 4791, 8381, 9862, 9873,
9908, 9916, 9924, 9938, 9955, 9977,
9978, 9993, 10001, 10026, 10085,
10127, 10170, 10173, 10191, 10235,
10244, 10246, 10255, 10283, 10302,
10311, 10379, 10382, 10392, 10393,
10411, 10412, 10437–10445, 10476–
10484, 10530, 10539, 10566, 10575,
10605, 10614, 10618, 10627, 10628,
10634, 10674, 10709, 10728, 10754,
10770, 10774, 10776, 10839, 10843,
10937, 10947, 10982, 10986, 11017,
11020, 11028, 11031, 11033, 11048,
11080, 11085, 11096, 11100, 11104,
11105, 11230, 11242, 11270, 11281,
11303, 11346, 11350, 11365, 11370,
11371, 11389, 11393, 11397, 11399,
11424, 11440, 11450, 11460, 11490,
11503, 11530, 11545, 11587, 11616,
11631, 11657, 11658, 11662, 11670,
11684, 11685, 11707, 11740, 11741,
11745, 11751, 11761, 11765, 11788,
11801, 11826, 11837, 11838, 11844,
11851, 11852, 11882, 11883, 11912,
11982, 12016, 12018, 12019, 12029,
12044, 12056, 12072, 12073, 12095,
12105, 12123, 12124, 12156, 12157,
12163, 12192, 12195, 12230, 12234,
12238, 12244, 12254, 12258, 12297,
12298, 12348, 12351, 12364, 12381,
12387, 12410, 12424, 12436, 12461,
12464, 12475, 12483, 12543, 12550,
12563, 12573, 12593, 12603, 12609,
12636, 12640, 12657, 12675, 12680,
12683, 12711, 12714, 12718, 12719,
12877–12880, 12903, 12906, 12912,
12921, 12924, 12938, 12942, 12946,
12956, 12960, 12964, 12968, 12973,
12988, 12992, 12996, 13000, 13005
- \if_int_odd:w 69, 3194,
3198, 3418, 3426, 11428, 12491, 12494
- \if_meaning:w 22, 782,
793, 1079, 1096, 1114, 1124, 1142,
1402, 1555, 1556, 1855, 1935, 1945,
1954, 1957, 1967, 1970, 1983, 1992,
2393, 2399, 2425, 2433, 2648, 2693,
2726, 2737, 2748, 2759, 2821, 2921,
3351, 3976, 4443, 4455, 4467, 4478,
4493, 4707, 4831, 5104, 5204, 5318,
6122, 6147, 6185, 6266, 12824, 12832
- \if_mode_horizontal 23
- \if_mode_horizontal: 794, 795, 2253
- \if_mode_inner 23
- \if_mode_inner: 794, 797, 2255
- \if_mode_math 23
- \if_mode_math: 794, 794, 2257
- \if_mode_vertical 23
- \if_mode_vertical: 794, 796, 2251
- \if_num:w 68, 2829, 3194, 3197
- \if_predicate:w
..... 22, 782, 790, 1291, 1919, 4736
- \if_true 22
- \if_true: 782, 782
- \if_vbox:N 130, 6390, 6391, 6396
- \ifcase 388
- \ifcat 389
- \ifcsname 672
- \ifdefined 671
- \ifdim 392
- \ifeof 393
- \iffalse 398
- \iffontchar 701

<code>\ifhbox</code>	394	<code>\int_decr:c</code>	3319
<code>\ifhmode</code>	400	<code>\int_decr:N</code>	61, 3319, 3321, 3326, 3328
<code>\ifinner</code>	403	<code>\int_div_round:nn</code>	59, 3236, 3261
<code>\ifmmode</code>	401	<code>\int_div_truncate:nn</code>	
<code>\ifnum</code>	390	60, 3236, 3236, 3265, 3506, 3596
<code>\ifodd</code>	169, 205, 391	<code>\int_do_until:nn</code> ...	63, 3432, 3454, 3458
<code>\iftrue</code>	399	<code>\int_do_until:nNnn</code> ..	63, 3460, 3482, 3486
<code>\ifvbox</code>	395	<code>\int_do_while:nn</code>	63, 3432, 3448
<code>\ifvmode</code>	402	<code>\int_do_while:nNnn</code>	
<code>\ifvoid</code>	396	63, 3452, 3460, 3476, 3480
<code>\ifx</code>	13, 62, 108, 135, 229, 233, 397	<code>\int_eval:n</code>	
<code>\ignorespaces</code>	445	59, 1333, 2098, 2204, 2213, 2222,
<code>\immediate</code>	407	3200, 3201, 3204, 3261, 3488, 3574,
<code>\indent</code>	541	3643, 3653, 3712, 3726, 3730, 3733,
<code>\initcatcodetable</code>	757	3748, 3757, 4578, 4583, 4897, 5386,
<code>\input</code>	415	5398, 5876, 5885, 5908, 5921, 5943
<code>\input@path</code>	9692, 9695, 9710	<code>\int_eval:w</code>	69, 1307, 2160, 2475,
<code>\inputlineno</code>	417	2477, 2479, 2545, 2547, 2549, 2551,
<code>\insert</code>	597	2553, 2555, 2557, 2559, 2561, 2563,
<code>\insertpenalties</code>	600	2565, 2567, 3194, 3195, 3201, 3204,
<code>\int_abs:n</code>	59, 3206, 3206	3209, 3212, 3216, 3218, 3227, 3229,
<code>\int_add:cn</code>	3307	3238, 3239, 3243, 3244, 3250, 3264,
<code>\int_add:Nn</code>	61, 3307,	3288, 3308, 3310, 3332, 3339, 3354,
.....	3307, 3312, 3315, 8290, 8303, 8343	3362, 3370, 3378, 3386, 3394, 3402,
<code>\int_compare:n</code>	3338, 3338	3410, 3418, 3426, 3604, 3631, 3786,
<code>\int_compare:nF</code>	3442, 3457	8363, 9888, 9891, 9909, 9925, 10286,
<code>\int_compare:nNn</code>	3408, 3408	10394, 10398, 10413, 10417, 10616,
<code>\int_compare:nNnF</code>	2209, 2218,	10617, 10710, 10736, 10747, 10751,
.....	3470, 3485, 8104, 8106, 8117, 8119	10763, 10767, 10780, 10784, 10826,
<code>\int_compare:nNnT</code>	3462, 3479,	10840, 10844, 10857, 10910, 10938,
.....	5400, 8033, 8052, 8070, 8089, 13193	10948, 10969, 10983, 10987, 11000,
<code>\int_compare:nNnTF</code>	11018, 11063, 11072, 11077–11079,
.....	62, 2052, 2104, 2198, 2201, 3278,	11093–11095, 11105, 11109, 11110,
.....	3280, 3491, 3577, 3583, 3730, 3754,	11222, 11229, 11254, 11264, 11384,
.....	3758, 3808, 5413, 5904, 5906, 5911,	11386, 11388, 11400, 11406, 11410,
.....	5919, 5939, 7987, 8000, 8291, 13163	11414, 11498, 11553, 11595, 11639,
<code>\int_compare:nT</code>	3434, 3451	11796, 11901, 11920, 11990, 12069,
<code>\int_compare:nTF</code>	62	12102, 12165, 12171, 12175, 12212,
<code>\int_compare:<:w</code>	3338	12231, 12299, 12329, 12372, 12476,
<code>\int_compare_=:w</code>	3338	12594, 12637, 12641, 12658, 12684,
<code>\int_compare_>:w</code>	3338	12756, 12806, 12903, 12906, 12921
<code>\int_compare_aux:Nw</code>	3338, 3342, 3350	<code>\int_eval_end</code>	69
<code>\int_compare_aux:nw</code>	3338, 3339, 3340	<code>\int_eval_end:</code>	1307, 2160, 2475,
<code>\int_compare_p:nNn</code>	4111, 4112	2477, 2479, 2545, 2547, 2549, 2551,
<code>\int_const:cn</code>	3276, 3767–3780	2553, 2555, 2557, 2559, 2561, 2563,
<code>\int_const:Nn</code>	60, 3276,	2565, 2567, 3194, 3196, 3201, 3204,
.....	3276, 3296, 3829–3847, 9767–9771	3212, 3218, 3223, 3229, 3234, 3259,
<code>\int_convert_from_base_ten:nn</code>	3855, 3855	3266, 3288, 3308, 3310, 3332, 3354,
<code>\int_convert_to_base_ten:nn</code>	3855, 3857	3362, 3370, 3378, 3386, 3394, 3402,
<code>\int_convert_to_symbols:nnn</code>	3855, 3856	3410, 3418, 3426, 3604, 3631, 8366,

- 10286, 10400, 10419, 11002, 11066,
 11072, 11077–11079, 11093–11095,
 11109, 11110, 11222, 11229, 11384,
 11386, 11388, 11408, 11412, 11416,
 11903, 12071, 12104, 12167, 12177
 \int_from_alph:n 66, 3710, 3710
 \int_from_alph_aux:N ... 3710, 3726, 3729
 \int_from_alph_aux:n ... 3710, 3715, 3718
 \int_from_alph_aux:nN
 3710, 3719, 3720, 3725
 \int_from_base:nn
 66, 3731, 3731, 3762, 3764, 3766, 3857
 \int_from_base_aux:N ... 3731, 3748, 3752
 \int_from_base_aux:nn .. 3731, 3736, 3740
 \int_from_base_aux:nnN
 3731, 3741, 3742, 3747
 \int_from_binary:n 66, 3761, 3761
 \int_from_hexadecimal:n . 66, 3761, 3763
 \int_from_octal:n 66, 3761, 3765
 \int_from_roman:n 66, 3781, 3781
 \int_from_roman_aux:NN
 3781, 3787, 3790, 3815, 3819
 \int_from_roman_clean_up:w
 3781, 3798, 3805, 3807, 3826
 \int_from_roman_end:w .. 3781, 3785, 3824
 \int_gadd:cn 3307
 \int_gadd:Nn
 .. 61, 3307, 3311, 3316, 13162, 13192
 \int_gdecr:c 3319
 \int_gdecr:N
 . 61, 2248, 3319, 3325, 3330, 4539,
 4549, 5240, 5793, 5803, 6201, 9100
 \int_get_digits:n 68, 3676, 3681, 3715, 3737
 \int_get_sign:n 68, 3676, 3676, 3714, 3735
 \int_get_sign_and_digits_aux:nnnN ..
 3676, 3678, 3683, 3686, 3709
 \int_get_sign_and_digits_aux:onnN ..
 3676, 3692, 3696, 3702
 \int_gincr:c 3319
 \int_gincr:N
 . 61, 2246, 3319, 3323, 3329, 4533,
 4543, 5236, 5787, 5797, 6196, 9094
 \int_gset:cn 3331
 \int_gset:Nn 61, 3283, 3293, 3331, 3333, 3335
 \int_gset_eq:cc 3301
 \int_gset_eq:cN 3301
 \int_gset_eq:Nc 3301
 \int_gset_eq:NN 60, 3301, 3304–3306
 \int_gsub:cn 3307
 \int_gsub:Nn .. 61, 3307, 3313, 3318, 13200
 \int_gzero:c 3297
 \int_gzero:N 60, 3297, 3298, 3300
 \int_if_even:n 3416, 3424
 \int_if_even:nTF 62
 \int_if_odd:n 3416, 3416
 \int_if_odd:nTF 62
 \int_incr:c 3319
 \int_incr:N 61, 3319, 3319, 3324, 3327,
 8051, 8088, 8309, 9237, 9264, 9331
 \int_max:nn 60, 3206, 3214
 \int_min:nn 60, 3206, 3225
 \int_mod:nn 60, 3236, 3262, 3496, 3587
 \int_new:c 3268
 \int_new:N 60, 2225, 3268, 3269, 3275,
 3282, 3292, 3848–3854, 5784, 6193,
 7958, 8196, 8198–8201, 9001, 9113,
 9772, 9774, 9776, 9778, 9780, 9782,
 9790–9818, 9820–9824, 9827, 9828,
 9830, 9831, 9833–9836, 13154, 13156
 \int_set:cn 3331
 \int_set:Nn 61, 3331, 3331,
 3333, 3334, 7985, 7998, 8014, 8022,
 8036, 8073, 8197, 8243, 8256, 8288,
 8316, 8583, 9233, 9259, 9327, 9773,
 9775, 9777, 9779, 9781, 9783, 10529,
 10564, 13022, 13024, 13026, 13028,
 13030, 13032, 13034, 13036, 13155
 \int_set_eq:cc 3301
 \int_set_eq:cN 3301
 \int_set_eq:Nc 3301
 \int_set_eq:NN
 60, 3301, 3301–3303, 8214, 8250
 \int_show:c 3827, 3828
 \int_show:N 66, 1426, 3827, 3827
 \int_sub:cn 3307
 \int_sub:Nn 61, 3307, 3309, 3314, 3317, 8349
 \int_to_Alph:n 64, 3509, 3541
 \int_to_alph:n 64, 3509, 3509
 \int_to_arabic:n 64, 3488, 3488
 \int_to_base:nn
 65, 3573, 3573, 3635, 3637, 3639, 3855
 \int_to_base_aux_i:nn .. 3573, 3574, 3575
 \int_to_base_aux_ii:nnN
 3573, 3578, 3579, 3581, 3595
 \int_to_base_aux_iii:nnnN 3573, 3586, 3593
 \int_to_binary:n 65, 3634, 3634
 \int_to_hexadecimal:n ... 65, 3634, 3636
 \int_to_letter:n 68, 3573, 3584, 3587, 3601
 \int_to_octal:n 65, 3634, 3638
 \int_to_Roman:n 66, 3640, 3650

- \int_to_roman:n 66, 3640, 3640
- \int_to_roman:w 68, 812, 813,
890, 892, 1059, 1065, 1075, 2007,
2012, 2158, 3194, 3343, 3643, 3653
- \int_to_Roman_aux:N 3652, 3655, 3658
- \int_to_roman_aux:N 3640, 3642, 3645, 3648
- \int_to_Roman_c:w 3640, 3672
- \int_to_roman_c:w 3640, 3664
- \int_to_Roman_d:w 3640, 3673
- \int_to_roman_d:w 3640, 3665
- \int_to_Roman_i:w 3640, 3668
- \int_to_roman_i:w 3640, 3660
- \int_to_Roman_l:w 3640, 3671
- \int_to_roman_l:w 3640, 3663
- \int_to_Roman_m:w 3640, 3674
- \int_to_roman_m:w 3640, 3666
- \int_to_Roman_Q:w 3640, 3675
- \int_to_roman_Q:w 3640, 3667
- \int_to_Roman_v:w 3640, 3669
- \int_to_roman_v:w 3640, 3661
- \int_to_Roman_x:w 3640, 3670
- \int_to_roman_x:w 3640, 3662
- \int_to_symbol:n 3858, 3858
- \int_to_symbol_math:n . . 3858, 3862, 3865
- \int_to_symbol_text:n . . 3858, 3863, 3880
- \int_to_symbols:nnn . . 65, 3489, 3489,
3505, 3511, 3543, 3856, 3867, 3882
- \int_to_symbols_aux:nnnn . . . 3493, 3503
- \int_until_do:nn . . . 64, 3432, 3440, 3445
- \int_until_do:nNnn . . 63, 3460, 3468, 3473
- \int_use:c 3336, 3337
- \int_use:N 62,
2230, 2236, 3336, 3336, 3337, 4534,
4537, 4544, 4547, 5234, 5242, 5788,
5792, 5798, 5802, 6197, 6200, 8028,
8029, 8038, 8043, 8045, 8054, 8065,
8066, 8075, 8080, 8082, 8091, 8484,
9048, 9070, 9096, 9098, 9234, 9260,
9328, 9901, 9941, 9958, 9971, 10005,
10019, 10030, 10044, 10090, 10093,
10095, 10132, 10135, 10137, 10544,
10547, 10549, 10580, 10583, 10585,
10597, 10624, 10653, 10656, 10658,
10679, 10682, 10684, 10733, 10739,
10854, 10860, 10904, 10916, 10997,
11004, 11016, 11246, 11258, 11274,
11286, 11296, 11307, 11320, 11339,
11495, 11501, 11550, 11556, 11592,
11598, 11636, 11642, 11793, 11799,
11917, 11923, 11987, 11993, 12066,
12099, 12125, 12128, 12209, 12215,
12326, 12332, 12369, 12376, 12389,
12411, 12428, 12441, 12451, 12462,
12535, 12537, 12539, 12567, 12578,
12753, 12759, 12786, 12788, 12790,
12792, 12794, 13023, 13025, 13027,
13029, 13031, 13033, 13035, 13037
- \int_value:w 69, 1978, 1979,
1999, 2001–2003, 2160, 3194, 3194,
3201, 3204, 3208, 3216, 3227, 3238,
3264, 3339, 3631, 3786, 3953, 6864,
6888–6890, 6892, 7003, 7012, 7014,
7019–7022, 7026–7029, 7080, 7086,
7088, 7090, 7092, 7097, 7102, 7107,
7114, 7121, 7260, 7290, 7291, 7330,
7349, 7355, 7418, 7420, 7440, 7442,
7467, 7505, 7525, 7533, 7559, 7561,
7565, 7567, 7600, 7614, 7621, 7748,
7848, 7862, 8363, 10286, 10398,
10417, 10736, 10857, 11000, 11498,
11553, 11595, 11639, 11796, 11920,
11990, 12212, 12329, 12372, 12756
- \int_while_do:nn . . . 64, 3432, 3432, 3437
- \int_while_do:nNnn . . 63, 3460, 3460, 3465
- \int_zero:c 3297
- \int_zero:N 60, 3297, 3297, 3299,
8244, 8246, 8337, 9227, 9246, 9321
- \interactionmode 699
- \interlinepenalties 720
- \interlinepenalty 579
- \io_new:c 136
- \ior_alloc_read:n 7986, 8010, 8018
- \ior_close:N
. . . 137, 7984, 8100, 8126, 9684, 9713
- \ior_gto:NN 140, 8393, 8395
- \ior_if_eof:N 8377
- \ior_if_eof:Nf 9703
- \ior_if_eof:Ntf 141, 9681
- \ior_if_eof_p:N 8377
- \ior_list_streams 137
- \ior_list_streams: 8128, 8128, 8408
- \ior_new:c 7974
- \ior_new:N . . . 136, 7974, 7974, 7979, 7980
- \ior_open:cn 7982
- \ior_open:Nn
. . . 136, 7982, 7982, 8008, 9680, 9702
- \ior_open_streams: 8407, 8408
- \ior_raw_new:c 7960, 8080
- \ior_raw_new:N
. . . 141, 7960, 7962, 7970, 7972, 8072

- \ior_show_aux:nn . [8128](#), [8138](#), [8143](#), [8163](#)
 - \ior_str_gto:NN [140](#), [8397](#), [8399](#)
 - \ior_str_to:NN [140](#), [8397](#), [8397](#)
 - \ior_stream_alloc:N [7990](#), [8026](#), [8063](#)
 - \ior_stream_alloc_aux:
 - [8026](#), [8069](#), [8086](#), [8094](#), [8096](#)
 - \ior_to:NN [140](#), [8393](#), [8393](#)
 - \iow_alloc_write:n [7999](#), [8010](#), [8010](#)
 - \iow_char:N
 - [138](#), [5312](#), [5866](#), [6229](#), [6231](#), [8195](#), [8195](#)
 - \iow_close:c [8100](#)
 - \iow_close:N [137](#), [7997](#), [8100](#), [8113](#), [8127](#)
 - \iow_indent:n [139](#), [8233](#), [8233](#), [8259](#)
 - \iow_indent_expandable:n [8233](#), [8234](#), [8259](#)
 - \iow_list_streams [137](#)
 - \iow_list_streams: [8128](#), [8148](#), [8409](#)
 - \iow_log:n [137](#), [8186](#), [8187](#), [9749](#)–[9751](#)
 - \iow_log:x [1163](#), [1163](#), [1202](#),
 - [1808](#), [8186](#), [8186](#), [8571](#), [8573](#), [8574](#)
 - \iow_new:c [7974](#)
 - \iow_new:N [136](#), [7974](#), [7979](#), [7981](#)
 - \iow_newline [138](#)
 - \iow_newline:
 - [5311](#), [5865](#), [6228](#), [7852](#)–[7856](#), [7875](#),
 - [8145](#), [8194](#), [8194](#), [8267](#), [8558](#), [8560](#)
 - \iow_now:Nn [137](#), [8184](#),
 - [8184](#), [8187](#), [8189](#), [8191](#), [8403](#), [8405](#)
 - \iow_now:Nx
 - [8183](#), [8183](#), [8185](#), [8186](#), [8188](#), [8193](#)
 - \iow_now_buffer_safe:Nn [8401](#), [8402](#)
 - \iow_now_buffer_safe:Nx [8401](#), [8404](#)
 - \iow_now_when_avail:Nn [137](#), [8190](#), [8190](#)
 - \iow_now_when_avail:Nx [8190](#), [8192](#)
 - \iow_open:cn [7982](#)
 - \iow_open:Nn [136](#), [7982](#), [7995](#), [8009](#)
 - \iow_open_streams: [8407](#), [8409](#)
 - \iow_raw_new:c [7960](#), [8043](#)
 - \iow_raw_new:N
 - [141](#), [7960](#), [7965](#), [7969](#), [7973](#), [8035](#)
 - \iow_shipout:Nn [137](#), [8180](#), [8180](#), [8182](#)
 - \iow_shipout:Nx [8180](#)
 - \iow_shipout_x:Nn
 - [138](#), [8178](#), [8178](#), [8179](#), [8181](#), [8183](#)
 - \iow_shipout_x:Nx [8178](#)
 - \iow_show_aux:nn [8128](#), [8158](#), [8163](#)
 - \iow_stream_alloc:N [8003](#), [8026](#), [8026](#)
 - \iow_stream_alloc_aux:
 - [8026](#), [8032](#), [8049](#), [8057](#), [8059](#)
 - \iow_term:n [137](#), [8186](#), [8189](#)
 - \iow_term:x [1163](#), [1165](#), [5294](#),
 - [5298](#), [5848](#), [5852](#), [6211](#), [6215](#), [7850](#),
 - [7858](#), [7869](#), [8132](#), [8136](#), [8152](#), [8156](#),
 - [8186](#), [8188](#), [8556](#), [8578](#), [8580](#), [8581](#)
 - \iow_wrap:xnnnN
 - [139](#), [8240](#), [8240](#), [8403](#), [8405](#),
 - [8515](#), [8518](#), [8523](#), [8526](#), [8572](#), [8579](#)
 - \iow_wrap_end: [8353](#)
 - \iow_wrap_end:w [8324](#)
 - \iow_wrap_indent: [8341](#)
 - \iow_wrap_indent:w [8324](#)
 - \iow_wrap_loop:w
 - [8270](#), [8279](#), [8279](#), [8294](#), [8331](#)
 - \iow_wrap_new_marker:n
 - [8213](#), [8217](#), [8228](#)–[8231](#)
 - \iow_wrap_newline: [8333](#)
 - \iow_wrap_newline:w [8324](#)
 - \iow_wrap_special:w [8283](#), [8324](#), [8324](#), [8330](#)
 - \iow_wrap_unindent: [8347](#)
 - \iow_wrap_unindent:w [8324](#)
 - \iow_wrap_word: [8284](#), [8286](#), [8286](#)
 - \iow_wrap_word_fits: [8286](#), [8292](#), [8296](#)
 - \iow_wrap_word_newline: [8286](#), [8293](#), [8312](#)
- J**
- \jobname [654](#)
- K**
- \K [2705](#)
 - \kern [532](#)
 - \kernel_register_show:c [1417](#), [1426](#), [3828](#)
 - \kernel_register_show:N
 - [1417](#), [1417](#), [3827](#), [4048](#), [4129](#), [4176](#)
 - \keys_bool_set:NN [9184](#), [9184](#), [9351](#), [9353](#)
 - \keys_bool_set_inverse:NN
 - [9199](#), [9199](#), [9355](#), [9357](#)
 - \keys_choice_code_store:x
 - [9266](#), [9266](#), [9367](#), [9369](#)
 - \keys_choice_find:n [9217](#), [9307](#), [9556](#), [9556](#)
 - \keys_choice_make: [9187](#),
 - [9202](#), [9214](#), [9214](#), [9226](#), [9245](#), [9359](#)
 - \keys_choices_generate:n [9240](#), [9240](#), [9399](#)
 - \keys_choices_generate_aux:n
 - [9240](#), [9247](#), [9254](#)
 - \keys_choices_make:nn [9224](#), [9224](#), [9361](#)
 - \keys_cmd_set:nn [9192](#), [9207](#), [9216](#), [9218](#),
 - [9277](#), [9277](#), [9298](#), [9310](#), [9312](#), [9363](#)
 - \keys_cmd_set:nx
 - [9188](#), [9190](#), [9203](#), [9205](#), [9230](#), [9256](#),
 - [9277](#), [9282](#), [9303](#), [9324](#), [9343](#), [9365](#)

\keys_cmd_set_aux:n	9277 , 9279 , 9284 , 9287	\keys_set_known:nVN	9454
\keys_default_set:n	\keys_set_known:nvN	9454
..	9197 , 9212 , 9293 , 9293 , 9295 , 9379	\keys_set_known_aux:nnnN	9454 , 9456 , 9467
\keys_default_set:V	\keys_set_known_aux:onnN	9454 , 9455
\keys_define:nn	... 151 , 9122 , 9122 , 9595	\keys_show:nn	... 159 , 9573 , 9573
\keys_define_aux:nnn	... 9122 , 9124 , 9130	\keys_value_or_default:n	9482 , 9505 , 9505
\keys_define_aux:onn	\keys_value_requirement:n
\keys_define_elt:n	9334 , 9334 , 9441 , 9443
\keys_define_elt:nn	... 9127 , 9131 , 9136	\keys_variable_set:cnN	.. 9340 , 9373 , 9385 , 9393 , 9403 , 9419 , 9427 , 9431
\keys_define_elt_aux:nn	\keys_variable_set:cnNN	.. 9340 , 9377 , 9389 , 9397 , 9407 , 9423 , 9435 , 9439
.....	9131 , 9134 , 9139 , 9141	\keys_variable_set:NnN
\keys_define_key:n	9340 , 9346 , 9349 , 9371 , 9383 , 9391 , 9401 , 9417 , 9425 , 9429
\keys_define_key_aux:w	.. 9167 , 9171 , 9182	\keys_variable_set:NnNN
\keys_execute: 9340 , 9340 , 9347 , 9348 , 9375 , 9387 , 9395 , 9405 , 9421 , 9433 , 9437
\keys_execute:nn	\keyval_parse:n
..	9527 , 9528 , 9531 , 9547 , 9558 , 9559	\keyval_parse:NnN
\keys_execute_unknown:	9092 , 9127 , 9449 , 9461 , 9639 – 9641
..	9460 , 9462 , 9527 , 9528 , 9529 , 9537	\keyval_parse_elt:w
\keys_execute_unknown_alt: 9022 , 9028 , 9028 , 9031 , 9036
.....	9460 , 9527 , 9538	\keyval_split_key:w	... 9042 , 9060 , 9060
\keys_execute_unknown_std:	\keyval_split_key_value:w	9035 , 9040 , 9040
.....	9462 , 9527 , 9537	\keyval_split_key_value_aux:wTF
\keys_if_choice_exist:nnn	... 9567 , 9567	9040 , 9053 , 9058
\keys_if_choice_exist:nnTF	\keyval_split_value:w	.. 9054 , 9065 , 9065
\keys_if_exist:nn	\keyval_split_value_aux:w	... 9083 , 9086
\keys_if_exist:nnTF	\KV_process_no_space_removal_no_sanitiz:NNn
\keys_if_value:n	9638 , 9641
\keys_if_value_p:n	\KV_process_space_removal_no_sanitiz:NNn
\keys_meta_make:n	9638 , 9640
\keys_meta_make:x	\KV_process_space_removal_sanitiz:NNn
\keys_multichoice_find:n	9306 , 9306 , 9311	9638 , 9639
\keys_multichoice_make:		
.....	9306 , 9308 , 9320 , 9413		
\keys_multichoices_make:nn		
.....	9306 , 9318 , 9415		
\keys_property_find:n	.. 9142 , 9150 , 9150		
\keys_property_find_aux:w		
.....	9150 , 9154 , 9157 , 9163		
\keys_set:nn		
...	158 , 9299 , 9304 , 9444 , 9444 , 9452		
\keys_set:no		
\keys_set:nV		
\keys_set:nv		
\keys_set_aux:nnn	... 9444 , 9446 , 9453		
\keys_set_aux:onn		
\keys_set_elt:n	.. 9449 , 9461 , 9468 , 9468		
\keys_set_elt:nn	.. 9449 , 9461 , 9468 , 9473		
\keys_set_elt_aux:nn	9468 , 9471 , 9476 , 9478		
\keys_set_known:nnN	159 , 9454 , 9454 , 9466		
\keys_set_known:noN		
	9454		

L

\L	2705
\l@expl@log@functions@bool	1194
\l_box_angle_fp	.. 6510 , 6510 , 6532 , 6580	
\l_box_bottom_dim	.. 6513 , 6514 , 6547 , 6612 , 6616 , 6621 , 6627 , 6632 , 6636 , 6645 , 6647 , 6660 , 6668 , 6691 , 6697 , 6707 , 6712 , 6728 , 6750 , 6769 , 6772	
\l_box_bottom_new_dim	
	6517 , 6518 , 6573 , 6613 , 6624 , 6635 , 6646 , 6690 , 6696 , 6769 , 6773 , 6789	
\l_box_cos_fp	6511 , 6511 , 6536 , 6552 , 6557 , 6584 , 6592 , 6603

\l_box_left_dim 6513, 6515, 6549,
6612, 6614, 6623, 6627, 6632, 6638,
6643, 6647, 6662, 6709, 6730, 6752
\l_box_left_new_dim 6517, 6519,
6564, 6575, 6615, 6626, 6637, 6648
\l_box_right_dim 6513,
6516, 6548, 6610, 6616, 6621, 6625,
6634, 6636, 6645, 6649, 6661, 6664,
6708, 6729, 6732, 6751, 6776, 6777
\l_box_right_new_dim 6517, 6520,
6575, 6617, 6628, 6639, 6650, 6684,
6685, 6776, 6777, 6792, 6794, 6800
\l_box_scale_x_fp 6652,
6652, 6663, 6665, 6670, 6714, 6731,
6733–6735, 6747, 6753, 6775, 6790
\l_box_scale_y_fp 6652, 6653, 6666, 6669,
6672, 6689, 6691, 6695, 6697, 6710,
6713–6715, 6734, 6748, 6755, 6766
\l_box_sin_fp 6511,
6512, 6534, 6550, 6583, 6593, 6604
\l_box_tmp_box 6521, 6521,
6561, 6562, 6568, 6572–6574, 6576,
6782, 6788, 6789, 6795, 6800, 6801
\l_box_tmp_fp 6521, 6522, 6580–6584, 6591, 6593,
6594, 6602, 6604, 6605, 6664, 6665,
6667, 6669, 6711, 6713, 6732, 6733
\l_box_top_dim 6513, 6513, 6546,
6610, 6614, 6623, 6625, 6634, 6638,
6643, 6649, 6659, 6668, 6689, 6695,
6706, 6712, 6727, 6749, 6768, 6773
\l_box_top_new_dim 6517, 6517, 6572, 6611, 6622, 6633,
6644, 6688, 6694, 6768, 6772, 6788
\l_box_x_fp 6523, 6523, 6588, 6590, 6599, 6602
\l_box_x_new_fp 6523, 6525, 6590, 6592, 6594, 6595
\l_box_y_fp 6523, 6524, 6589, 6591, 6600, 6601
\l_box_y_new_fp 6523, 6526, 6601, 6603, 6605, 6606
\l_cctab_tmp_tl 13188, 13201–13204, 13218
\l_clist_if_in_clist 5737, 5737, 5750, 5751
\l_clist_remove_clist 5689, 5689, 5696, 5699, 5700, 5702
\l_clist_show_tl 5857, 5860
\l_clist_tmpa_tl 5539, 5539,
5628, 5630, 5632, 5633, 5825, 5826
\l_coffin_aligned_coffin 6996, 6998, 7247,
7248, 7252, 7258, 7260, 7261, 7277,
7278, 7284–7288, 7290, 7292, 7296,
7297, 7302–7306, 7340, 7355, 7400,
7402, 7831, 7838, 7840, 7842, 7844
\l_coffin_aligned_internal_coffin 6996, 6999, 7319, 7326
\l_coffin_bottom_corner_dim 7407, 7409,
7432, 7436, 7503, 7512, 7528, 7536
\l_coffin_bounding_prop 7405, 7405, 7423,
7448, 7450, 7453, 7455, 7461, 7518
\l_coffin_bounding_shift_dim 7406, 7406, 7430, 7517, 7522
\l_coffin_calc_a_fp 6832, 6832, 7191, 7195, 7202, 7204–
7206, 7209–7211, 7214, 7234, 7238
\l_coffin_calc_b_fp 6832, 6833, 7192, 7195,
7198, 7207, 7215, 7218, 7235, 7241
\l_coffin_calc_c_fp 6832, 6834, 7193, 7196, 7236, 7240
\l_coffin_calc_d_fp . 6832, 6835, 7194,
7196, 7198, 7212, 7216, 7237, 7239
\l_coffin_calc_result_fp 6832, 6836, 7201, 7203, 7208,
7213, 7217, 7220, 7233, 7238–7242
\l_coffin_cos_fp 6842, 6843, 7417, 7488, 7493
\l_coffin_Depth_dim 6852, 6852, 7038
\l_coffin_display_coffin 7627, 7627, 7756, 7763,
7833, 7834, 7839, 7841, 7843, 7844
\l_coffin_display_coord_coffin 7627, 7628,
7694, 7714, 7730, 7778, 7798, 7817
\l_coffin_display_font_tl 7672, 7672, 7674, 7677, 7702, 7786
\l_coffin_display_handles_prop 7630, 7630, 7631, 7633, 7635, 7637,
7639, 7641, 7643, 7645, 7647, 7649,
7651, 7653, 7655, 7657, 7659, 7661,
7663, 7665, 7705, 7709, 7789, 7793
\l_coffin_display_offset_dim . 7667,
7667, 7668, 7731, 7732, 7818, 7819
\l_coffin_display_pole_coffin 7627, 7629, 7682, 7693, 7737, 7776

\l_coffin_display_poles_prop	\l_coffin_top_corner_dim
..... 7671, 7671, 7407, 7410, 7436, 7501, 7513
7747, 7752, 7755, 7758, 7760, 7767	\l_coffin_TotalHeight_dim 6852, 6854, 7039
\l_coffin_display_x_dim	\l_coffin_Width_dim 6852, 6855, 7040
..... 7669, 7669, 7773, 7828	\l_coffin_x_dim 6844, 6844, 7140,
\l_coffin_display_y_dim	7149, 7169, 7172, 7179, 7186, 7188,
..... 7669, 7670, 7774, 7830	7219, 7222, 7312, 7316, 7335, 7343,
\l_coffin_error_bool 6837, 6837, 7133,	7460, 7462, 7466, 7468, 7472, 7477,
7137, 7151, 7166, 7199, 7769, 7771	7599, 7601, 7605, 7608, 7773, 7825
\l_coffin_handles_tmp_prop	\l_coffin_x_fp 6848, 6848, 7484, 7486, 7492
..... 7679, 7679, 7757	\l_coffin_x_prime_dim ... 6844, 6846,
\l_coffin_Height_dim ... 6852, 6853, 7037	7312, 7316, 7474, 7478, 7825, 7828
\l_coffin_left_corner_dim 7407, 7407,	\l_coffin_x_prime_fp
7431, 7439, 7504, 7510, 7527, 7535	.. 6848, 6850, 7486, 7488, 7490, 7496
\l_coffin_offset_x_dim	\l_coffin_y_dim
..... 6838, 6838, 7250,	6844, 6845, 7141, 7154, 7157, 7164,
7251, 7254, 7262, 7264, 7266, 7272,	7181, 7223, 7313, 7318, 7336, 7343,
7275, 7295, 7315, 7323, 7827, 7835	7460, 7462, 7466, 7468, 7472, 7477,
\l_coffin_offset_y_dim	7599, 7601, 7605, 7608, 7774, 7826
... 6838, 6839, 7265, 7267, 7272,	\l_coffin_y_fp 6848, 6849, 7485, 7487, 7491
7275, 7295, 7317, 7324, 7829, 7836	\l_coffin_y_prime_dim ... 6844, 6847,
\l_coffin_pole_a_tl	7313, 7318, 7474, 7479, 7826, 7830
... 6840, 6840, 7131, 7136, 7364,	\l_coffin_y_prime_fp
7367, 7368, 7371, 7749, 7751, 7754	.. 6848, 6851, 7491, 7493, 7495, 7497
\l_coffin_pole_b_tl	\l_doc_pTF_name_tl
6840, 6841, 7132, 7136, 7365, 7367,	21,
7369, 7371, 7750, 7751, 7753, 7754	22, 36, 37, 40, 41, 44, 51, 52, 53,
\l_coffin_right_corner_dim	54, 62, 72, 76, 85, 86, 87, 91, 92,
..... 7407, 7408, 7439, 7502, 7511	98, 108, 109, 117, 125, 141, 159, 166
\l_coffin_scale_x_fp 7540, 7540,	\l_exp_tl 30, 1519, 1519, 1538, 1539
7548, 7550, 7563, 7576, 7581, 7591	\l_expl_status_bool
\l_coffin_scale_y_fp 96, 294, 309, 323, 327, 328
7541, 7551, 7553, 7577, 7578, 7594	\l_expl_status_stack_tl
\l_coffin_scaled_total_height_dim ..	197
..... 7542, 7542, 7579, 7580, 7585	\l_file_name_tl
\l_coffin_scaled_width_dim	162, 9670,
..... 7542, 7543, 7582, 7583, 7585	9670, 9717, 9718, 9724, 9725, 9735
\l_coffin_sin_fp	\l_file_search_path_saved_seq
..... 6842, 6842, 7416, 7489, 7494 162, 9672, 9673, 9694, 9711
\l_coffin_tmp_box	\l_file_search_path_seq
6811, 6811, 162, 9671, 9671, 9694, 9696,
6928, 6932, 6936, 6971, 6976, 6981	9697, 9700, 9711, 9741, 9742, 9745
\l_coffin_tmp_dim	\l_file_tmpa_seq
6811, 6812,	162, 9675, 9676, 9695, 9697, 9756, 9757
7253, 7255, 7256, 7452, 7454, 7456	\l_fp_arg_tl
\l_coffin_tmp_fp 6811, 6813, 7413-7417,	9789, 9789, 11488,
7487, 7489, 7490, 7492, 7494, 7495,	11507, 11511, 11521, 11542, 11543,
7549, 7550, 7552, 7553, 7590-7595	11585, 11600, 11608, 11628, 11629,
\l_coffin_tmp_tl .. 6811, 6814, 6821-	11786, 11805, 11809, 11819, 11829,
6831, 7338, 7339, 7341, 7706, 7707,	11853, 11884, 11909, 11910, 11980,
7710, 7711, 7719, 7724, 7790, 7791,	11995, 12004, 12006, 12034, 12074,
7794, 7795, 7804, 7809, 7859, 7866	12206, 12323, 12334, 12342, 12362
	\l_fp_count_int
	9790, 9790,
	11010, 11045, 11057, 11065, 11683,

11715, 11732, 11734, 11737, 11739,
 12183, 12220, 12228, 12458, 12461,
 12462, 12484, 12528, 12588, 12599
 \l_fp_div_offset_int 9791, 9791,
 10966, 11020, 11065, 11067, 11898
 \l_fp_exp_decimal_int . . . 9792, 9793,
 12058, 12092, 12111, 12131, 12150,
 12159, 12164, 12170, 12186, 12235,
 12240, 12244, 12247, 12251, 12255,
 12258, 12260, 12404, 12414, 12425,
 12428, 12437, 12445, 12471, 12534,
 12542, 12546, 12557, 12600, 12601
 \l_fp_exp_exponent_int
 9792, 9795, 12060,
 12094, 12116, 12136, 12152, 12402,
 12406, 12424, 12431, 12454, 12538
 \l_fp_exp_extended_int 9792,
 9794, 12059, 12093, 12111, 12131,
 12151, 12160, 12163, 12168, 12174,
 12186, 12236, 12238, 12241, 12252,
 12254, 12256, 12405, 12414, 12447,
 12451, 12453, 12471, 12536, 12543,
 12545, 12548, 12557, 12600, 12602
 \l_fp_exp_integer_int
 9792, 9792, 12057, 12091,
 12111, 12131, 12149, 12158, 12162,
 12186, 12246, 12259, 12403, 12414,
 12436, 12441, 12444, 12471, 12533
 \l_fp_input_a_decimal_int
 . . 9796, 9798, 9847, 10038, 10039,
 10044, 10046, 10077, 10079, 10093,
 10119, 10121, 10135, 10533, 10547,
 10569, 10583, 10595, 10597, 10604,
 10608, 10646, 10656, 10671, 10682,
 10752, 10768, 10867, 10949, 11014,
 11016, 11018, 11034, 11047, 11048,
 11050, 11073, 11244, 11246, 11255,
 11263, 11264, 11271, 11274, 11282,
 11290, 11371, 11385, 11386, 11390,
 11393, 11395, 11401, 11409, 11411,
 11424, 11425, 11432, 11434, 11442,
 11452, 11455, 11461, 11463, 11467,
 11486, 11499, 11583, 11596, 11670,
 11676, 11677, 11707, 11710, 11713,
 11724, 11728, 11784, 11797, 11851,
 11875, 11880, 11882, 11977, 11991,
 12156, 12159, 12166, 12172, 12181,
 12223, 12295, 12300, 12330, 12477,
 12491, 12495, 12498, 12513, 12518,
 12522, 12525, 12526, 12590, 12592,
 12595, 12598, 12628, 12634, 12642,
 12659, 12685, 12718, 12768, 12779,
 12799, 12812, 12845, 12861, 12879,
 12904, 12974, 13006, 13026, 13035
 \l_fp_input_a_exponent_int
 9796, 9799, 9848,
 9993, 10001, 10026, 10047, 10078,
 10095, 10120, 10137, 10549, 10565,
 10585, 10609, 10658, 10684, 10717,
 10827, 10970, 11242, 11266, 11270,
 11299, 11346, 11487, 11501, 11503,
 11584, 11598, 11785, 11799, 11801,
 11826, 11876, 11881, 11902, 11978,
 11993, 12016, 12296, 12332, 12381,
 12382, 12387, 12389, 12400, 12410,
 12411, 12511, 12520, 12629, 12635,
 12680, 12714, 12769, 12780, 12807,
 12814, 12846, 12862, 12881, 12956,
 12960, 12988, 12992, 13028, 13037
 \l_fp_input_a_extended_int
 9804, 9804, 11256,
 11258, 11265, 11292, 11296, 11298,
 11347, 11387–11389, 11391, 11401,
 11413, 11415, 11426, 11433, 11435,
 11443, 11453, 11456, 11464, 11468,
 11676, 11677, 11711, 11714, 11724,
 11728, 11760, 11979, 12160, 12176,
 12182, 12223, 12253, 12459, 12492,
 12499, 12519, 12523, 12525, 12526,
 12590, 12592, 12595, 12598, 12679,
 12685, 12716, 12797, 12800, 12813
 \l_fp_input_a_integer_int
 . . 9796, 9797, 9846, 10027, 10030,
 10037, 10076, 10090, 10118, 10132,
 10544, 10580, 10603, 10605, 10607,
 10653, 10679, 10748, 10764, 10877,
 10882, 10886, 10891, 10949, 11013,
 11018, 11028, 11031, 11046, 11049,
 11071, 11072, 11243, 11253, 11254,
 11281, 11286, 11289, 11350, 11352,
 11365, 11370, 11383, 11384, 11394,
 11397, 11403, 11405, 11407, 11432,
 11434, 11450, 11452, 11455, 11463,
 11467, 11485, 11495, 11582, 11592,
 11783, 11793, 11852, 11874, 11879,
 11883, 11976, 11987, 12019, 12029,
 12044, 12056, 12066, 12068, 12070,
 12095, 12099, 12101, 12103, 12123,
 12125, 12128, 12294, 12300, 12326,
 12477, 12483, 12494, 12497, 12510,

- 12517, 12627, 12633, 12642, 12659,
12719, 12767, 12778, 12799, 12811,
12844, 12860, 12878, 12904, 12964,
12969, 12996, 13001, 13024, 13033
- `\l_fp_input_a_sign_int`
..... 9796, 9796, 9842, 9844,
10075, 10085, 10117, 10127, 10539,
10575, 10674, 10711, 10745, 10789,
10841, 10984, 11351, 11356, 11398,
11484, 11490, 11538, 11545, 11581,
11587, 11624, 11631, 11657, 11659,
11664, 11782, 11788, 11837, 11878,
11975, 11982, 12018, 12072, 12105,
12124, 12157, 12180, 12221, 12293,
12297, 12626, 12632, 12711, 12765,
12810, 12843, 12859, 12877, 12924,
12938, 12942, 12946, 13022, 13031
- `\l_fp_input_b_decimal_int`
..... 9796, 9802, 10013, 10014,
10019, 10021, 10701, 10752, 10768,
10804, 10822, 10869, 10935, 10939,
11034, 11047, 11866, 11871, 12181,
12224, 12225, 12227, 12229, 12232,
12235, 12251, 12513, 12527, 12591,
12628, 12638, 12771, 12779, 12789,
12801, 12851, 12861, 12879, 12907,
12922, 12974, 13006, 13027, 13034
- `\l_fp_input_b_exponent_int`
.. 9796, 9803, 9993, 10001, 10022,
10026, 10702, 10805, 10823, 10827,
10936, 10970, 11867, 11872, 11902,
12514, 12629, 12772, 12780, 12793,
12807, 12852, 12862, 12881, 12956,
12960, 12988, 12992, 13029, 13036
- `\l_fp_input_b_extended_int`
..... 9804, 9805, 12182,
12224, 12225, 12227, 12229, 12232,
12237, 12527, 12591, 12791, 12802
- `\l_fp_input_b_integer_int`
..... 9796, 9801, 10002, 10005,
10012, 10700, 10748, 10764, 10803,
10821, 10880, 10884, 10887, 10891,
10934, 10939, 11028, 11031, 11046,
11865, 11870, 12512, 12627, 12638,
12770, 12778, 12787, 12801, 12850,
12860, 12878, 12907, 12922, 12964,
12969, 12996, 13001, 13025, 13032
- `\l_fp_input_b_sign_int` .. 9796, 9800,
10699, 10711, 10775, 10802, 10806,
10820, 10841, 10933, 10984, 11869,
12180, 12221, 12234, 12626, 12675,
12785, 12810, 12849, 12859, 12877,
12912, 12938, 12942, 13023, 13030
- `\l_fp_mul_a_i_int` 9806, 9806,
10868, 10873, 10878, 10883, 10887,
11117, 11126, 11133, 11139, 11144,
11150, 11153, 11161, 11170, 11178,
11186, 11193, 11201, 11206, 11210
- `\l_fp_mul_a_ii_int`
..... 9806, 9807, 10868, 10874,
10879, 10884, 11117, 11127, 11134,
11140, 11145, 11151, 11161, 11171,
11179, 11187, 11194, 11202, 11207
- `\l_fp_mul_a_iii_int` 9806,
9808, 10868, 10875, 10880, 11117,
11128, 11135, 11141, 11146, 11161,
11172, 11180, 11188, 11195, 11203
- `\l_fp_mul_a_iv_int` 9806,
9809, 11119, 11129, 11136, 11142,
11163, 11173, 11181, 11189, 11196
- `\l_fp_mul_a_v_int`
..... 9806, 9810, 11119, 11130,
11137, 11163, 11174, 11182, 11190
- `\l_fp_mul_a_vi_int` 9806, 9811,
11119, 11131, 11163, 11175, 11183
- `\l_fp_mul_b_i_int` 9806, 9812,
10870, 10875, 10879, 10883, 10886,
11121, 11131, 11137, 11142, 11146,
11151, 11153, 11165, 11175, 11182,
11189, 11195, 11202, 11206, 11209
- `\l_fp_mul_b_ii_int`
..... 9806, 9813, 10870, 10874,
10878, 10882, 11121, 11130, 11136,
11141, 11145, 11150, 11165, 11174,
11181, 11188, 11194, 11201, 11205
- `\l_fp_mul_b_iii_int` 9806,
9814, 10870, 10873, 10877, 11121,
11129, 11135, 11140, 11144, 11165,
11173, 11180, 11187, 11193, 11200
- `\l_fp_mul_b_iv_int` 9806,
9815, 11123, 11128, 11134, 11139,
11167, 11172, 11179, 11186, 11192
- `\l_fp_mul_b_v_int`
..... 9806, 9816, 11123, 11127,
11133, 11167, 11171, 11178, 11185
- `\l_fp_mul_b_vi_int` 9806, 9817,
11123, 11126, 11167, 11170, 11177
- `\l_fp_mul_output_int` 9818, 9818, 10871,
10876, 10909, 10910, 10914, 10916,
10921, 11124, 11132, 11168, 11176

`\l_fp_mul_output_tl` [9818](#), [9819](#),
[10872](#), [10889](#), [10890](#), [10893](#), [10920](#),
[11125](#), [11148](#), [11149](#), [11156](#), [11169](#),
[11198](#), [11199](#), [11212](#), [11213](#), [11216](#)
`\l_fp_output_decimal_int`
. [9820](#), [9822](#), [10721](#), [10737](#),
[10750](#), [10754](#), [10757](#), [10766](#), [10770](#),
[10772](#), [10776](#), [10779](#), [10781](#), [10832](#),
[10845](#), [10858](#), [10889](#), [10964](#), [10975](#),
[10988](#), [11001](#), [11062](#), [11064](#), [11315](#),
[11320](#), [11328](#), [11358](#), [11533](#), [11534](#),
[11540](#), [11554](#), [11619](#), [11620](#), [11626](#),
[11640](#), [11672](#), [11687](#), [11691](#), [11696](#),
[11700](#), [11708](#), [11710](#), [11742](#), [11747](#),
[11751](#), [11754](#), [11758](#), [11762](#), [11765](#),
[11767](#), [11865](#), [11874](#), [11896](#), [11907](#),
[11921](#), [12048](#), [12092](#), [12112](#), [12114](#),
[12132](#), [12134](#), [12187](#), [12189](#), [12194](#),
[12195](#), [12197](#), [12204](#), [12213](#), [12350](#),
[12351](#), [12353](#), [12360](#), [12373](#), [12392](#),
[12404](#), [12415](#), [12417](#), [12469](#), [12472](#),
[12518](#), [12521](#), [12522](#), [12535](#), [12555](#),
[12558](#), [12564](#), [12567](#), [12574](#), [12582](#),
[12601](#), [12605](#), [12609](#), [12612](#), [12757](#),
[12771](#), [12790](#), [12803](#), [12812](#), [12816](#)
`\l_fp_output_exponent_int` [9820](#),
[9823](#), [10717](#), [10722](#), [10739](#), [10825](#),
[10833](#), [10860](#), [10968](#), [10976](#), [11004](#),
[11342](#), [11359](#), [11531](#), [11535](#), [11541](#),
[11556](#), [11617](#), [11621](#), [11627](#), [11642](#),
[11900](#), [11908](#), [11923](#), [12050](#), [12094](#),
[12116](#), [12136](#), [12205](#), [12215](#), [12361](#),
[12376](#), [12394](#), [12406](#), [12424](#), [12431](#),
[12520](#), [12539](#), [12563](#), [12584](#), [12759](#),
[12772](#), [12794](#), [12805](#), [12814](#), [12818](#)
`\l_fp_output_extended_int`
. [9824](#), [9824](#), [11333](#),
[11334](#), [11339](#), [11341](#), [11534](#), [11620](#),
[11688](#), [11692](#), [11697](#), [11699](#), [11711](#),
[11743](#), [11745](#), [11748](#), [11759](#), [11761](#),
[11763](#), [11866](#), [11875](#), [12049](#), [12093](#),
[12113](#), [12115](#), [12133](#), [12135](#), [12188](#),
[12190](#), [12192](#), [12348](#), [12393](#), [12405](#),
[12416](#), [12418](#), [12470](#), [12473](#), [12523](#),
[12537](#), [12556](#), [12559](#), [12602](#), [12603](#),
[12606](#), [12792](#), [12804](#), [12813](#), [12817](#)
`\l_fp_output_integer_int`
. [9820](#), [9821](#), [10720](#),
[10733](#), [10746](#), [10756](#), [10762](#), [10771](#),
[10774](#), [10777](#), [10783](#), [10785](#), [10831](#),
[10845](#), [10854](#), [10893](#), [10963](#), [10974](#),
[10988](#), [10997](#), [11057](#), [11303](#), [11304](#),
[11307](#), [11314](#), [11357](#), [11530](#), [11533](#),
[11539](#), [11550](#), [11616](#), [11619](#), [11625](#),
[11636](#), [11671](#), [11686](#), [11690](#), [11695](#),
[11706](#), [11753](#), [11766](#), [11895](#), [11906](#),
[11917](#), [12047](#), [12073](#), [12091](#), [12112](#),
[12114](#), [12132](#), [12134](#), [12187](#), [12189](#),
[12198](#), [12203](#), [12209](#), [12354](#), [12359](#),
[12369](#), [12391](#), [12403](#), [12415](#), [12417](#),
[12469](#), [12472](#), [12517](#), [12555](#), [12558](#),
[12573](#), [12578](#), [12581](#), [12611](#), [12753](#),
[12770](#), [12788](#), [12803](#), [12811](#), [12815](#)
`\l_fp_output_sign_int`
. [9820](#), [9820](#), [10719](#),
[10728](#), [10745](#), [10775](#), [10789](#), [10830](#),
[10973](#), [11839](#), [11841](#), [11845](#), [11847](#),
[11905](#), [11912](#), [12202](#), [12358](#), [12364](#),
[12383](#), [12385](#), [12464](#), [12550](#), [12786](#)
`\l_fp_round_carry_bool` . . [9825](#), [9825](#),
[10592](#), [10602](#), [10615](#), [10621](#), [10629](#)
`\l_fp_round_decimal_tl` . . [9826](#), [9826](#),
[10594](#), [10604](#), [10623](#), [10624](#), [10626](#)
`\l_fp_round_position_int` . [9827](#), [9827](#),
[10593](#), [10614](#), [10627](#), [10633](#), [10634](#)
`\l_fp_round_target_int`
. [9827](#), [9828](#), [10529](#),
[10530](#), [10564](#), [10566](#), [10614](#), [10627](#)
`\l_fp_sign_tl`
. [9829](#), [9829](#), [12676](#), [12688](#), [12752](#)
`\l_fp_split_sign_int`
. [9830](#), [9830](#), [9855](#), [9857](#), [9870](#)
`\l_fp_tmp_dim` . [10102](#), [10110](#), [10114](#), [10150](#)
`\l_fp_tmp_int` [9831](#),
[9831](#), [9899](#), [9901](#), [10616](#)–[10619](#),
[10624](#), [11220](#)–[11222](#), [11227](#)–[11229](#)
`\l_fp_tmp_skip` [10102](#), [10109](#), [10110](#), [10151](#)
`\l_fp_tmp_tl` . . [9832](#), [9832](#), [9852](#)–[9854](#),
[9858](#), [9863](#), [9865](#), [9868](#), [9874](#), [9876](#),
[9879](#), [9968](#), [9973](#), [10015](#), [10021](#),
[10040](#), [10046](#), [10672](#), [10687](#), [11284](#),
[11290](#), [11293](#), [11298](#), [11316](#), [11323](#),
[11335](#), [11341](#), [12063](#), [12070](#), [12077](#),
[12083](#), [12096](#), [12103](#), [12106](#), [12108](#),
[12439](#), [12445](#), [12448](#), [12453](#), [12576](#),
[12582](#), [13020](#), [13039](#), [13045](#), [13050](#)
`\l_fp_trig_decimal_int`
. [9834](#), [9835](#), [11678](#), [11680](#),
[11682](#), [11685](#), [11700](#), [11713](#), [11723](#),

- 11725, 11727, 11729, 11731, 11733,
11736, 11738, 11740, 11742, 11758
- \l_fp_trig_extended_int . 9834, 9836,
11678, 11680, 11682, 11684, 11699,
11714, 11723, 11725, 11727, 11729,
11731, 11733, 11736, 11738, 11744
- \l_fp_trig_octant_int
..... 9833, 9833, 11422, 11428,
11440, 11441, 11457, 11469, 11561,
11647, 11658, 11662, 11838, 11844
- \l_fp_trig_sign_int
9834, 9834, 11674, 11712, 11721, 11741
- \l_ior_stream_int
..... 7958, 7959, 7985, 7987,
7991, 8022, 8065, 8066, 8070, 8073,
8080, 8082, 8088, 8089, 8091, 8093
- \l_iow_current_indentation_int
..... 8199, 8201,
8244, 8304, 8319, 8343, 8349, 8351
- \l_iow_current_indentation_tl 8202,
8204, 8245, 8302, 8322, 8344, 8350
- \l_iow_current_line_int
..... 8199, 8199, 8246,
8290, 8291, 8303, 8309, 8316, 8337
- \l_iow_current_line_tl
..... 8202, 8202, 8247, 8301,
8307, 8315, 8321, 8336, 8338, 8356
- \l_iow_current_word_int
..... 8199, 8200, 8288, 8290, 8318
- \l_iow_current_word_tl .. 8202, 8203,
8281, 8282, 8289, 8302, 8308, 8322
- \l_iow_line_length_int
..... 139, 8196, 8196, 8197, 8243
- \l_iow_line_start_bool
.. 8207, 8207, 8249, 8298, 8300, 8339
- \l_iow_stream_int 7958,
7958, 7959, 7998, 8000, 8004, 8014,
8028, 8029, 8033, 8036, 8038, 8043,
8045, 8051, 8052, 8054, 8056, 8075
- \l_iow_target_length_int
..... 8198, 8198, 8243, 8291
- \l_iow_wrap_tl
.. 8205, 8205, 8248, 8262, 8265, 8271
- \l_iow_wrapped_tl
.. 8206, 8206, 8277, 8314, 8335, 8355
- \l_keys_choice_int .. 157, 9113, 9113,
9227, 9233, 9234, 9237, 9246, 9259,
9260, 9264, 9321, 9327, 9328, 9331
- \l_keys_choice_tl . 157, 9232, 9258, 9326
- \l_keys_choices_tl 9113, 9114
- \l_keys_key_tl 158, 9115,
9115, 9195, 9210, 9480, 9481, 9542
- \l_keys_module_tl
... 9116, 9116, 9123, 9126, 9128,
9152, 9299, 9304, 9445, 9448, 9450,
9455, 9458, 9463, 9481, 9531, 9534
- \l_keys_no_value_bool
... 9117, 9117, 9133, 9138, 9169,
9470, 9475, 9486, 9496, 9508, 9543
- \l_keys_path_tl
.... 158, 9118, 9118, 9147, 9152,
9159, 9162, 9177, 9188, 9190, 9192,
9203, 9205, 9207, 9216, 9218, 9221,
9230, 9243, 9251, 9256, 9262, 9269,
9272, 9274, 9294, 9298, 9303, 9310,
9312, 9315, 9324, 9337, 9343, 9363,
9365, 9481, 9490, 9500, 9510, 9512,
9515, 9523, 9528, 9534, 9558, 9559
- \l_keys_property_tl . 9119, 9119, 9143,
9147, 9165, 9172, 9173, 9176, 9180
- \l_keys_unknown_clist
..... 9120, 9120, 9459, 9464, 9540
- \l_keys_value_tl 158, 9121,
9121, 9500, 9507, 9514, 9544, 9552
- \l_keyval_key_tl
..... 9002, 9002, 9049, 9062, 9071
- \l_keyval_parse_tl .. 9004, 9005, 9021,
9025, 9045, 9067, 9076, 9080, 9089
- \l_keyval_sanitise_tl
..... 9004, 9004, 9017-9020, 9023
- \l_keyval_value_tl 9002,
9003, 9073, 9075, 9078, 9088, 9090
- \l_last_box 125, 6411, 6411, 6413
- \l_msg_class_tl 8684, 8685, 8735, 8736, 8739
- \l_msg_current_class_tl
..... 8684, 8686, 8717, 8736
- \l_msg_current_module_tl 8684, 8687, 8718
- \l_msg_redirect_classes_prop 8591, 8591
- \l_msg_redirect_classes_seq
..... 8684, 8684, 8692, 8697, 8700
- \l_msg_redirect_kernel_info_prop . 8836
- \l_msg_redirect_kernel_warning_prop
..... 8814
- \l_msg_redirect_names_prop
..... 8591, 8592, 8719, 8750
- \l_msg_text_tl 8497, 8497, 8530, 8563, 8565
- \l_msg_tmp_tl 8418, 8418, 8534, 8537, 8545
- \l_peek_search_tl
..... 2856, 2856, 2874, 2895, 2938

<code>\mathchardef</code>	359	<code>\msg_error:nnxxx</code>	8640
<code>\mathchoice</code>	459	<code>\msg_error:nnxxxx</code>	145, 8640
<code>\mathclose</code>	492	<code>\msg_error_text:n</code>	
<code>\mathcode</code>	670		143, 8584, 8586, 8645, 8654, 8784, 8797
<code>\mathinner</code>	493	<code>\msg_expandable_error:n</code>	
<code>\mathop</code>	494		149, 1570, 2199, 4573, 5004,
<code>\mathopen</code>	498		5474, 8328, 8951, 8959, 13139, 13144
<code>\mathord</code>	499	<code>\msg_expandable_error_aux:w</code> .	8965, 8972
<code>\mathparagraph</code>	3873	<code>\msg_fatal:nn</code>	8618
<code>\mathpunct</code>	500	<code>\msg_fatal:nnx</code>	8618
<code>\mathrel</code>	501	<code>\msg_fatal:nnxx</code>	8618
<code>\mathsection</code>	3872	<code>\msg_fatal:nnxxx</code>	8618
<code>\mathsurround</code>	512	<code>\msg_fatal:nnxxxx</code>	144, 8618
<code>\maxdeadcycles</code>	582	<code>\msg_fatal_text:n</code>	
<code>\maxdepth</code>	583		143, 8584, 8584, 8621, 8762
<code>\maxdimen</code>	4057	<code>\msg_generic_new:nn</code>	8985, 8987
<code>\meaning</code>	642	<code>\msg_generic_new:nnn</code>	8985, 8986
<code>\medmuskip</code>	513	<code>\msg_generic_set:nn</code>	8985, 8989
<code>\message</code>	419	<code>\msg_generic_set:nnn</code>	8985, 8988
<code>\MessageBreak</code>	222, 238–244	<code>\msg_gset:nnn</code>	8421, 8448
<code>\middle</code>	724	<code>\msg_gset:nnnn</code> ...	8421, 8428, 8441, 8449
<code>\mkern</code>	466	<code>\msg_if_more_text:c</code>	8607
<code>\mode_if_horizontal:</code>	2252, 2252	<code>\msg_if_more_text:cTF</code>	8642, 8781
<code>\mode_if_horizontalTF</code>	40	<code>\msg_if_more_text:N</code>	8607, 8607
<code>\mode_if_inner:</code>	2254, 2254	<code>\msg_if_more_text:NF</code>	8616
<code>\mode_if_innerTF</code>	40	<code>\msg_if_more_text:NT</code>	8615
<code>\mode_if_math:</code>	2256, 2256	<code>\msg_if_more_text:NTF</code>	8617
<code>\mode_if_math:TF</code>	3861	<code>\msg_if_more_text_p:N</code>	8614
<code>\mode_if_mathTF</code>	40	<code>\msg_info:nn</code>	8670
<code>\mode_if_vertical:</code>	2250, 2250	<code>\msg_info:nnx</code>	8670
<code>\mode_if_verticalTF</code>	41	<code>\msg_info:nnxx</code>	8670
<code>\month</code>	652	<code>\msg_info:nnxxx</code>	8670
<code>\moveleft</code>	602	<code>\msg_info:nnxxxx</code>	145, 8670
<code>\moveright</code>	603	<code>\msg_info_text:n</code> 144, 8584, 8588, 8674, 8843	
<code>\msg_class_new:nn</code>	8975, 8976	<code>\msg_interrupt:xxx</code>	
<code>\msg_class_set:nn</code>			147, 8498, 8498, 8620, 8631,
	144, 8593, 8593, 8618, 8629,		8644, 8653, 8761, 8783, 8796, 8936
	8640, 8662, 8670, 8678, 8683, 8976	<code>\msg_interrupt_aux:</code> ...	8498, 8504, 8554
<code>\msg_critical:nn</code>	8629	<code>\msg_interrupt_details:xxx</code>	
<code>\msg_critical:nnx</code>	8629		8498, 8503, 8521
<code>\msg_critical:nnxx</code>	8629	<code>\msg_interrupt_more_text:n</code>	
<code>\msg_critical:nnxxx</code>	8629		8498, 8517, 8525, 8531
<code>\msg_critical:nnxxxx</code>	144, 8629	<code>\msg_interrupt_no_details:xx</code>	
<code>\msg_critical_text:n</code> 143, 8584, 8585, 8632			8498, 8502, 8513
<code>\msg_direct_interrupt:xxxxx</code> .	8985, 8990	<code>\msg_interrupt_text:n</code>	
<code>\msg_direct_log:xx</code>	8985, 8991		8498, 8519, 8527, 8529
<code>\msg_direct_term:xx</code>	8985, 8992	<code>\msg_kernel_bug:x</code>	8934, 8934
<code>\msg_error:nn</code>	8640	<code>\msg_kernel_error:nn</code>	
<code>\msg_error:nnx</code>	8640		1167, 1181, 7139, 7988,
<code>\msg_error:nnxx</code>	8640		

- 8001, 8425, 8698, [8779](#), 8812, 9055,
13101, 13109, 13114, 13123, 13194
- \msg_kernel_error:nnx
..... [1167](#), [1179](#), [1422](#), [4387](#),
[5205](#), [6867](#), [6872](#), [8707](#), [8779](#), 8810,
[9155](#), [9194](#), [9209](#), [9250](#), [9489](#), [13172](#)
- \msg_kernel_error:nnxx
[1167](#), [1167](#), [1180](#), [1182](#), [1189](#), [1199](#),
[1212](#), [1332](#), [7005](#), [8713](#), [8779](#), 8808,
[9146](#), [9175](#), [9220](#), [9314](#), [9499](#), [9533](#)
- \msg_kernel_error:nnxxx [8779](#), 8806
- \msg_kernel_error:nnxxxx
[148](#), [8779](#), [8779](#), 8807, 8809, 8811, 8813
- \msg_kernel_fatal:nn [8759](#), [8777](#)
- \msg_kernel_fatal:nnx . [8759](#), [8775](#), [13169](#)
- \msg_kernel_fatal:nnxx [8759](#), [8773](#)
- \msg_kernel_fatal:nnxxx [8759](#), [8771](#)
- \msg_kernel_fatal:nnxxxx
[148](#), [8759](#), [8759](#), [8772](#), [8774](#), [8776](#), [8778](#)
- \msg_kernel_info:nn [8814](#), 8856
- \msg_kernel_info:nnx [8814](#), 8854
- \msg_kernel_info:nnxx [8814](#), 8852
- \msg_kernel_info:nnxxx [8814](#), 8850
- \msg_kernel_info:nnxxxx
[148](#), [8814](#), 8837, 8851, 8853, 8855, 8857
- \msg_kernel_new:nnn [8751](#), [8753](#)
- \msg_kernel_new:nnnn [148](#), 7880, 7888,
[7891](#), [8164](#), [8171](#), [8751](#), 8751, 8858,
[8867](#), [8875](#), [8882](#), [8889](#), [8897](#), 8906,
[8913](#), [8920](#), [8927](#), [9102](#), [9575](#), [9578](#),
[9584](#), [9591](#), [9600](#), [9606](#), [9612](#), [9619](#),
[9626](#), [9632](#), [13094](#), [13102](#), [13110](#), [13116](#)
- \msg_kernel_set:nnn [8751](#), [8757](#)
- \msg_kernel_set:nnnn ... [148](#), [8751](#), [8755](#)
- \msg_kernel_warning:nn [8814](#), 8834
- \msg_kernel_warning:nnx [8814](#), 8832
- \msg_kernel_warning:nnxx [8814](#), 8830
- \msg_kernel_warning:nnxxx . . . [8814](#), 8828
- \msg_kernel_warning:nnxxxx
[148](#), [8814](#), 8815, 8829, 8831, 8833, 8835
- \msg_line_context [143](#)
- \msg_line_context:
..... [1183](#), [1183](#), [1202](#), [8484](#), 8485
- \msg_line_number [143](#)
- \msg_line_number: [8484](#), [8484](#), [8489](#), 9103
- \msg_log:nn [8678](#), 8983
- \msg_log:nnx [8678](#), 8982
- \msg_log:nnxx [8678](#), 8981
- \msg_log:nnxxx [8678](#), 8980
- \msg_log:nnxxxx [145](#), [8678](#), 8979
- \msg_log:x [147](#), [8569](#), 8569, 8672, 8680, 8841
- \msg_new:nnn [8421](#), 8430, 8754
- \msg_new:nnnn . [142](#), [8421](#), 8421, 8431, 8752
- \msg_newline [146](#)
- \msg_newline: [8482](#), [8482](#), 8542
- \msg_no_more_text:xxxx . [8607](#), 8609, 8613
- \msg_none:nn [8683](#)
- \msg_none:nnx [8683](#)
- \msg_none:nnxx [8683](#)
- \msg_none:nnxxx [8683](#)
- \msg_none:nnxxxx [145](#), [8683](#)
- \msg_redirect_class:nn . [146](#), [8745](#), 8745
- \msg_redirect_module:nnn [146](#), [8747](#), 8747
- \msg_redirect_name:nnn . [146](#), [8749](#), 8749
- \msg_see_documentation_text:n
..... [8589](#), [8589](#), [8624](#), [8635](#),
[8648](#), [8657](#), [8766](#), [8788](#), 8801, 8939
- \msg_set:nnn [8421](#), 8439, 8758
- \msg_set:nnnn . [142](#), [8421](#), 8432, 8440, 8756
- \msg_term:x ... [147](#), [8569](#), 8576, 8664, 8819
- \msg_trace:nn [8978](#), 8983
- \msg_trace:nnx [8978](#), 8982
- \msg_trace:nnxx [8978](#), 8981
- \msg_trace:nnxxx [8978](#), 8980
- \msg_trace:nnxxxx [8978](#), 8979
- \msg_two_newlines [146](#)
- \msg_two_newlines: [8482](#), 8483
- \msg_use:nnnnxxxx
..... [8597](#), [8688](#), 8688, 8817, 8839
- \msg_use_aux:nn [8688](#), 8721, 8723
- \msg_use_aux:nnn [8688](#), 8712, 8715
- \msg_use_code: [8688](#), 8690, 8730, 8738, 8742
- \msg_use_loop:n . . [8688](#), 8695, 8743, 8744
- \msg_use_loop:o [8688](#), 8739
- \msg_use_loop_check:nn
..... [8688](#), 8720, 8726, 8729, 8733
- \msg_warning:nn [8662](#)
- \msg_warning:nnx [8662](#)
- \msg_warning:nnxx [8662](#)
- \msg_warning:nnxxx [8662](#)
- \msg_warning:nnxxxx [145](#), [8662](#)
- \msg_warning_text:n
..... [144](#), [8584](#), 8587, 8666, 8821
- \mskip 463
- \muexpr 712
- \multiply 364
- \muskip 660
- \muskip_add:cn [4162](#)
- \muskip_add:Nn . [78](#), [4162](#), 4162, 4164, 4165
- \muskip_eval:n [79](#), [4172](#), 4172

\muskip_gadd:cn		4162	\nolimits		497
\muskip_gadd:Nn		78 , 4162 , 4164 , 4166	\nonscript		477
\muskip_gset:cn		4151	\nonstopmode		440
\muskip_gset:Nn		78 , 4151 , 4153 , 4155	\nulldelimiterspace		510
\muskip_gset_eq:cc		4156	\nullfont		628
\muskip_gset_eq:cN		4156	\number		637
\muskip_gset_eq:Nc		4156	\numexpr		709
\muskip_gset_eq:NN		78 , 4156 , 4159–4161			
\muskip_gsub:cn		4162	O		
\muskip_gsub:Nn		79 , 4162 , 4169 , 4171	\O		1821 , 2705
\muskip_gzero:c		4146	\omit		383
\muskip_gzero:N		78 , 4146 , 4148 , 4150	\openin		409
\muskip_new:c		4138	\openout		410
\muskip_new:N		78 , 4138 , 4139 , 4145	\or		22 , 69 , 406
\muskip_set:cn		4151	\or:		782 , 784 , 1309–1317 , 1509 ,
\muskip_set:Nn		78 , 4151 , 4151 , 4153 , 4154			3606–3630 , 11562 , 11564 , 11566 ,
\muskip_set_eq:cc		4156			11568 , 11648 , 11650 , 11652 , 11654
\muskip_set_eq:cN		4156	\outer		369
\muskip_set_eq:Nc		4156	\output		584
\muskip_set_eq:NN		78 , 4156 , 4156–4158	\outputpenalty		594
\muskip_show:c		4176	\over		471
\muskip_show:N		80 , 4176 , 4176 , 4177	\overfullrule		622
\muskip_sub:cn		4162	\overline		502
\muskip_sub:Nn		79 , 4162 , 4167 , 4169 , 4170	\overwithdelims		472
\muskip_use:c		4174	P		
\muskip_use:N		79 , 4173 , 4174 , 4174 , 4175	\P		1819 , 2705
\muskip_zero:c		4146	\package_check_loaded_expl:		780 , 1517 ,
\muskip_zero:N		78 , 4146 , 4146 , 4148 , 4149			1873 , 2382 , 2472 , 3192 , 3900 , 4196 ,
\muskipdef		358			5000 , 5537 , 6000 , 6321 , 6809 , 7903 ,
\mutogluue		718			7923 , 8416 , 8999 , 9648 , 9765 , 13129
N			\PackageError		219 , 235
\n		2701	\pagedepth		586
\name_primitive:NN		339 , 339 , 346–760	\pageDiscards		727
\negative_replication		2185	\pagefilllstretch		590
\newbox		6327	\pagefillstretch		589
\newcatcodetable		13183	\pagefilstretch		588
\newcount		3272	\pagegoal		592
\newdimen		3909	\pageshrink		591
\newlinechar		249 , 414	\pagestretch		587
\newmuskip		4142	\pagetotal		593
\newread		7970	\par		542
\newskip		4068	\parfillskip		573
\newwrite		7969	\parindent		566
\noalign		382	\parshape		558
\noboundary		517	\parshapedimen		708
\noexpand		35 , 39 , 40 , 166 , 169 , 172 , 174 ,	\parshapeindent		706
		175 , 184 , 187–189 , 191 , 193 , 194 ,	\parshapelength		707
		203 , 205–209 , 268 , 270 , 275 , 277 , 375	\parskip		565
\noindent		543	\patterns		648

<code>\pausing</code>	435	<code>\peek_catcode_ignore_spaces:N</code>	54
<code>\pdf@strcmp</code>	59	<code>\peek_catcode_remove:N</code>	2978
<code>\pdfcolorstack</code>	738	<code>\peek_catcode_remove:N</code>	55
<code>\pdfcompresslevel</code>	739	<code>\peek_catcode_remove_ignore_spaces:N</code>	
<code>\pdfcreationdate</code>	737	2978
<code>\pdfdecimaldigits</code>	740	<code>\peek_catcode_remove_ignore_spaces:N</code>	
<code>\pdfhorigin</code>	741	55
<code>\pdfinfo</code>	742	<code>\peek_charcode:N</code>	2994
<code>\pdfliteral</code>	743	<code>\peek_charcode:N</code>	55
<code>\pdfminorversion</code>	744	<code>\peek_charcode_ignore_spaces:N</code> ..	2994
<code>\pdfobjcompresslevel</code>	745	<code>\peek_charcode_ignore_spaces:N</code>	55
<code>\pdfoutput</code>	746	<code>\peek_charcode_remove:N</code>	2994
<code>\pdfpkresolution</code>	750	<code>\peek_charcode_remove:N</code>	55
<code>\pdfrestore</code>	747	<code>\peek_charcode_remove_ignore_spaces:N</code>	
<code>\pdfsave</code>	748	2994
<code>\pdfsetmatrix</code>	749	<code>\peek_charcode_remove_ignore_spaces:N</code>	
<code>\pdfstrcmp</code> . 33, 59, 230, 235, 238, 252, 753		55
<code>\pdftex_if_engine:</code>	1427	<code>\peek_def:nnnn</code>	2961, 2962,
<code>\pdftex_if_engine:F</code>	1431, 1442, 1456	2978, 2982, 2986, 2990, 2994, 2998,	
<code>\pdftex_if_engine:T</code>	1430, 1441, 1455	3002, 3006, 3010, 3014, 3018, 3022	
<code>\pdftex_if_engine:TF</code> ...	1432, 1443, 1457	<code>\peek_def_aux:nnnnn</code> 2961, 2964–2966, 2968	
<code>\pdftex_if_engine_p:</code> 1437, 1447, 1459, 1505		<code>\peek_execute_branches:</code>	2957, 2973
<code>\pdftex_if_engineTF</code>	21	<code>\peek_execute_branches_catcode:</code>	
<code>\pdftex_pdfcolorstack:D</code>	738	.. 2910, 2910, 2981, 2983, 2989, 2991	
<code>\pdftex_pdfcompresslevel:D</code>	739	<code>\peek_execute_branches_charcode:</code> ..	
<code>\pdftex_pdfcreationdate:D</code>	737	.. 2927, 2927, 2997, 2999, 3005, 3007	
<code>\pdftex_pdfdecimaldigits:D</code>	740	<code>\peek_execute_branches_charcode:NN</code> 2927	
<code>\pdftex_pdfhorigin:D</code>	741	<code>\peek_execute_branches_charcode_aux:NN</code>	
<code>\pdftex_pdfinfo:D</code>	742	2937, 2941
<code>\pdftex_pdfliteral:D</code>	743	<code>\peek_execute_branches_meaning:</code>	
<code>\pdftex_pdfminorversion:D</code>	744	.. 2910, 2919, 3013, 3015, 3021, 3023	
<code>\pdftex_pdfobjcompresslevel:D</code>	745	<code>\peek_execute_branches_N_type:</code>	
<code>\pdftex_pdfoutput:D</code>	746	3077, 3077, 3089, 3091, 3093
<code>\pdftex_pdfpkresolution:D</code>	750	<code>\peek_false:w</code>	2857, 2859, 2880,
<code>\pdftex_pdfrestore:D</code>	747	2898, 2916, 2924, 2935, 2946, 3085	
<code>\pdftex_pdfsave:D</code>	748	<code>\peek_gafter:NN</code>	3118, 3120
<code>\pdftex_pdfsetmatrix:D</code>	749	<code>\peek_gafter:Nw</code>	54, 2863, 3120
<code>\pdftex_pdftextrevision:D</code>	751	<code>\peek_ignore_spaces_execute_branches:</code>	
<code>\pdftex_pdfvorigin:D</code>	752	2950, 2950, 2960,
<code>\pdftex_strcmp:D</code>	753,	2985, 2993, 3001, 3009, 3017, 3025	
1464, 1470, 2407, 2414, 2445, 2454,		<code>\peek_ignore_spaces_execute_branches_aux:</code>	
2677, 4100, 4753, 4792, 9862, 9873		2950, 2954, 2959
<code>\pdftexrevision</code>	751	<code>\peek_meaning:N</code>	3010
<code>\pdfvorigin</code>	752	<code>\peek_meaning:N</code>	56
<code>\peek_after:NN</code>	3118, 3119	<code>\peek_meaning_ignore_spaces:N</code> ...	3010
<code>\peek_after:Nw</code>		<code>\peek_meaning_ignore_spaces:N</code>	56
54, 2861, 2861, 2886, 2904, 2960, 3119		<code>\peek_meaning_remove:N</code>	3010
<code>\peek_catcode:N</code>	2978	<code>\peek_meaning_remove:N</code>	56
<code>\peek_catcode:N</code>	54	<code>\peek_meaning_remove_ignore_spaces:N</code>	
<code>\peek_catcode_ignore_spaces:N</code> ...	2978	3010

\peek_meaning_remove_ignore_spaces:NTF	\prg_conditional_form_TF:nnn	1052
..... 56	\prg_define_quicksort:nnn	2295, 2295, 2370
\peek_N_type:..... 3077	\prg_do_nothing	9
\peek_N_type:F 3092	\prg_do_nothing: 1461, 1461, 4410, 4415,	
\peek_N_type:T 3090	4567, 4755, 5483, 5486, 5489, 5503,	
\peek_N_type:TF 3088	5510, 5933, 5937, 5944, 9868, 9879	
\peek_N_typeTF 58	\prg_generate_conditional_aux:nnNNnnnn	
\peek_tmp:w 2857, 2860, 2869, 2955 897,	
\peek_token_generic:NN 2871	905, 912, 920, 928, 937, 945, 954, 965	
\peek_token_generic:NNF 2890, 3093	\prg_generate_conditional_aux:nnw	
\peek_token_generic:NNT 2888, 3091 967, 973, 979	
\peek_token_generic:NNTF 2871, 2889, 2891, 3089	\prg_generate_conditional_parm_aux:nnNNnnnn	
\peek_token_remove_generic:NN ... 2892 965	
\peek_token_remove_generic:NNF .. 2908	\prg_generate_conditional_parm_aux:nw	
\peek_token_remove_generic:NNT .. 2906 965	
\peek_token_remove_generic:NNTF	\prg_generate_F_form_count:Nnnnn	
..... 2892, 2907, 2909 1010, 1026	
\peek_true:w 2857, 2857,	\prg_generate_F_form_parm:Nnnnn	981, 997
2875, 2896, 2914, 2922, 2944, 3086	\prg_generate_p_form_count:Nnnnn	
\peek_true_aux:w . 2857, 2858, 2868, 2897 1010, 1010	
\peek_true_remove:w 2865, 2865, 2896	\prg_generate_p_form_parm:Nnnnn	981, 981
\penalty 643	\prg_generate_T_form_count:Nnnnn	
\postdisplaypenalty 490 1010, 1018	
\predisplaydirection 734	\prg_generate_T_form_parm:Nnnnn	981, 989
\predisdisplaypenalty 489	\prg_generate_TF_form_count:Nnnnn	
\predisplaysize 488 1010, 1034	
\pretolerance 569	\prg_generate_TF_form_parm:Nnnnn	
\prevdepth 616 981, 1005	
\prevgraf 575	\prg_get_count_aux:nn	
\prg_case_dim:nnn 39, 2108, 2108 926, 935, 944, 952, 963, 963	
\prg_case_dim_aux:nnn .. 2108, 2111, 2113	\prg_get_parm_aux:nw	
\prg_case_dim_aux:nw 2108, 2114, 2115, 2119 895, 903, 911, 918, 963, 964	
\prg_case_end:nw 2094,	\prg_new_conditional:Nnn	924,
2094, 2105, 2118, 2129, 2141, 2152	933, 1875, 2423, 2431, 2443, 2452, 8377	
\prg_case_int:nnn 38, 2095, 2095, 3495, 3501	\prg_new_conditional:Npnn	
\prg_case_int_aux:nnn .. 2095, 2098, 2100 32, 893, 901, 1400,	
\prg_case_int_aux:nw 2095, 2101, 2102, 2106	1462, 1468, 1875, 1903, 1917, 2250,	
\prg_case_str:nnn 39, 2121, 2121, 2132, 4937	2252, 2254, 2256, 2588, 2593, 2598,	
\prg_case_str:onn 2121	2603, 2610, 2616, 2621, 2626, 2631,	
\prg_case_str:xxn 2121, 2133	2636, 2641, 2646, 2651, 2656, 2670,	
\prg_case_str_aux:nw 2121, 2124, 2126, 2130	2684, 2689, 2710, 2717, 2724, 2735,	
\prg_case_str_x_aux:nw	2746, 2757, 2768, 2777, 2784, 2802,	
..... 2121, 2136, 2138, 2142	3338, 3408, 3416, 3424, 3954, 3959,	
\prg_case_tl:cnn 2144	4097, 4107, 4431, 4453, 4474, 4476,	
\prg_case_tl:Nnn ... 39, 2144, 2144, 2155	4666, 4682, 4698, 4732, 4734, 4750,	
\prg_case_tl_aux:Nw 2144, 2147, 2149, 2153	4803, 4805, 6120, 6132, 6393, 6395,	
\prg_conditional_form_F:nnn 1054	6405, 9520, 9561, 9567, 12822, 12830	
\prg_conditional_form_p:nnn 1051	\prg_new_eq_conditional:NN	34
\prg_conditional_form_T:nnn 1053	\prg_new_eq_conditional:NNn	
 959, 961, 1875,	

- 5092, 5094, 5738–5743, 6311–6314
 \prg_new_map_functions:Nn . . . [2373](#), [2374](#)
 \prg_new_protected_conditional:Nnn .
 [924](#), [950](#), [1875](#), [9715](#)
 \prg_new_protected_conditional:Npnn
 [33](#), [893](#), [916](#),
[1875](#), [4488](#), [4509](#), [5096](#), [5323](#), [5331](#),
[5344](#), [5351](#), [5358](#), [5365](#), [5744](#), [5748](#),
[6161](#), [6235](#), [6241](#), [12838](#), [12855](#), [13042](#)
 \prg_quicksort:n [42](#), [2370](#)
 \prg_quicksort_compare:nnTF
 [42](#), [2371](#), [2372](#)
 \prg_quicksort_function:n [42](#), [2371](#), [2371](#)
 \prg_replicate:nn [39](#),
[2156](#), [2156](#), [8351](#), [10186](#), [10222](#), [10292](#)
 \prg_replicate_ [2156](#), [2167](#)
 \prg_replicate_0:n [2156](#)
 \prg_replicate_1:n [2156](#)
 \prg_replicate_2:n [2156](#)
 \prg_replicate_3:n [2156](#)
 \prg_replicate_4:n [2156](#)
 \prg_replicate_5:n [2156](#)
 \prg_replicate_6:n [2156](#)
 \prg_replicate_7:n [2156](#)
 \prg_replicate_8:n [2156](#)
 \prg_replicate_9:n [2156](#)
 \prg_replicate_aux:N [2156](#), [2163](#), [2164](#), [2166](#)
 \prg_replicate_first_-:n [2156](#)
 \prg_replicate_first_0:n [2156](#)
 \prg_replicate_first_1:n [2156](#)
 \prg_replicate_first_2:n [2156](#)
 \prg_replicate_first_3:n [2156](#)
 \prg_replicate_first_4:n [2156](#)
 \prg_replicate_first_5:n [2156](#)
 \prg_replicate_first_6:n [2156](#)
 \prg_replicate_first_7:n [2156](#)
 \prg_replicate_first_8:n [2156](#)
 \prg_replicate_first_9:n [2156](#)
 \prg_replicate_first_aux:N
 [2156](#), [2159](#), [2165](#)
 \prg_return_false [34](#)
 \prg_return_false:
 [889](#), [891](#), [1097](#), [1102](#), [1115](#),
[1120](#), [1128](#), [1145](#), [1403](#), [1466](#), [1471](#),
[1875](#), [1908](#), [1922](#), [2251](#), [2253](#), [2255](#),
[2257](#), [2428](#), [2436](#), [2449](#), [2458](#), [2591](#),
[2596](#), [2601](#), [2606](#), [2613](#), [2619](#), [2624](#),
[2629](#), [2634](#), [2639](#), [2644](#), [2649](#), [2654](#),
[2659](#), [2680](#), [2687](#), [2694](#), [2696](#), [2716](#),
[2723](#), [2727](#), [2734](#), [2738](#), [2745](#), [2756](#),
[2760](#), [2767](#), [2776](#), [2783](#), [2792](#), [2805](#),
[2824](#), [2841](#), [2850](#), [3357](#), [3365](#), [3371](#),
[3381](#), [3389](#), [3395](#), [3403](#), [3413](#), [3421](#),
[3427](#), [3957](#), [3965](#), [4104](#), [4115](#), [4446](#),
[4458](#), [4471](#), [4481](#), [4498](#), [4513](#), [4675](#),
[4695](#), [4710](#), [4718](#), [4728](#), [4745](#), [4759](#),
[4796](#), [5109](#), [5319](#), [5758](#), [6125](#), [6150](#),
[6165](#), [6239](#), [6245](#), [6394](#), [6396](#), [6406](#),
[8387](#), [8610](#), [9525](#), [9565](#), [9571](#), [9719](#),
[12827](#), [12835](#), [12872](#), [12886](#), [12890](#),
[12894](#), [12898](#), [12910](#), [12914](#), [12929](#),
[12944](#), [12962](#), [12971](#), [12979](#), [12994](#),
[13003](#), [13011](#), [13056](#), [13062](#), [13068](#),
[13074](#), [13080](#), [13086](#), [13091](#), [13092](#)
 \prg_return_true [34](#)
 \prg_return_true: [889](#), [889](#), [1100](#),
[1117](#), [1125](#), [1130](#), [1143](#), [1148](#), [1403](#),
[1466](#), [1471](#), [1875](#), [1906](#), [1920](#), [2251](#),
[2253](#), [2255](#), [2257](#), [2426](#), [2434](#), [2447](#),
[2456](#), [2591](#), [2596](#), [2601](#), [2606](#), [2613](#),
[2619](#), [2624](#), [2629](#), [2634](#), [2639](#), [2644](#),
[2649](#), [2654](#), [2659](#), [2678](#), [2687](#), [2694](#),
[2716](#), [2723](#), [2734](#), [2745](#), [2756](#), [2767](#),
[2776](#), [2783](#), [2792](#), [2822](#), [2848](#), [3355](#),
[3363](#), [3373](#), [3379](#), [3387](#), [3397](#), [3405](#),
[3411](#), [3419](#), [3429](#), [3957](#), [3963](#), [4102](#),
[4114](#), [4444](#), [4456](#), [4469](#), [4479](#), [4495](#),
[4513](#), [4673](#), [4693](#), [4708](#), [4726](#), [4747](#),
[4757](#), [4794](#), [5112](#), [5327](#), [5335](#), [5348](#),
[5355](#), [5362](#), [5369](#), [5758](#), [6123](#), [6148](#),
[6170](#), [6251](#), [6394](#), [6396](#), [6406](#), [8382](#),
[8385](#), [8391](#), [8611](#), [9524](#), [9564](#), [9570](#),
[9720](#), [12825](#), [12833](#), [12883](#), [12917](#),
[12926](#), [12940](#), [12958](#), [12966](#), [12976](#),
[12990](#), [12998](#), [13008](#), [13056](#), [13062](#),
[13068](#), [13074](#), [13080](#), [13086](#), [13092](#)
 \prg_set_conditional:Nnn . [924](#), [924](#), [1875](#)
 \prg_set_conditional:Npnn
 [32](#), [893](#), [893](#), [1094](#),
[1106](#), [1122](#), [1134](#), [1875](#), [4441](#), [8607](#)
 \prg_set_eq_conditional:NN [34](#)
 \prg_set_eq_conditional:NNn
 [959](#), [959](#), [1875](#)
 \prg_set_eq_conditional_aux:NNNn
 [960](#), [962](#), [1039](#), [1039](#)
 \prg_set_eq_conditional_aux:NNNw
 [1039](#), [1040](#), [1041](#), [1049](#)
 \prg_set_map_functions:Nn . . . [2373](#), [2375](#)
 \prg_set_protected_conditional:Nnn .
 [924](#), [943](#), [1875](#)

<code>\prg_set_protected_conditional:Npnn</code>	<code>\prop_get:NnNF</code>	6173, 6176
..... 33, 893, 909, 1875	<code>\prop_get:NnNT</code>	6172, 6175
<code>\prg_stepwise_aux:NNnnnn</code>	<code>\prop_get:NnNTF</code>	117, 6174, 6177
..... 2226, 2228, 2234, 2243	<code>\prop_get:NoN</code>	6052, 6161
<code>\prg_stepwise_function:nnnN</code>	<code>\prop_get:NVN</code>	6052, 6161
..... 40, 2196, 2196, 2247	<code>\prop_get_aux:Nnnn</code>	6052, 6055, 6058
<code>\prg_stepwise_function_decr:nnnN</code>	<code>\prop_get_aux_true:Nnnn</code>	6161, 6164, 6167
..... 2196, 2203, 2216, 2221	<code>\prop_get_gdel:NnN</code>	6299, 6300
<code>\prg_stepwise_function_incr:nnnN</code>	<code>\prop_get_Nn_aux:nwn</code>	6273, 6275, 6280, 6284
..... 2196, 2202, 2207, 2212	<code>\prop_gget:cnN</code>	6291
<code>\prg_stepwise_inline:nnnn</code>	<code>\prop_gget:cVN</code>	6291
..... 40, 2226, 2226, 13258, 13263	<code>\prop_gget:NnN</code>	6291, 6292, 6296, 6297
<code>\prg_stepwise_variable:nnnNn</code>	<code>\prop_gget:NVN</code>	6291
..... 40, 2226, 2232	<code>\prop_gget_aux:Nnnn</code>	6291, 6293, 6294
<code>\prg_variable_get_scope:N</code>	<code>\prop_gpop:cnN</code>	6062, 6235
..... 41, 2263, 2269	<code>\prop_gpop:coN</code>	6062
<code>\prg_variable_get_scope_aux:w</code>	<code>\prop_gpop:NnN</code>	116, 6062, 6068, 6081, 6082, 6241, 6300
..... 2263, 2271, 2274	<code>\prop_gpop:NnNF</code>	6257
<code>\prg_variable_get_type:N</code>	<code>\prop_gpop:NnNT</code>	6256
..... 41, 2263, 2283	<code>\prop_gpop:NnNTF</code>	119, 6258
<code>\prg_variable_get_type:w</code>	<code>\prop_gpop:NoN</code>	6062
..... 2263	<code>\prop_gput:ccx</code>	6307
<code>\prg_variable_get_type_aux:w</code>	<code>\prop_gput:cnN</code>	6083
..... 2285, 2288, 2292	<code>\prop_gput:cno</code>	6083
<code>\prop_clear:c</code>	<code>\prop_gput:cnV</code>	6083
..... 6006, 6007, 7259	<code>\prop_gput:cnx</code>	6083
<code>\prop_clear:N</code>	<code>\prop_gput:con</code>	6083
..... 114, 6006, 6006, 6011, 7757	<code>\prop_gput:coo</code>	6083
<code>\prop_clear_new:c</code>	<code>\prop_gput:cVn</code>	6083
..... 6010, 6888, 6889, 8595	<code>\prop_gput:cVV</code>	6083
<code>\prop_clear_new:N</code>	<code>\prop_gput:Nnn</code>	115, 6083, 6084, 6101, 6103, 6308
..... 114, 6010, 6010, 6012	<code>\prop_gput:Nno</code>	6083
<code>\prop_del:cn</code>	<code>\prop_gput:NnV</code>	6083
..... 6042	<code>\prop_gput:Nnx</code>	6083
<code>\prop_del:cV</code>	<code>\prop_gput:Non</code>	6083
..... 6042	<code>\prop_gput:Noo</code>	6083
<code>\prop_del:Nn</code>	<code>\prop_gput:NVn</code>	6083, 7991, 8004
..... 116, 6042, 6042, 6048, 6049, 7752, 7755, 7760	<code>\prop_gput:NVV</code>	6083
<code>\prop_del:NV</code>	<code>\prop_gput_if_new:cnN</code>	6105
..... 6042	<code>\prop_gput_if_new:Nnn</code>	115, 6105, 6107, 6119
<code>\prop_del_aux:NNnnn</code>	<code>\prop_gset_eq:cc</code>	6016, 6023, 7026, 7028
..... 6042, 6043, 6045, 6046	<code>\prop_gset_eq:cN</code>	6016, 6022, 6890, 6892
<code>\prop_display:c</code>	<code>\prop_gset_eq:Nc</code>	6016, 6021
..... 6287, 6289	<code>\prop_gset_eq:NN</code>	115, 6016, 6020
<code>\prop_display:N</code>	<code>\prop_if_empty:c</code>	6120
..... 6287, 6288	<code>\prop_if_empty:N</code>	6120, 6120
<code>\prop_gclear:c</code>	<code>\prop_if_empty:Nf</code>	6131
..... 6006, 6009, 6015	<code>\prop_if_empty:NT</code>	6130
<code>\prop_gclear:N</code>		
..... 114, 6006, 6008, 6014		
<code>\prop_gclear_new:c</code>		
..... 6010, 6015		
<code>\prop_gclear_new:N</code>		
..... 114, 6010, 6013		
<code>\prop_gdel:cn</code>		
..... 6042		
<code>\prop_gdel:cV</code>		
..... 6042		
<code>\prop_gdel:Nn</code>		
..... 116, 6042, 6044, 6050, 6051		
<code>\prop_gdel:NV</code>		
..... 6042, 8108, 8121		
<code>\prop_get:cn</code>		
..... 6273		
<code>\prop_get:cnN</code>		
..... 6052, 6161, 8735		
<code>\prop_get:cnNF</code>		
..... 7002		
<code>\prop_get:coN</code>		
..... 6161		
<code>\prop_get:cVN</code>		
..... 6052, 6161		
<code>\prop_get:Nn</code>		
..... 119, 6273, 6273, 6286		
<code>\prop_get:NnN</code>		
..... 116, 6052, 6052, 6060, 6061,		
6161, 6161, 7705, 7709, 7789, 7793		

\prop_if_empty:NTF	\prop_pop:NnNT	6253
..... 117, 6129, 6209, 8130, 8150	\prop_pop:NnNTF	117, 6255
\prop_if_empty_p:N	\prop_pop:NoN	6062
\prop_if_eq:cc	\prop_pop_aux:NNNnnn 6062, 6065, 6071, 6074	
\prop_if_eq:cN	\prop_pop_aux_true:NNNnnn	6235, 6238, 6244, 6247
\prop_if_eq:Nc 6083, 7080, 8746, 8748	
\prop_if_eq:NN	\prop_put:cno	6083
\prop_if_in:cc	\prop_put:cnV	6083
\prop_if_in:cn	\prop_put:cnx	6083, 7086, 7088, 7090, 7092,
\prop_if_in:cnTF 7097, 7102, 7107, 7114, 7121, 7354,	
\prop_if_in:co 7467, 7525, 7533, 7600, 7614, 7621	
\prop_if_in:cV	\prop_put:con	6083
\prop_if_in:Nn	\prop_put:coo	6083
\prop_if_in:NnF 6157, 6158, 6304, 8012, 8020	\prop_put:cVn	6083
\prop_if_in:NnT	\prop_put:cVV	6083
\prop_if_in:NnTF 117, 6159, 6160, 6305, 8719	\prop_put:Nnn .. 115, 6083, 6083, 6097,	
\prop_if_in:No 6099, 6816–6819, 7631, 7633, 7635,	
\prop_if_in:NV 7637, 7639, 7641, 7643, 7645, 7647,	
\prop_if_in:NVT 7649, 7651, 7653, 7655, 7657, 7659,	
\prop_if_in_aux:Nw 7661, 7663, 7665, 7953–7956, 8750	
\prop_if_in_aux:nwn 6132, 6134, 6139, 6143	\prop_put:Nno 6083, 6822–6824, 6826–6831	
\prop_if_in_p:Nn	\prop_put:NnV	6083
\prop_map_break	\prop_put:Nnx	6083, 7448, 7450, 7453, 7455, 7461
\prop_map_break: . 6186, 6204, 6204, 6267	\prop_put:Non	6083
\prop_map_break:n	\prop_put:Noo	6083
\prop_map_function:cc	\prop_put:NVn	6083
\prop_map_function:cN	\prop_put:NVV	6083
\prop_map_function:Nc	\prop_put_aux:NNnn	6083–6085
\prop_map_function:NN	\prop_put_aux:NNnnnnn .. 6083, 6087, 6089	
..... 6178, 6191, 6192, 6221, 8138, 8158	\prop_put_if_new:cnn	6105
\prop_map_function_aux:Nwn	\prop_put_if_new:Nnn 115, 6105, 6105, 6118	
..... 6178, 6180, 6183, 6189	\prop_put_if_new_aux:NNnn 6106, 6108, 6109	
\prop_map_inline:cn	\prop_set_eq:cc 6016, 6019, 7019, 7021, 7289	
..... 6194, 7330, 7349, 7418, 7420, 7440,	\prop_set_eq:cN .. 6016, 6018, 7012, 7014	
..... 7442, 7505, 7559, 7561, 7565, 7567	\prop_set_eq:Nc	6016, 6017, 7747
\prop_map_inline:Nn	\prop_set_eq:NN	115, 6016, 6016
..... 118, 6194, 6194, 6203, 7423, 7518, 7758, 7767	\prop_show:c	6207, 6289
\prop_map_tokens:cn	\prop_show:N .. 119, 6207, 6207, 6234, 6288	
\prop_map_tokens:Nn 119, 6259, 6259, 6272	\prop_show_aux:n	6207
\prop_map_tokens_aux:nwn	\prop_show_aux:nn	6221, 6226
..... 6259, 6261, 6264, 6270	\prop_show_aux:w	6207, 6223, 6233, 8140, 8160
\prop_new:c	\prop_split:Nnn 120, 6036, 6036, 6087, 6293	
\prop_new:N 114, 6004, 6004, 6011, 6014,	\prop_split:NnTF	120,
..... 6815, 6820, 7405, 7630, 7671, 7679, 6024, 6024, 6038, 6043, 6045, 6054,	
..... 7950, 7951, 8591, 8592, 8814, 8836 6064, 6070, 6111, 6163, 6237, 6243	
\prop_pop:cnN	\prop_split_aux:nnnn .. 6024, 6030, 6034	
\prop_pop:coN		
\prop_pop:NnN		
..... 116, 6062, 6062, 6079, 6080, 6235, 6235		
\prop_pop:NnNF		

- `\prop_split_aux:NnTF` . . . 6024, 6025, 6026
`\prop_split_aux:w` 6024, 6028, 6031, 6035
`\protect` 235
`\protected` 68,
82, 98, 104, 126, 133, 141, 143, 147,
152, 157, 213, 266, 273, 292, 325, 736
`\protected@edef` 8265, 8537
`\ProvidesClass` 154
`\ProvidesExplClass` 6, 146, 152
`\ProvidesExplFile` 6, 146, 157
`\ProvidesExplPackage`
. 6, 146, 147, 333, 778, 1515,
1871, 2380, 2470, 3190, 3898, 4194,
4998, 5535, 5998, 6319, 6807, 7901,
7921, 8414, 8997, 9646, 9763, 13127
`\ProvidesFile` 159
`\ProvidesPackage` 47, 149
- Q**
- `\q` 1075, 2007, 2012
`\q_mark` 43, 1833, 1835, 1839,
2385, 2386, 3348, 3350, 4394, 4402,
4406, 4417, 4603, 4606, 4607, 4613,
4617, 4619, 4621, 4647, 4648, 5581,
5582, 5596, 5609, 5613, 5715, 5721,
5734, 5833, 5840, 6029, 6031, 6032
`\q_nil` 874, 877, 2298, 2302,
2385, 2385, 2425, 2446, 3719, 3741,
4455, 4467, 4468, 4605, 4609, 4626,
4629, 4632, 4671, 4687, 4707, 5580,
5584, 5591, 5659, 5668, 9023, 9054,
9074, 9081, 9086, 13048, 13055,
13061, 13067, 13073, 13079, 13085
`\q_no_value` 43, 2036, 2385, 2387,
2433, 2455, 6040, 6056, 6066, 6072,
9023, 9031, 9036, 9053, 9059, 9290
`\q_prop` 119, 6002, 6002, 6003, 6029, 6030,
6032, 6094, 6115, 6136, 6139, 6147,
6181, 6183, 6262, 6264, 6277, 6280
`\q_recursion_stop` 44, 876,
879, 971, 1040, 1772, 2094, 2101,
2114, 2124, 2136, 2147, 2389, 2390,
4519, 4523, 4538, 4548, 4553, 4590,
4591, 4594, 5597, 5774, 5813, 5833,
5889, 6137, 6145, 6181, 6262, 6278
`\q_recursion_tail`
. 2389, 2389, 2393, 2399,
2408, 2415, 4519, 4523, 4538, 4548,
4553, 4590, 5597, 5774, 5813, 5833,
5889, 6137, 6181, 6185, 6262, 6266
- `\q_stop` 43, 875, 878,
1075, 1077, 1085, 1087, 1298, 1302,
1794, 1836, 1839, 2036, 2039, 2272,
2274, 2286, 2288, 2292, 2298, 2302,
2364, 2385, 2388, 2673, 2675, 2713,
2715, 2720, 2722, 2730, 2733, 2741,
2744, 2752, 2755, 2763, 2766, 2772,
2775, 2780, 2782, 2788, 2791, 2808,
2811, 2814, 2836, 3029, 3036, 3045,
3054, 3339, 3340, 3348, 3352, 3360,
3368, 3376, 3384, 3392, 3400, 3787,
3824, 4402, 4417, 4611, 4632, 4642,
4643, 4645, 4647, 4648, 4655, 4663,
4665, 4671, 4687, 4707, 4991, 4993,
5122, 5125, 5135, 5138, 5326, 5347,
5354, 5435, 5437, 5442, 5586, 5591,
5649, 5650, 5659, 5661, 5721, 5915,
5948, 6029, 6032, 8274, 8365, 8970,
8972, 9035, 9040, 9042, 9053, 9058,
9060, 9083, 9086, 9154, 9157, 9163,
9172, 9182, 9838, 9839, 9858, 9863,
9868, 9874, 9879, 9885, 9891, 9893,
9894, 9897, 9901, 9905, 9941, 9946,
10163, 10165, 10180, 10182, 10183,
10189, 10196, 10198, 10201, 10203,
10204, 10206, 10208, 10210, 10212,
10214, 10216, 10218, 10219, 10228,
10230, 10240, 10242, 10253, 10260,
10262–10264, 10266, 10268, 10270,
10272, 10274, 10276, 10278, 10280,
10289, 10295, 10297, 10307, 10309,
10316, 10318–10320, 10326, 10331,
10336, 10341, 10346, 10351, 10356,
10361, 10371, 10376, 10377, 10388,
10390, 10409, 10429, 10899, 10904,
11016, 11069, 11246, 11251, 11258,
11261, 13048, 13052, 13055, 13058,
13061, 13064, 13067, 13070, 13073,
13076, 13079, 13082, 13085, 13088
- `\q_tl_act_mark`
. 94, 2465, 2465, 4811, 4814, 4831
`\q_tl_act_stop` 94,
2465, 2466, 4811, 4814, 4818, 4827,
4829, 4835, 4839, 4842, 4846, 4849
`\quark_if_nil:N` 2423, 2423
`\quark_if_nil:n` 2443, 2443
`\quark_if_nil:nF` 2464
`\quark_if_nil:nTF` 2305, 2309, 2463
`\quark_if_nil:NnTF` 44, 3722, 3744, 9078
`\quark_if_nil:nTF` 44, 2313, 2322, 2331,

2340, 2462, 5664, 13054, 13060, 13066, 13072, 13078, 13084, 13090	S
\quark_if_nil:o 2443	\S 2705
\quark_if_nil:oF 9033	\savecatcodetable 760
\quark_if_nil:V 2443	\savingshyphcodes 725
\quark_if_nil_p:n 2461	\savingsvdiscards 726
\quark_if_no_value:c 2423	\scan_align_safe_stop 41
\quark_if_no_value:cF 9510	\scan_align_safe_stop: . 2262, 2262, 3860
\quark_if_no_value:N 2431	\scan_stop 9
\quark_if_no_value:n 2443, 2452	\scan_stop: . . 308, 322, 809, 809, 1043,
\quark_if_no_value:N. 2423	1067, 1096, 1114, 1124, 1142, 1162,
\quark_if_no_value:NF 2441	1556, 1817–1825, 2264, 2278, 2609,
\quark_if_no_value:NT 2440, 13202	2686, 3038, 3047, 3056, 3089, 3091,
\quark_if_no_value:NTF 44, 2041, 2442, 7707, 7711, 7791, 7795	3093, 4077, 4088, 4093, 4118, 4123,
\quark_if_no_value:nTF 44	4126, 4152, 4163, 4168, 4173, 4187,
\quark_if_no_value_p:N 2439	4188, 4303–4306, 4663, 6433, 6469,
\quark_if_recursion_tail_aux:w .. 2405	6470, 7685, 7740, 7992, 8005, 9846–
\quark_if_recursion_tail_stop:N 44, 2391, 2391, 4559	9848, 9888, 9899, 9907, 9912, 9948,
\quark_if_recursion_tail_stop:n 45, 2405, 2405,	9949, 9965, 9973, 10012, 10021,
2421, 4527, 5606, 5779, 5818, 5837	10037, 10046, 10109, 10604, 10616,
\quark_if_recursion_tail_stop:o .. 2405	10617, 10749, 10753, 10765, 10769,
\quark_if_recursion_tail_stop_do:Nn 45, 2391, 2397	10782, 10786, 10828, 10889, 10893,
\quark_if_recursion_tail_stop_do:nn 45, 2405, 2412, 2422, 4593	10900–10902, 10910, 10921, 10971,
\quark_if_recursion_tail_stop_do:on 2405	11073, 11148, 11156, 11198, 11212,
\quark_new:N 43, 2384, 2384–2390, 2465, 2466, 6002	11216, 11254, 11255, 11264, 11265,
R	11282, 11290, 11298, 11314, 11328,
\R 1820, 2705	11341, 11696, 12019, 12029, 12073,
\radical 464	12149–12152, 12173, 12374, 12400,
\raise 604	12437, 12445, 12453, 12535, 12537,
\read 411	12539, 12574, 12582, 12786, 12788,
\readline 686	12790, 12792, 12794, 12808, 13204
\relax 3–6, 9, 13, 62, 70–80, 84–93, 96, 101, 131, 133, 141, 143, 229, 233, 249, 281–289, 446	\scantokens 684
\relpenalty 507	\scriptfont 630
\RequirePackage 57, 58	\scriptscriptfont 631
\reverse_if:N 22, 782, 787, 3985–3987, 4662	\scriptscriptstyle 476
\right 505	\scriptspace 516
\righthyphenmin 561	\scriptstyle 475
\rightskip 563	\scrollmode 441
\romannumeral 638	\seq_break 103
\rule 7689, 7744	\seq_break: . . . 5160, 5192, 5197, 5197,
	5199, 5206, 5320, 5328, 5335, 5348,
	5355, 5362, 5369, 5406, 5426, 5434
	\seq_break:n 104, 5109, 5112, 5197, 5198, 5200, 5414
	\seq_break_point:n .. 104, 5110, 5123,
	5136, 5149, 5177, 5197, 5198, 5201,
	5201, 5213, 5248, 5259, 5329, 5336,
	5349, 5356, 5363, 5370, 5408, 5427
	\seq_clear:c 5011, 5012
	\seq_clear:N . . . 95, 5011, 5011, 5056, 8692
	\seq_clear_new:c 5015, 5016

<code>\seq_clear_new:N</code>	95, 5015, 5015	<code>\seq_gpush:Nn</code>	101, 5263, 5273
<code>\seq_concat:ccc</code>	5027	<code>\seq_gpush:No</code>	5263, 5276
<code>\seq_concat:NNN</code> 96, 5027, 5027, 5031, 9696		<code>\seq_gpush:NV</code>	5263, 5274, 9733
<code>\seq_display:c</code>	5528, 5530	<code>\seq_gpush:Nv</code>	5263, 5275
<code>\seq_display:N</code>	5528, 5529	<code>\seq_gpush:Nx</code>	5263, 5277, 13190
<code>\seq_gclear:c</code>	5011, 5014	<code>\seq_gput_left:cn</code>	5041, 5278
<code>\seq_gclear:N</code>	95, 5011, 5013	<code>\seq_gput_left:co</code>	5041, 5281
<code>\seq_gclear_new:c</code>	5015, 5018	<code>\seq_gput_left:cV</code>	5041, 5279
<code>\seq_gclear_new:N</code>	95, 5015, 5017	<code>\seq_gput_left:cv</code>	5041, 5280
<code>\seq_gconcat:ccc</code>	5027	<code>\seq_gput_left:cx</code>	5041, 5282
<code>\seq_gconcat:NNN</code> 96, 5027, 5029, 5032, 9757		<code>\seq_gput_left:Nn</code>	
<code>\seq_get:cN</code>	5283, 5284 96, 5041, 5041, 5045, 5046, 5273	
<code>\seq_get:NN</code>	100, 5283, 5283	<code>\seq_gput_left:No</code>	5041, 5276
<code>\seq_get_left:cN</code>	5119, 5284, 5323, 5526	<code>\seq_gput_left:NV</code>	5041, 5274
<code>\seq_get_left:NN</code>	97, 5119,	<code>\seq_gput_left:Nv</code>	5041, 5275
5119, 5127, 5283, 5323, 5525		<code>\seq_gput_left:Nx</code>	5041, 5277
<code>\seq_get_left:NnF</code>	5339	<code>\seq_gput_right:cn</code>	5041
<code>\seq_get_left:NNT</code>	5338	<code>\seq_gput_right:co</code>	5041
<code>\seq_get_left:NNTF</code>	101, 5340	<code>\seq_gput_right:cV</code>	5041
<code>\seq_get_left_aux:NnwN</code>	5119, 5122, 5125	<code>\seq_gput_right:cv</code>	5041
<code>\seq_get_left_aux:Nw</code>	5326	<code>\seq_gput_right:cx</code>	5041
<code>\seq_get_right:cN</code>	5145, 5323	<code>\seq_gput_right:Nn</code>	
<code>\seq_get_right:NN</code> 96, 5041, 5043, 5047, 5048	
. 97, 5145, 5145, 5168, 5323, 5331		<code>\seq_gput_right:No</code>	5041
<code>\seq_get_right:NnF</code>	5342	<code>\seq_gput_right:NV</code>	5041, 9667
<code>\seq_get_right:NNT</code>	5341	<code>\seq_gput_right:Nv</code>	5041
<code>\seq_get_right:NNTF</code>	101, 5343	<code>\seq_gput_right:Nx</code>	5041, 9728
<code>\seq_get_right_aux:NN</code>		<code>\seq_gremove_all:cn</code>	5066
. 5145, 5148, 5151, 5334		<code>\seq_gremove_all:Nn</code>	98, 5066, 5068, 5091
<code>\seq_get_right_loop:nn</code>		<code>\seq_gremove_duplicates:c</code>	5050
. 5145, 5154, 5163, 5166, 5183		<code>\seq_gremove_duplicates:N</code>	
<code>\seq_gpop:cN</code>	5283, 5288 98, 5050, 5052, 5065	
<code>\seq_gpop:NN</code>	100, 5283, 5287, 9736, 13201	<code>\seq_gset_eq:cc</code>	5019, 5026
<code>\seq_gpop_left:cN</code>	5128, 5288, 5344	<code>\seq_gset_eq:cN</code>	5019, 5025
<code>\seq_gpop_left:NN</code>		<code>\seq_gset_eq:Nc</code>	5019, 5024
97, 5128, 5130, 5144, 5287, 5344, 5351		<code>\seq_gset_eq:NN</code>	96, 5019, 5023, 5053
<code>\seq_gpop_left:NnF</code>	5376	<code>\seq_gset_from_clist:cc</code>	5446
<code>\seq_gpop_left:NNT</code>	5375	<code>\seq_gset_from_clist:cN</code>	5446
<code>\seq_gpop_left:NNTF</code>	101, 5377	<code>\seq_gset_from_clist:cn</code>	5446
<code>\seq_gpop_right:cN</code>	5169, 5344	<code>\seq_gset_from_clist:Nc</code>	5446
<code>\seq_gpop_right:NN</code>		<code>\seq_gset_from_clist:NN</code>	
. 97, 5169, 5171, 5196, 5344, 5365	 103, 5446, 5456, 5470, 5471	
<code>\seq_gpop_right:NnF</code>	5382	<code>\seq_gset_from_clist:Nn</code> 5446, 5461, 5472	
<code>\seq_gpop_right:NNT</code>	5381	<code>\seq_gset_reverse:N</code>	103, 5473, 5477
<code>\seq_gpop_right:NNTF</code>	102, 5383	<code>\seq_gset_split:Nnn</code>	103, 5492, 5494
<code>\seq_gpush:cn</code>	5263, 5278	<code>\seq_if_empty:c</code>	5092, 5094
<code>\seq_gpush:co</code>	5263, 5281	<code>\seq_if_empty:N</code>	5092, 5092
<code>\seq_gpush:cV</code>	5263, 5279	<code>\seq_if_empty:NNTF</code>	98, 5292, 5960
<code>\seq_gpush:cv</code>	5263, 5280		
<code>\seq_gpush:cx</code>	5263, 5282		

- \seq_if_empty_break_return_false:N [5316](#), [5316](#),
[5325](#), [5333](#), [5346](#), [5353](#), [5360](#), [5367](#)
- \seq_if_empty_err_break:N
[103](#), [5121](#), [5134](#), [5147](#), [5175](#), [5202](#), [5202](#)
- \seq_if_in:cn [5096](#)
- \seq_if_in:co [5096](#)
- \seq_if_in:cV [5096](#)
- \seq_if_in:cv [5096](#)
- \seq_if_in:cx [5096](#)
- \seq_if_in:Nn [5096](#), [5096](#)
- \seq_if_in:NnF [5059](#), [5115](#), [5116](#), [9741](#)
- \seq_if_in:NnT [5113](#), [5114](#)
- \seq_if_in:NnTF [98](#), [5117](#), [5118](#), [8697](#)
- \seq_if_in:No [5096](#)
- \seq_if_in:Nv [5096](#)
- \seq_if_in:Nx [5096](#)
- \seq_if_in_aux: [5096](#), [5105](#), [5112](#)
- \seq_item:cn [5394](#)
- \seq_item:n [103](#), [5002](#), [5002](#), [5034](#),
[5036](#), [5042](#), [5044](#), [5084](#), [5101](#), [5125](#),
[5138](#), [5181](#), [5225](#), [5230](#), [5235](#), [5241](#),
[5466](#), [5481](#), [5482](#), [5484](#), [5490](#), [5523](#)
- \seq_item:Nn [102](#), [5394](#), [5394](#), [5417](#)
- \seq_item_aux:nnn [5394](#), [5396](#), [5410](#), [5415](#)
- \seq_length:c [5384](#)
- \seq_length:N [102](#), [5384](#), [5384](#), [5393](#), [5401](#)
- \seq_length_aux:n [5384](#), [5389](#), [5392](#)
- \seq_map_break [99](#)
- \seq_map_break: [5199](#), [5199](#), [5212](#), [9706](#)
- \seq_map_break:n [100](#), [5199](#), [5200](#)
- \seq_map_function:cN [5209](#)
- \seq_map_function:NN [4](#),
[99](#), [5209](#), [5209](#), [5221](#), [5304](#), [5389](#), [5418](#)
- \seq_map_function_aux:NNn
. [5209](#), [5211](#), [5215](#), [5219](#)
- \seq_map_inline:cn [5244](#)
- \seq_map_inline:Nn
[99](#), [5057](#), [5244](#), [5244](#), [5250](#), [9700](#), [9750](#)
- \seq_map_variable:ccn [5251](#)
- \seq_map_variable:cNn [5251](#)
- \seq_map_variable:Ncn [5251](#)
- \seq_map_variable:NNn
. [99](#), [5251](#), [5251](#), [5261](#), [5262](#)
- \seq_mapthread_function:ccN [5420](#)
- \seq_mapthread_function:cNN [5420](#)
- \seq_mapthread_function:NcN [5420](#)
- \seq_mapthread_function:NNN
. [102](#), [5420](#), [5420](#), [5444](#), [5445](#)
- \seq_mapthread_function_aux:NN
. [5420](#), [5422](#), [5429](#)
- \seq_mapthread_function_aux:Nnnwnn
. [5420](#), [5431](#), [5437](#), [5442](#)
- \seq_new:c [5009](#), [5010](#)
- \seq_new:N [4](#), [95](#), [5009](#), [5009](#), [5049](#), [8684](#),
[9661](#), [9662](#), [9671](#), [9673](#), [9676](#), [13157](#)
- \seq_pop:cN [5283](#), [5286](#)
- \seq_pop:NN [100](#), [5283](#), [5285](#)
- \seq_pop_item_def [103](#)
- \seq_pop_item_def: [5088](#), [5159](#),
[5191](#), [5222](#), [5238](#), [5248](#), [5259](#), [5971](#)
- \seq_pop_left:cN [5128](#), [5286](#), [5344](#)
- \seq_pop_left:NN
[97](#), [5128](#), [5128](#), [5143](#), [5285](#), [5344](#), [5344](#)
- \seq_pop_left:NnF [5373](#)
- \seq_pop_left:NNT [5372](#)
- \seq_pop_left:NNTF [101](#), [5374](#)
- \seq_pop_left_aux:NNN
. [5128](#), [5129](#), [5131](#), [5132](#)
- \seq_pop_left_aux:NnnNNN
. [5128](#), [5135](#), [5138](#), [5347](#), [5354](#)
- \seq_pop_right:cN [5169](#), [5344](#)
- \seq_pop_right:NN
. [97](#), [5169](#), [5169](#), [5195](#), [5344](#), [5358](#)
- \seq_pop_right:NnF [5379](#)
- \seq_pop_right:NNT [5378](#)
- \seq_pop_right:NNTF [102](#), [5380](#)
- \seq_pop_right_aux:NNN
. [5169](#), [5170](#), [5172](#), [5173](#)
- \seq_pop_right_aux_ii:NNN
. [5169](#), [5176](#), [5179](#), [5361](#), [5368](#)
- \seq_push:cn [5263](#), [5268](#)
- \seq_push:co [5263](#), [5271](#)
- \seq_push:cV [5263](#), [5269](#)
- \seq_push:cv [5270](#)
- \seq_push:cx [5263](#), [5272](#)
- \seq_push:Nn [101](#), [5263](#), [5263](#)
- \seq_push:No [5263](#), [5266](#)
- \seq_push:Nv [5263](#), [5264](#)
- \seq_push:Nv [5263](#), [5265](#)
- \seq_push:Nx [5263](#), [5267](#)
- \seq_push_item_def:n [103](#), [5072](#),
[5153](#), [5181](#), [5222](#), [5222](#), [5246](#), [5963](#)
- \seq_push_item_def:x [5222](#), [5227](#), [5253](#)
- \seq_push_item_def_aux:
. [5222](#), [5224](#), [5229](#), [5232](#)
- \seq_put_left:cn [5033](#), [5268](#)
- \seq_put_left:co [5033](#), [5271](#)
- \seq_put_left:cV [5033](#), [5269](#)

<code>\seq_put_left:cv</code>	5033 , 5270	<code>\seq_set_split_aux_i:w</code>	
<code>\seq_put_left:cx</code>	5033 , 5272	5492 , 5503 , 5510 , 5516
<code>\seq_put_left:Nn</code>		<code>\seq_set_split_aux_ii:w</code>	5492 , 5518 , 5522
.....	96 , 5033 , 5033 , 5037 , 5038 , 5263	<code>\seq_show:c</code>	5290 , 5530
<code>\seq_put_left:No</code>	5033 , 5266	<code>\seq_show:N</code> ...	101 , 5290 , 5290 , 5315 , 5529
<code>\seq_put_left:Nv</code>	5033 , 5264	<code>\seq_show_aux:n</code>	5290 , 5304 , 5309
<code>\seq_put_left:Nv</code>	5033 , 5265	<code>\seq_show_aux:w</code>	5290 , 5306 , 5314
<code>\seq_put_left:Nx</code>	5033 , 5267	<code>\seq_tmp:w</code>	5473 , 5481 , 5484
<code>\seq_put_right:cn</code>	5033	<code>\seq_top:cN</code>	5524 , 5526
<code>\seq_put_right:co</code>	5033	<code>\seq_top:NN</code>	5524 , 5525
<code>\seq_put_right:cV</code>	5033	<code>\seq_use:c</code>	5418
<code>\seq_put_right:cv</code>	5033	<code>\seq_use:N</code>	102 , 5418 , 5418 , 5419
<code>\seq_put_right:cx</code>	5033	<code>\seq_wrap_item:n</code>	
<code>\seq_put_right:Nn</code>	96 , 5033 , 5035 , 5039 , 5040 , 5060 , 8700 , 9742	..	5446 , 5449 , 5454 , 5459 , 5464 , 5466
<code>\seq_put_right:No</code>	5033	<code>\set@color</code>	7916
<code>\seq_put_right:Nv</code>	5033	<code>\setbox</code>	612
<code>\seq_put_right:Nv</code>	5033	<code>\setlanguage</code>	370
<code>\seq_put_right:Nx</code>	5033	<code>\sfcode</code>	667
<code>\seq_remove_all:cn</code>	5066	<code>\sffamily</code>	7677
<code>\seq_remove_all:Nn</code>		<code>\shipout</code>	577
.....	98 , 5066 , 5066 , 5090 , 9745	<code>\show</code>	420
<code>\seq_remove_all_aux:NNn</code>		<code>\showbox</code>	422
.....	5066 , 5067 , 5069 , 5070	<code>\showboxbreadth</code>	436
<code>\seq_remove_duplicates:c</code>	5050	<code>\showboxdepth</code>	437
<code>\seq_remove_duplicates:N</code>		<code>\showgroups</code>	697
.....	98 , 5050 , 5050 , 5064 , 9748	<code>\showifs</code>	698
<code>\seq_remove_duplicates_aux:NN</code>		<code>\showlists</code>	423
.....	5050 , 5051 , 5053 , 5054	<code>\showthe</code>	421
<code>\seq_reverse_aux:NN</code>	5476 , 5478 , 5479	<code>\showtokens</code>	685
<code>\seq_reverse_aux_item:w</code>	5482 , 5486	<code>\skewchar</code>	634
<code>\seq_set_eq:cc</code>	5019 , 5022	<code>\skip</code>	658
<code>\seq_set_eq:cN</code>	5019 , 5021	<code>\skip_add:cn</code>	4087
<code>\seq_set_eq:Nc</code>	5019 , 5020	<code>\skip_add:Nn</code> ...	75 , 4087 , 4087 , 4089 , 4090
<code>\seq_set_eq:NN</code>		<code>\skip_eval:n</code>	77 , 4100 , 4117 , 4117
.....	95 , 5019 , 5019 , 5051 , 9694 , 9711	<code>\skip_gadd:cn</code>	4087
<code>\seq_set_from_clist:cc</code>	5446	<code>\skip_gadd:Nn</code>	75 , 4087 , 4089 , 4091
<code>\seq_set_from_clist:cN</code>	5446	<code>\skip_gset:cn</code>	4076
<code>\seq_set_from_clist:cn</code>	5446	<code>\skip_gset:Nn</code>	76 , 4076 , 4078 , 4080
<code>\seq_set_from_clist:Nc</code>	5446	<code>\skip_gset_eq:cc</code>	4081
<code>\seq_set_from_clist:NN</code>		<code>\skip_gset_eq:cN</code>	4081
.....	102 , 5446 , 5446 , 5467 , 5468 , 9695 , 9756	<code>\skip_gset_eq:Nc</code>	4081
<code>\seq_set_from_clist:Nn</code>	5446 , 5451 , 5469	<code>\skip_gset_eq:NN</code> ...	76 , 4081 , 4084 – 4086
<code>\seq_set_reverse:N</code>	103 , 5473 , 5475	<code>\skip_gsub:cn</code>	4087
<code>\seq_set_split:Nnn</code>	103 , 5492 , 5492	<code>\skip_gsub:Nn</code>	76 , 4087 , 4094 , 4096
<code>\seq_set_split_aux:NNnn</code>		<code>\skip_gzero:c</code>	4072
.....	5492 , 5493 , 5495 , 5496	<code>\skip_gzero:N</code>	75 , 4072 , 4073 , 4075
<code>\seq_set_split_aux_end:</code>		<code>\skip_horizontal:c</code>	4121
..	5492 , 5505 , 5509 , 5516 , 5520 , 5522	<code>\skip_horizontal:N</code>	
		79 , 4121 , 4121 , 4123 , 4127
		<code>\skip_horizontal:n</code>	4121 , 4122

- \skip_if_eq:nn 4097, 4097
 - \skip_if_eq:nnTF 76
 - \skip_if_infinite_glue:n 4107, 4107
 - \skip_if_infinite_glue:nTF 76, 4180
 - \skip_new:c 4064
 - \skip_new:N 75, 4064, 4065, 4071, 4133–4137, 10151
 - \skip_set:cn 4076
 - \skip_set:Nn ... 75, 4076, 4076, 4078, 4079
 - \skip_set_eq:cc 4081
 - \skip_set_eq:cN 4081
 - \skip_set_eq:Nc 4081
 - \skip_set_eq:NN 76, 4081, 4081–4083
 - \skip_show:c 4129
 - \skip_show:N 77, 4129, 4129, 4130
 - \skip_split_finite_else_action:nnNN
..... 80, 4178, 4178
 - \skip_sub:cn 4087
 - \skip_sub:Nn ... 76, 4087, 4092, 4094, 4095
 - \skip_use:c 4119
 - \skip_use:N 77, 4118, 4119, 4119, 4120
 - \skip_vertical:c 4121
 - \skip_vertical:N 79, 4121, 4124, 4126, 4128
 - \skip_vertical:n 4121, 4125
 - \skip_zero:c 4072
 - \skip_zero:N 75, 4072, 4072–4074
 - \skipdef 357
 - \space 49, 205
 - \spacefactor 576
 - \spaceskip 571
 - \span 384
 - \special 646
 - \splitbotmark 455
 - \splitbotmarks 681
 - \splitdiscards 728
 - \splitfirstmark 454
 - \splitfirstmarks 680
 - \splitmaxdepth 624
 - \splittopskip 625
 - \str_head:n 91, 4651, 4651, 4672, 4715
 - \str_head_aux:w 4651, 4653, 4657
 - \str_if_eq:nn 1462, 1462
 - \str_if_eq:nnF 1864, 1865
 - \str_if_eq:nnT 1862, 1863, 5074
 - \str_if_eq:nnTF 21, 1866, 1867, 2128, 3792, 3795, 9043
 - \str_if_eq:no 1860
 - \str_if_eq:nV 1860
 - \str_if_eq:on 1860
 - \str_if_eq:Vn 1860
 - \str_if_eq:VV 1860
 - \str_if_eq:xx 1462, 1468
 - \str_if_eq:xxTF .. 2140, 6141, 6282, 8329
 - \str_if_eq_p:nn 1860, 1861
 - \str_if_eq_return:on 4733, 4789, 4789, 4804, 4808, 4809
 - \str_length_loop:NNNNNNNN 8359, 8364, 8368, 8374
 - \str_length_skip_spaces:N 8289, 8359, 8359
 - \str_length_skip_spaces:n 8359, 8360, 8361
 - \str_tail:n 91, 4651, 4659
 - \str_tail_aux:w 4661, 4665
 - \strcmp 230
 - \string 223, 238, 252, 639
- T**
- \T 1822, 2665, 2699, 2798
 - \t 2702
 - \tabskip 385
 - \tempa 106, 108, 109, 118, 124
 - \tempb 107, 108
 - \tex_above:D 467
 - \tex_abovedisplayshortskip:D 480
 - \tex_abovedisplayskip:D 481
 - \tex_abovewithdelims:D 468
 - \tex_accent:D 518
 - \tex_adjdemerits:D 555
 - \tex_advance:D 362, 3308, 3310, 3320, 3322, 3941, 3946, 4088, 4093, 4163, 4168, 9939, 9950, 9956, 9966, 9974, 10002, 10013, 10022, 10027, 10038, 10047, 10079, 10121, 10533, 10569, 10595, 10603, 10609, 10633, 10646, 10671, 10756, 10757, 10771, 10772, 10897, 10905, 10914, 11014, 11045–11047, 11049, 11050, 11082, 11083, 11087, 11088, 11097, 11098, 11101, 11102, 11108, 11231, 11232, 11244, 11256, 11266, 11271, 11299, 11304, 11315, 11333, 11342, 11390, 11391, 11394, 11395, 11457, 11469, 11699, 11700, 11734, 11739, 11742, 11743, 11747, 11748, 11753, 11754, 11758, 11759, 11762, 11763, 11766, 11767, 12116, 12136, 12194, 12198, 12220, 12235, 12236, 12240, 12241, 12246, 12247, 12251, 12252, 12255, 12256, 12259, 12260, 12350, 12354, 12402, 12425, 12454, 12484, 12492, 12495, 12542, 12545, 12546,

12548, 12564, 12584, 12588, 12601, 12602, 12605, 12606, 12611, 12612	\tex_displaywidowpenalty:D 484
\tex_afterassignment:D	\tex_displaywidth:D 486
372, 2868, 2954, 9889	\tex_divide:D 363, 9967, 10014, 10039, 10608, 10876, 11067, 11132, 11176, 11221, 11224, 11226, 11228, 11292, 11334, 12447, 12497–12499
\tex_aftergroup:D 373, 814	\tex_doublehyphendemerits:D 553
\tex_atop:D 469	\tex_dp:D 664, 6364
\tex_atopwithdelims:D 470	\tex_dump:D 647
\tex_badness:D 617	\tex_edef:D 351, 823
\tex_baselineskip:D 545	\tex_else:D 404, 785
\tex_batchmode:D 438	\tex_emergencystretch:D 568
\tex_begingroup:D 376, 810	\tex_end:D 442, 763, 1177, 8627, 8769
\tex_belowdisplayshortskip:D 482	\tex_endcsname:D 444, 806
\tex_belowdisplayskip:D 483	\tex_endgroup:D 377, 761, 811
\tex_binoppenalty:D 506	\tex_endinput:D 416, 8638
\tex_botmark:D 453	\tex_endlinechar:D
\tex_box:D 661, 6358, 6378	307, 308, 322, 458, 4321, 4348, 4361
\tex_boxmaxdepth:D 623	\tex_eqno:D 478
\tex_brokenpenalty:D 580	\tex_errhelp:D 424, 8545
\tex_catcode:D	\tex_errmessage:D 418, 1169, 8565
665, 1068, 1819–1825, 2265, 2279, 2475, 2477, 2479, 3095, 4305, 4306	\tex_errorcontextlines:D 425, 8583
\tex_char:D 519	\tex_errorstopmode:D 439
\tex_chardef:D 354, 1055, 1056, 1157–1161, 1898, 1900, 2794, 7963, 7966, 7977, 8109, 8122	\tex_escapechar:D 457, 8214, 8250, 8256
\tex_cleaders:D 537	\tex_everycr:D 386
\tex_closein:D 413, 8107, 8120	\tex_everydisplay:D 487, 764
\tex_closeout:D 408	\tex_everyhbox:D 626
\tex_clubpenalty:D 548	\tex_everyjob:D
\tex_copy:D 605, 6352, 6379	655, 4772, 4774, 9652, 9654, 9664, 9666
\tex_count:D 656	\tex_everymath:D 511, 765
\tex_countdef:D 355, 1154, 2726	\tex_everypar:D 574
\tex_cr:D 380	\tex_everyvbox:D 627
\tex_crcr:D 381	\tex_exhyphenpenalty:D 550
\tex_csname:D 443, 805	\tex_expandafter:D 374, 800
\tex_day:D 651	\tex_fam:D 366
\tex_deadcycles:D 585	\tex_fi:D 405, 786
\tex_def:D 350, 818–822	\tex_finalhyphendemerits:D 554
\tex_defaultthyphenchar:D 635	\tex_firstmark:D 452
\tex_defaultskewchar:D 636	\tex_floatingpenalty:D 599
\tex_delcode:D 666	\tex_font:D 365
\tex_delimiter:D 460	\tex_fontdimen:D 632
\tex_delimiterfactor:D 509	\tex_fontname:D 456
\tex_delimitershortfall:D 508	\tex_futurelet:D 361, 2862, 2864
\tex_dimen:D 657	\tex_gdef:D 352, 836
\tex_dimendef:D 356, 2748	\tex_global:D 336, 341, 343, 367, 815, 815, 1267, 1272, 1900, 2864, 3287, 3298, 3304, 3312, 3314, 3324, 3326, 3333, 3914, 3919, 3925, 3942, 3947, 4073, 4078, 4084, 4089, 4094, 4148, 4153, 4159, 4164,
\tex_discretionary:D 520	
\tex_displayindent:D 485	
\tex_displaylimits:D 495	
\tex_displaystyle:D 473	

4169, 4957, 6354, 6360, 6415, 6435, 6441, 6447, 6475, 6481, 6487, 6493, 7977, 8109, 8122, 8396, 8400, 13166	\tex_italic_correction:D 768
\tex_globaldefs:D 371	\tex_italiccor:D 347
\tex_halign:D 378	\tex_jobname:D 654, 4782, 9655
\tex_hangafter:D 556	\tex_kern:D 532, 6564, 6794, 7251, 7256, 7322, 7323, 7430, 7431, 7834, 7835
\tex_hangindent:D 557	\tex_language:D 449
\tex_hbadness:D 618	\tex_lastbox:D 606, 6411
\tex_hbox:D	\tex_lastkern:D 539
613, 6433, 6434, 6439, 6445, 6459, 6460	\tex_lastpenalty:D 645
\tex_hfil:D 521	\tex_lastskip:D 540
\tex_hfill:D 523	\tex_lccode:D
\tex_hfilneg:D 522	668, 1067, 1817, 1818, 2264, 2278, 2551, 2553, 2555, 3097, 4303, 4304
\tex_hfuzz:D 620	\tex_leaders:D 536
\tex_hoffset:D 595	\tex_left:D 504
\tex_holdinginserts:D 598	\tex_lefthyphenmin:D 560
\tex_hruler:D 534	\tex_leftskip:D 562
\tex_hsize:D 559, 6919, 6962	\tex_leqno:D 479
\tex_hskip:D 524, 4121	\tex_let:D 337, 341, 343, 349, 763–773, 782–817, 822, 823, 836, 837, 1151, 1260, 1511
\tex_hss:D 525, 6462, 6464, 6796	\tex_limits:D 496
\tex_ht:D 663, 6363	\tex_linepenalty:D 552
\tex_hyphen:D 348, 766	\tex_lineskip:D 546
\tex_hyphenation:D 649	\tex_lineskiplimit:D 547
\tex_hyphenchar:D 633	\tex_long:D 368, 815, 816, 818, 825, 827, 833, 835, 839, 841, 847, 849
\tex_hyphenpenalty:D 551	\tex_looseness:D 564
\tex_if:D 387, 788, 791	\tex_lower:D 601, 6389
\tex_ifcase:D 388, 3199	\tex_lowercase:D 640, 1069, 1826, 4307, 4368
\tex_ifcat:D 389, 792	\tex_mag:D 448
\tex_ifdim:D 392, 3902	\tex_mark:D 450
\tex_ifeof:D 393, 7925	\tex_mathaccent:D 461
\tex_iffalse:D 398, 783	\tex_mathbin:D 491
\tex_ifhbox:D 394, 6390	\tex_mathchar:D 462
\tex_ifhmode:D 400, 795	\tex_mathchardef:D 359, 1162, 3287, 13166
\tex_ifinner:D 403, 797	\tex_mathchoice:D 459
\tex_ifmmode:D 401, 794	\tex_mathclose:D 492
\tex_ifnum:D 390, 812, 3197	\tex_mathcode:D 670, 2545, 2547, 2549, 3096
\tex_ifodd:D 391, 789, 790, 1194, 3198	\tex_mathinner:D 493
\tex_iftrue:D 399, 782	\tex_mathop:D 494
\tex_ifvbox:D 395, 6391	\tex_mathopen:D 498
\tex_ifvmode:D 402, 796	\tex_mathord:D 499
\tex_ifvoid:D 396, 6392	\tex_mathpunct:D 500
\tex_ifx:D 397, 793	\tex_mathrel:D 501
\tex_ignorespaces:D 445	\tex_mathsurround:D 512
\tex_immediate:D 407, 1164, 1166, 8005, 8183	\tex_maxdeadcycles:D 582
\tex_indent:D 541	\tex_maxdepth:D 583
\tex_input:D 415, 767, 9735	\tex_meaning:D 642, 803, 807
\tex_inputlineno:D 417, 1184, 1812, 8484	\tex_medmuskip:D 513
\tex_insert:D 597	
\tex_insertpenalties:D 600	
\tex_interlinepenalty:D 579	

<code>\tex_message:D</code>	419	<code>\tex_predisplaysize:D</code>	488
<code>\tex_mkern:D</code>	466	<code>\tex_pretolerance:D</code>	569
<code>\tex_month:D</code>	652	<code>\tex_prevdepth:D</code>	616
<code>\tex_moveleft:D</code>	602, 6383	<code>\tex_prevgraf:D</code>	575
<code>\tex_moveright:D</code>	603, 6385	<code>\tex_protected:D</code> ..	815, 817, 824, 826, 828–831, 833, 835, 843, 845, 847, 849
<code>\tex_mskip:D</code>	463	<code>\tex_radical:D</code>	464
<code>\tex_multiply:D</code>	364, 10806, 11013, 11227, 11243, 12221, 12810	<code>\tex_raise:D</code>	604, 6387
<code>\tex_muskip:D</code>	660	<code>\tex_read:D</code>	411, 8394, 8396
<code>\tex_muskipdef:D</code>	358	<code>\tex_relax:D</code>	446, 809, 3196, 3904
<code>\tex_newlinechar:D</code> .	414, 4322, 4349, 4362	<code>\tex_relpenny:D</code>	507
<code>\tex_noalign:D</code>	382	<code>\tex_right:D</code>	505
<code>\tex_noboundary:D</code>	517	<code>\tex_righthyphenmin:D</code>	561
<code>\tex_noexpand:D</code>	375, 801	<code>\tex_rightskip:D</code>	563
<code>\tex_noindent:D</code>	543	<code>\tex_romannumeral:D</code> ..	638, 813, 1532, 1544, 1550, 1592, 1596, 1601, 1607, 1613, 1619, 1631, 1636, 1638, 1645, 1699, 1706, 1711, 1714, 1716, 1719, 1724, 1730, 1742, 1752, 1756, 1761, 2097, 2110, 2123, 2135, 2146, 4811, 4863, 4882, 4922, 4924, 4944, 8963
<code>\tex_nolimits:D</code>	497	<code>\tex_scriptfont:D</code>	630
<code>\tex_nonscript:D</code>	477	<code>\tex_scriptscriptfont:D</code>	631
<code>\tex_nonstopmode:D</code>	440	<code>\tex_scriptscriptstyle:D</code>	476
<code>\tex_nulldelimiterspace:D</code>	510	<code>\tex_scriptspace:D</code>	516
<code>\tex_nullfont:D</code>	628, 2821	<code>\tex_scriptstyle:D</code>	475
<code>\tex_number:D</code>	637, 3194	<code>\tex_scrollmode:D</code>	441
<code>\tex_omit:D</code>	383	<code>\tex_setbox:D</code>	612, 6352, 6358, 6413, 6434, 6439, 6445, 6474, 6479, 6485, 6491, 6509
<code>\tex_openin:D</code>	409, 7992	<code>\tex_setlanguage:D</code>	370
<code>\tex_openout:D</code>	410, 8005	<code>\tex_sfcode:D</code> .	667, 2563, 2565, 2567, 3099
<code>\tex_or:D</code>	406, 784	<code>\tex_shipout:D</code>	577
<code>\tex_outer:D</code>	369	<code>\tex_show:D</code>	420, 808
<code>\tex_output:D</code>	584	<code>\tex_showbox:D</code>	422, 6431
<code>\tex_outputpenalty:D</code>	594	<code>\tex_showboxbreadth:D</code>	436
<code>\tex_over:D</code>	471	<code>\tex_showboxdepth:D</code>	437
<code>\tex_overfullrule:D</code>	622	<code>\tex_showlists:D</code>	423
<code>\tex_overline:D</code>	502	<code>\tex_showthe:D</code>	421, 1420, 2479, 2549, 2555, 2561, 2567, 3104, 3107, 3110, 3113, 3116
<code>\tex_overwithdelims:D</code>	472	<code>\tex_skewchar:D</code>	634
<code>\tex_pagedepth:D</code>	586	<code>\tex_skip:D</code>	658
<code>\tex_pagefilllstretch:D</code>	590	<code>\tex_skipdef:D</code>	357, 2737
<code>\tex_pagefillstretch:D</code>	589	<code>\tex_space:D</code>	346
<code>\tex_pagefilstretch:D</code>	588	<code>\tex_spacefactor:D</code>	576
<code>\tex_pagegoal:D</code>	592	<code>\tex_spaceskip:D</code>	571
<code>\tex_pageshrink:D</code>	591	<code>\tex_span:D</code>	384
<code>\tex_pagestretch:D</code>	587	<code>\tex_special:D</code>	646
<code>\tex_pagetotal:D</code>	593	<code>\tex_splitbotmark:D</code>	455
<code>\tex_par:D</code>	542, 7908		
<code>\tex_parfillskip:D</code>	573		
<code>\tex_parindent:D</code>	566		
<code>\tex_parshape:D</code>	558		
<code>\tex_parskip:D</code>	565		
<code>\tex_patterns:D</code>	648		
<code>\tex_pausing:D</code>	435		
<code>\tex_penalty:D</code>	643		
<code>\tex_postdisplaypenalty:D</code>	490		
<code>\tex_predisplaysize:D</code>	489		

<code>\tex_splitfirstmark:D</code>	454	<code>\tex_voffset:D</code>	596
<code>\tex_splitmaxdepth:D</code>	624	<code>\tex_vrule:D</code>	535, 7685, 7740
<code>\tex_splittopskip:D</code>	625	<code>\tex_vsize:D</code>	578
<code>\tex_string:D</code>	639, 804	<code>\tex_vskip:D</code>	529, 4124
<code>\tex_tabskip:D</code>	385	<code>\tex_vsplit:D</code>	607, 6509
<code>\tex_textfont:D</code>	629	<code>\tex_vss:D</code>	530
<code>\tex_textstyle:D</code>	474	<code>\tex_vtop:D</code>	615, 6470, 6479
<code>\tex_the:D</code>		<code>\tex_wd:D</code>	662, 6365
. 308, 447, 1184, 1562, 1566, 1812,		<code>\tex_widowpenalty:D</code>	549
2477, 2547, 2553, 2559, 2565, 3102,		<code>\tex_write:D</code>	412, 1164, 1166, 8178
3105, 3108, 3111, 3114, 3336, 4046,		<code>\tex_xdef:D</code>	353, 837
4119, 4174, 4774, 9654, 9666, 13190		<code>\tex_xleaders:D</code>	538
<code>\tex_thickmuskip:D</code>	515	<code>\tex_xspaceskip:D</code>	572
<code>\tex_thinmuskip:D</code>	514	<code>\tex_year:D</code>	653
<code>\tex_time:D</code>	650	<code>\textasteriskcentered</code>	3884, 3890
<code>\tex_toks:D</code>	659	<code>\textbardbl</code>	3889
<code>\tex_toksdef:D</code>	360, 2759	<code>\textdagger</code>	3885, 3891
<code>\tex_tolerance:D</code>	570	<code>\textdaggerdbl</code>	3886, 3892
<code>\tex_topmark:D</code>	451	<code>\textfont</code>	629
<code>\tex_topskip:D</code>	581	<code>\textparagraph</code>	3888
<code>\tex_tracingcommands:D</code>	426	<code>\textsection</code>	3887
<code>\tex_tracinglostchars:D</code>	427	<code>\textstyle</code>	474
<code>\tex_tracingmacros:D</code>	428	<code>\texttt</code>	8510
<code>\tex_tracingonline:D</code>	429	<code>\TeXETstate</code>	729
<code>\tex_tracingoutput:D</code>	430	<code>\the</code>	70–79, 447
<code>\tex_tracingpages:D</code>	431	<code>\thickmuskip</code>	515
<code>\tex_tracingparagraphs:D</code>	432	<code>\thinmuskip</code>	514
<code>\tex_tracingrestores:D</code>	433	<code>\time</code>	650
<code>\tex_tracingstats:D</code>	434	<code>\tiny</code>	7677
<code>\tex_uccode:D</code> . 669, 2557, 2559, 2561, 3098		<code>\tl_act:NNNnn</code>	4811, 4811
<code>\tex_uchyph:D</code>	567	<code>\tl_act_aux:NNNnn</code>	4811,
<code>\tex_undefined:D</code>	336, 343	4811, 4812, 4864, 4883, 4899, 4927	
<code>\tex_underline:D</code>	503, 769	<code>\tl_act_case_aux:nn</code> 4922, 4924, 4925, 4944	
<code>\tex_unhbox:D</code>	608, 6466	<code>\tl_act_case_group:nn</code> . . 4921, 4929, 4941	
<code>\tex_unhcopy:D</code>	609, 6465	<code>\tl_act_case_normal:nN</code> . 4921, 4928, 4933	
<code>\tex_unkern:D</code>	533	<code>\tl_act_case_space:n</code> . . 4921, 4930, 4932	
<code>\tex_unpenalty:D</code>	644	<code>\tl_act_end:w</code>	4811
<code>\tex_unskip:D</code>	531	<code>\tl_act_end:wn</code>	4832, 4838
<code>\tex_unvbox:D</code>	610, 6505	<code>\tl_act_group:nwnNNN</code> . . 4811, 4824, 4839	
<code>\tex_unvcopy:D</code>	611, 6504	<code>\tl_act_group_recurse:Nnn</code> 4811, 4856, 4876	
<code>\tex_uppercase:D</code>	641, 4369	<code>\tl_act_length_group:nn</code> 4895, 4901, 4909	
<code>\tex_vadjust:D</code>	544	<code>\tl_act_length_normal:nN</code> 4895, 4900, 4907	
<code>\tex_valign:D</code>	379	<code>\tl_act_length_space:n</code> . 4895, 4902, 4908	
<code>\tex_vbadness:D</code>	619	<code>\tl_act_loop:w</code>	
<code>\tex_vbox:D</code> 614, 6469, 6472–6474, 6485, 6491		. . 4811, 4814, 4818, 4835, 4842, 4849	
<code>\tex_vcenter:D</code>	465	<code>\tl_act_normal:NwnNNN</code> . . 4811, 4821, 4829	
<code>\tex_vfil:D</code>	526	<code>\tl_act_output:n</code>	
<code>\tex_vfill:D</code>	528 4811, 4852, 4932, 4935, 4943	
<code>\tex_vfilneg:D</code>	527	<code>\tl_act_result:n</code> . . 4816, 4838, 4852–4855	
<code>\tex_vfuzz:D</code>	621	<code>\tl_act_reverse_group:nn</code> 4861, 4866, 4874	

- \tl_act_reverse_group_preserve:nn 4885, 4889
- \tl_act_reverse_normal:nN 4861, 4865, 4872, 4884
- \tl_act_reverse_output:n 4811, 4854, 4871, 4873, 4877, 4890
- \tl_act_reverse_space:n 4861, 4867, 4870, 4886
- \tl_act_space:wwnNNN . . . 4811, 4825, 4846
- \tl_clear:c 4216, 5012, 5544
- \tl_clear:N 81, 4216, 4216, 4220, 4223, 4324, 5011, 5543, 8245, 8247, 8248, 8338, 9017, 9021, 9690, 10594, 10872, 10890, 11125, 11149, 11169, 11199, 11213
- \tl_clear_new:c 4222, 5016, 5548, 9289, 9291
- \tl_clear_new:N 82, 4222, 4222, 4226, 5015, 5547
- \tl_const:cn 4204, 11926–11965, 12272–12283
- \tl_const:cx 4204, 8219, 12362
- \tl_const:Nn 81, 2384, 4204, 4204, 4214, 4310, 4783, 4784, 4911, 4916, 6003, 7930, 8211, 8419, 8420, 8450, 8455, 8457, 8459, 8461, 8463, 8468, 8469, 8476, 8494, 8943, 8945, 9108–9112, 9784–9788
- \tl_const:Nx . 4204, 4209, 4215, 4782, 8215
- \tl_elt_count:c 4981, 4986
- \tl_elt_count:N 4981, 4985
- \tl_elt_count:n 4981, 4982
- \tl_elt_count:o 4981, 4984
- \tl_elt_count:V 4981, 4983
- \tl_expandable_lowercase:n 94, 4921, 4923
- \tl_expandable_uppercase:n 94, 4921, 4921
- \tl_gclear:c 4216, 5014, 5546
- \tl_gclear:N 82, 4216, 4218, 4221, 4225, 5013, 5545
- \tl_gclear_new:c 4222, 5018, 5550
- \tl_gclear_new:N 82, 4222, 4224, 4227, 5017, 5549
- \tl_gput_left:cn 4254
- \tl_gput_left:co 4254
- \tl_gput_left:cV 4254
- \tl_gput_left:cx 4254
- \tl_gput_left:Nn 83, 4254, 4262, 4274, 5042
- \tl_gput_left:No 4254, 4266, 4276
- \tl_gput_left:NV 4254, 4264, 4275
- \tl_gput_left:Nx . 4254, 4268, 4277, 5625
- \tl_gput_right:cn 4278
- \tl_gput_right:co 4278
- \tl_gput_right:cV 4278
- \tl_gput_right:cx 4278
- \tl_gput_right:Nn 83, 4278, 4286, 4298, 5044
- \tl_gput_right:No 4278, 4290, 4300
- \tl_gput_right:NV 4278, 4288, 4299
- \tl_gput_right:Nx 4278, 4292, 4301, 5643, 6108
- \tl_gremove_all:cn 4425, 4979
- \tl_gremove_all:Nn 84, 4425, 4427, 4430, 4978
- \tl_gremove_all_in:cn 4971, 4979
- \tl_gremove_all_in:Nn 4971, 4978
- \tl_gremove_in:cn 4971, 4975
- \tl_gremove_in:Nn 4971, 4974
- \tl_gremove_once:cn 4419, 4975
- \tl_gremove_once:Nn 84, 4419, 4421, 4424, 4974
- \tl_greplace_all:cnn 4371, 4969
- \tl_greplace_all:Nnn 83, 4371, 4377, 4382, 4428, 4968
- \tl_greplace_all_in:cnn 4961, 4969
- \tl_greplace_all_in:Nnn 4961, 4968
- \tl_greplace_in:cnn 4961, 4965
- \tl_greplace_in:Nnn 4961, 4964
- \tl_greplace_once:cnn 4371, 4965
- \tl_greplace_once:Nnn 83, 4371, 4373, 4380, 4422, 4964
- \tl_gset:cf 4236
- \tl_gset:cn 4236
- \tl_gset:co 4236
- \tl_gset:cx 4236, 11543, 11629, 11910
- \tl_gset:Nc 4955, 4956
- \tl_gset:Nf 4236, 5656
- \tl_gset:Nn 82, 4236, 4242, 4251, 4253, 4316, 4343, 4950, 5131, 5354, 6045, 6071, 6244, 6295, 9734, 10069, 10523, 10558, 10639, 10664, 10690, 10793, 10811, 10924, 11476, 11573, 11774, 11967, 12285, 12619
- \tl_gset:No 4236, 4244
- \tl_gset:NV 4236
- \tl_gset:Nv 4236
- \tl_gset:Nx . . . 4236, 4246, 4252, 4374, 4378, 4639, 5030, 5069, 5172, 5368, 5458, 5463, 5478, 5495, 5562, 5619, 5709, 5957, 6084, 9655, 10105, 12269
- \tl_gset_eq:cc 4228, 4235, 5026, 5558, 6023, 10159

<code>\tl_gset_eq:cN</code>	<code>\tl_if_empty:V</code>
. 4228 , 4233 , 5025 , 5557 , 6022 , 10157	<code>\tl_if_empty_p:N</code>
<code>\tl_gset_eq:Nc</code>	<code>\tl_if_empty_p:n</code>
. 4228 , 4234 , 5024 , 5556 , 6021 , 10158	<code>\tl_if_empty_return:o</code>
<code>\tl_gset_eq:NN</code> 4432 , 4465 , 4465 , 4475
4219 , 4228 , 4232 , 5023 , 5555 , 5625 ,	<code>\tl_if_eq:cc</code>
5643 , 6020 , 9659 , 10053 , 10065 , 10156 4476 , 5743 , 6314
<code>\tl_gset_rescan:cnn</code>	<code>\tl_if_eq:ccTF</code>
4313 9522
<code>\tl_gset_rescan:cno</code>	<code>\tl_if_eq:cN</code>
4313 4476 , 5742 , 6312
<code>\tl_gset_rescan:cnx</code>	<code>\tl_if_eq:Nc</code>
4340 4476 , 5741 , 6313
<code>\tl_gset_rescan:Nnn</code>	<code>\tl_if_eq:NN</code>
..... 85 , 4313 , 4315 , 4338 , 4339 4476 , 4476 , 5740 , 6311
<code>\tl_gset_rescan:Nno</code>	<code>\tl_if_eq:nn</code>
4313 4488 , 4488
<code>\tl_gset_rescan:Nnx</code>	<code>\tl_if_eq:NNF</code>
..... 4340 , 4342 , 4356 4487
<code>\tl_gtrim_spaces:c</code>	<code>\tl_if_eq:NNT</code>
4597 4486 , 5081 , 7751 , 7754
<code>\tl_gtrim_spaces:N</code>	<code>\tl_if_eq:NNTF</code>
..... 90 , 4597 , 4638 , 4641 86 , 2151 , 4485 , 8282 , 8736
<code>\tl_head:f</code>	<code>\tl_if_eq:nnTF</code>
..... 4642 86
<code>\tl_head:n</code>	<code>\tl_if_eq_p:NN</code>
..... 90 , 4642 , 4644 , 4649 , 4989 4484
<code>\tl_head:V</code>	<code>\tl_if_head_eq_catcode:nN</code>
..... 4642 4666 , 4682
<code>\tl_head:v</code>	<code>\tl_if_head_eq_catcode:nNTF</code>
..... 4642 91
<code>\tl_head:w</code>	<code>\tl_if_head_eq_charcode:fN</code>
..... 90 , 4642 , 4642 , 4645 , 4658 , 4666 , 4666
4671 , 4687 , 4707 , 4990 , 9863 , 9874	<code>\tl_if_head_eq_charcode:nN</code>
<code>\tl_head_i:n</code> 4681
..... 4988 , 4989	<code>\tl_if_head_eq_charcode:nNT</code>
<code>\tl_head_i:w</code> 4680
..... 4988 , 4990	<code>\tl_if_head_eq_charcode:nNTF</code>
<code>\tl_head_iii:f</code> 91 , 3688 , 3701 , 4679
..... 4988	<code>\tl_if_head_eq_charcode_p:nN</code>
<code>\tl_head_iii:n</code> 4678
..... 4988 , 4991 , 4992	<code>\tl_if_head_eq_meaning:nN</code>
<code>\tl_head_iii:w</code> 4666 , 4698
..... 4988 , 4991 , 4993	<code>\tl_if_head_eq_meaning:nNTF</code>
<code>\tl_if_blank:n</code> 92 , 9059
..... 4431 , 4431	<code>\tl_if_head_eq_meaning_aux_normal:nN</code>
<code>\tl_if_blank:nF</code> 4701 , 4705
..... 3783 , 4435 , 4439 , 5892	<code>\tl_if_head_eq_meaning_aux_special:nN</code>
<code>\tl_if_blank:nT</code> 4702 , 4713
..... 4434 , 4438	<code>\tl_if_head_group:n</code>
<code>\tl_if_blank:nTF</code> 4734 , 4734
..... 85 , 4436 , 4440 , 5936	<code>\tl_if_head_group:nTF</code>
<code>\tl_if_blank:o</code> 92 , 4689 , 4723 , 4823
..... 4431	<code>\tl_if_head_N_type:n</code>
<code>\tl_if_blank:oTF</code> 4732 , 4732
..... 9030	<code>\tl_if_head_N_type:nTF</code>
<code>\tl_if_blank:V</code> 92 , 4670 , 4686 , 4700 , 4807 , 4820
..... 4431	<code>\tl_if_head_space:n</code>
<code>\tl_if_blank_p:n</code> 4750 , 4750
..... 4433 , 4437	<code>\tl_if_head_space:nTF</code>
<code>\tl_if_blank_p_aux:NNw</code> 92
..... 4441 , 5094 , 5739	<code>\tl_if_head_space_aux:w</code>
<code>\tl_if_empty:c</code> 4750 , 4755 , 4762
..... 4441 , 4441 , 5092 , 5738	<code>\tl_if_in:cn</code>
<code>\tl_if_empty:n</code> 4503
..... 4453 , 4453	<code>\tl_if_in:Nn</code>
<code>\tl_if_empty:NF</code> 4503
..... 4451 , 5632 , 9725	<code>\tl_if_in:nn</code>
<code>\tl_if_empty:nF</code> 4509 , 4509
..... 2972 , 4464 , 5838	<code>\tl_if_in:NnF</code>
<code>\tl_if_empty:NT</code> 4504 , 4507
..... 4450	<code>\tl_if_in:nnF</code>
<code>\tl_if_empty:nT</code> 4504 , 4516
..... 4463	<code>\tl_if_in:NnT</code>
<code>\tl_if_empty:NTF</code> 4503 , 4506
..... 86 , 2716 , 2723 , 2734 , 2745 ,	<code>\tl_if_in:nnT</code>
2756 , 2767 , 2776 , 2783 , 2792 , 3825 , 4503 , 4515
4385 , 4462 , 5498 , 5607 , 8501 , 9183	<code>\tl_if_in:NnTF</code>
<code>\tl_if_empty:o</code> 86 , 4505 , 4508
..... 4465 , 4474	<code>\tl_if_in:nnTF</code>
<code>\tl_if_empty:oTF</code> 86 , 4505 , 4517 , 7337 , 9153 , 9160
..... 2813 , 4512 , 5756 , 5966	<code>\tl_if_in:no</code>
 4509

`\tl_if_in:on` 4509
`\tl_if_in:Vn` 4509
`\tl_if_single:N` 4799
`\tl_if_single:n` 4803, 4803
`\tl_if_single:NF` 4801
`\tl_if_single:nF` 4801
`\tl_if_single:NT` 4800
`\tl_if_single:nT` 4800
`\tl_if_single:NTF` 86, 4802
`\tl_if_single:nTF` 87, 4802
`\tl_if_single_p:N` 4799
`\tl_if_single_p:n` 4799
`\tl_if_single_token:n` 4805, 4805
`\tl_if_single_token:nTF` 87
`\tl_length:c` 4576, 4986
`\tl_length:N` ... 89, 4576, 4581, 4588, 4985
`\tl_length:n` ... 89, 4576, 4576, 4587, 4982
`\tl_length:o` 4576, 4984
`\tl_length:V` 4576, 4983
`\tl_length_aux:n` . 4576, 4579, 4584, 4586
`\tl_length_tokens:n` . 94, 4895, 4895, 4910
`\tl_map_break` 88
`\tl_map_break:` ... 4563, 4563, 8015, 8023
`\tl_map_function:cn` 4518
`\tl_map_function:NN`
 87, 4518, 4520, 4530, 4584, 7986, 7999
`\tl_map_function:nN` . 87, 4518, 4518, 4579
`\tl_map_function_aux:NN` 4518
`\tl_map_function_aux:Nn`
 .. 4519, 4522, 4525, 4528, 4536, 4546
`\tl_map_inline:cn` 4531
`\tl_map_inline:Nn` .. 87, 4531, 4541, 4551
`\tl_map_inline:nn` .. 87, 2705, 4531, 4531
`\tl_map_inline_aux:n` 4531
`\tl_map_variable:cNn` 4552
`\tl_map_variable:NNn` 87, 4552, 4554, 4562
`\tl_map_variable:nNn` 88, 4552, 4552, 4555
`\tl_map_variable_aux:NnN` 4552
`\tl_map_variable_aux:Nnn` 4553, 4556, 4560
`\tl_new:c` 4198,
 5010, 5542, 9271, 11542, 11628, 11909
`\tl_new:cn` 4946
`\tl_new:N` 81, 4198, 4198, 4203,
 4223, 4225, 4312, 4370, 4501, 4502,
 4785–4788, 4949, 5007–5009, 5289,
 5539, 5541, 6206, 6814, 6840, 6841,
 7672, 8202–8206, 8418, 8492, 8497,
 8685–8687, 9002–9005, 9114–9116,
 9118–9121, 9650, 9670, 9789, 9819,
 9826, 9829, 9832, 10052, 12268, 13218
`\tl_new:Nn` 4946, 4947, 4952, 4953
`\tl_new:Nx` 4946
`\tl_put_left:cn` 4254
`\tl_put_left:co` 4254
`\tl_put_left:cV` 4254
`\tl_put_left:cx` 4254
`\tl_put_left:Nn` 82, 4254, 4254, 4270, 5034
`\tl_put_left:No` 4254, 4258, 4272
`\tl_put_left:NV` 4254, 4256, 4271
`\tl_put_left:Nx` .. 4254, 4260, 4273, 5623
`\tl_put_right:cn` 4278
`\tl_put_right:co` 4278
`\tl_put_right:cV` 4278
`\tl_put_right:cx` 4278
`\tl_put_right:Nn`
 83, 4278, 4278, 4294, 5036, 9076
`\tl_put_right:No` . 4278, 4282, 4296, 8563
`\tl_put_right:NV` 4278, 4280, 4295
`\tl_put_right:Nx` ... 4278, 4284, 4297,
 5641, 6106, 8301, 8307, 8314, 8335,
 8344, 8355, 9045, 9067, 9080, 9089
`\tl_remove_all:cn` 4425, 4977
`\tl_remove_all:Nn` 84, 4425, 4425, 4429, 4976
`\tl_remove_all_in:cn` 4971, 4977
`\tl_remove_all_in:Nn` 4971, 4976
`\tl_remove_in:cn` 4971, 4973
`\tl_remove_in:Nn` 4971, 4972
`\tl_remove_once:cn` 4419, 4973
`\tl_remove_once:Nn`
 84, 4419, 4419, 4423, 4972
`\tl_replace_all:cnn` 4371, 4967
`\tl_replace_all:Nnn` .. 83, 4371, 4375,
 4381, 4426, 4966, 5507, 9019, 9020
`\tl_replace_all_aux:`
 4371, 4376, 4378, 4407, 4410
`\tl_replace_all_in:cnn` 4961, 4967
`\tl_replace_all_in:Nnn` 4961, 4966
`\tl_replace_aux:NNNnn`
 .. 4371, 4372, 4374, 4376, 4378, 4383
`\tl_replace_aux_ii:w` 4371, 4406, 4409, 4414
`\tl_replace_in:cnn` 4961, 4963
`\tl_replace_in:Nnn` 4961, 4962
`\tl_replace_once:cnn` 4371, 4963
`\tl_replace_once:Nnn`
 83, 4371, 4371, 4379, 4420, 4962
`\tl_replace_once_aux:`
 4371, 4372, 4374, 4412
`\tl_replace_once_aux_end:w`
 4371, 4415, 4417
`\tl_rescan:nn` 85, 4357, 4357

`\tl_rescan_aux:w`
 .. [4313](#), [4325](#), [4329](#), [4331](#), [4335](#), [4364](#)
`\tl_reverse:c` [4892](#)
`\tl_reverse:N` [89](#), [4892](#), [4892](#), [4894](#)
`\tl_reverse:n` [89](#), [4880](#), [4880](#), [4891](#)
`\tl_reverse:o` [4880](#), [4893](#)
`\tl_reverse:V` [4880](#)
`\tl_reverse_group_preserve:nn` ... [4880](#)
`\tl_reverse_items:n` [89](#), [4589](#), [4589](#)
`\tl_reverse_items_aux:nN` [4589](#)
`\tl_reverse_items_aux:nw` [4590](#), [4591](#), [4594](#)
`\tl_reverse_tokens:n` [93](#), [4861](#), [4861](#), [4878](#)
`\tl_set:cf` [4236](#)
`\tl_set:cn` [4236](#), [9290](#), [9294](#)
`\tl_set:co` [4236](#)
`\tl_set:cx` [4236](#), [9274](#)
`\tl_set:Nc` [4955](#), [4957](#), [4958](#)
`\tl_set:Nf` [4236](#), [5654](#)
`\tl_set:Nn` [82](#), [2239](#), [2874](#),
 [2895](#), [4236](#), [4236](#), [4248](#), [4250](#), [4314](#),
 [4333](#), [4341](#), [4491](#), [4492](#), [4558](#), [5077](#),
 [5086](#), [5100](#), [5103](#), [5126](#), [5129](#), [5141](#),
 [5156](#), [5187](#), [5255](#), [5347](#), [5501](#), [5651](#),
 [5663](#), [5819](#), [6043](#), [6056](#), [6059](#), [6065](#),
 [6066](#), [6072](#), [6076](#), [6169](#), [6238](#), [6249](#),
 [6821](#), [6825](#), [7007](#), [7338](#), [7339](#), [7674](#),
 [7677](#), [8281](#), [8530](#), [8717](#), [8718](#), [9018](#),
 [9128](#), [9165](#), [9232](#), [9258](#), [9326](#), [9450](#),
 [9463](#), [9507](#), [10068](#), [10520](#), [10555](#),
 [10638](#), [10663](#), [10689](#), [10792](#), [10810](#),
 [10923](#), [11475](#), [11572](#), [11773](#), [11966](#),
 [12284](#), [12618](#), [12676](#), [12688](#), [13203](#)
`\tl_set:No` [4236](#), [4238](#), [4893](#), [4959](#)
`\tl_set:Nv` [4236](#)
`\tl_set:Nv` [4236](#)
`\tl_set:Nx`
 [4236](#), [4240](#), [4249](#), [4351](#), [4372](#), [4376](#),
 [4637](#), [5028](#), [5067](#), [5170](#), [5303](#), [5361](#),
 [5448](#), [5453](#), [5476](#), [5493](#), [5512](#), [5560](#),
 [5617](#), [5628](#), [5707](#), [5825](#), [5857](#), [5955](#),
 [6083](#), [6220](#), [7859](#), [8137](#), [8157](#), [8262](#),
 [8321](#), [8350](#), [8534](#), [9062](#), [9073](#), [9088](#),
 [9126](#), [9152](#), [9159](#), [9162](#), [9448](#), [9458](#),
 [9480](#), [9481](#), [9685](#), [9705](#), [9852](#), [9865](#),
 [9876](#), [9968](#), [10015](#), [10040](#), [10103](#),
 [10623](#), [10626](#), [10672](#), [10920](#), [11284](#),
 [11293](#), [11316](#), [11335](#), [11488](#), [11585](#),
 [11786](#), [11980](#), [12063](#), [12096](#), [12323](#),
 [12439](#), [12448](#), [12576](#), [13020](#), [13045](#)
`\tl_set_eq:cc` [4228](#),
 [4231](#), [5022](#), [5554](#), [6019](#), [9336](#), [10155](#)
`\tl_set_eq:cN`
 .. [4228](#), [4229](#), [5021](#), [5553](#), [6018](#), [10153](#)
`\tl_set_eq:Nc` [4228](#),
 [4230](#), [5020](#), [5552](#), [6017](#), [9514](#), [10154](#)
`\tl_set_eq:NN` [82](#), [4217](#), [4228](#), [4228](#), [5019](#),
 [5551](#), [5623](#), [5641](#), [6016](#), [10063](#), [10152](#)
`\tl_set_rescan:cnm` [4313](#)
`\tl_set_rescan:cno` [4313](#)
`\tl_set_rescan:cnx` [4340](#)
`\tl_set_rescan:Nnn`
 [84](#), [4313](#), [4313](#), [4336](#), [4337](#)
`\tl_set_rescan:Nno` [4313](#), [9853](#)
`\tl_set_rescan:Nnx` [4340](#), [4340](#), [4355](#)
`\tl_set_rescan_aux:NNnn`
 [4313](#), [4314](#), [4316](#), [4317](#)
`\tl_set_rescan_aux:NNnx`
 [4340](#), [4341](#), [4343](#), [4344](#)
`\tl_show:c` [4768](#), [10161](#)
`\tl_show:N` ... [92](#), [4768](#), [4768](#), [4769](#), [10160](#)
`\tl_show:n` [93](#), [4770](#), [4770](#), [5295](#),
 [5849](#), [6212](#), [6222](#), [7870](#), [8133](#), [8153](#)
`\tl_tail:f` [4642](#)
`\tl_tail:n` [91](#), [4642](#), [4646](#), [4650](#)
`\tl_tail:V` [4642](#)
`\tl_tail:v` [4642](#)
`\tl_tail:w` [91](#), [4642](#), [4643](#), [9868](#), [9879](#)
`\tl_tail_aux:w` [4647](#), [4648](#)
`\tl_tmp:w` [4391](#),
 [4410](#), [4415](#), [4511](#), [4512](#), [4597](#), [4635](#)
`\tl_to_lowercase:n` [85](#),
 [2266](#), [2280](#), [2667](#), [2707](#), [2800](#), [3067](#),
 [4368](#), [4368](#), [8211](#), [8551](#), [8957](#), [9011](#)
`\tl_to_str:c` [4565](#)
`\tl_to_str:N` [88](#), [4565](#), [4565](#), [4566](#), [8271](#), [8272](#)
`\tl_to_str:n` [88](#), [3029](#), [3970](#), [4388](#), [4455](#),
 [4468](#), [4564](#), [4564](#), [4654](#), [4663](#), [6025](#),
 [6094](#), [6115](#), [6135](#), [6136](#), [6276](#), [6277](#),
 [7703](#), [7787](#), [8216](#), [8364](#), [9126](#), [9159](#),
 [9448](#), [9458](#), [9480](#), [9558](#), [9574](#), [10145](#)
`\tl_to_uppercase:n` [85](#), [4368](#), [4369](#)
`\tl_trim_spaces:c` [4597](#)
`\tl_trim_spaces:N` .. [90](#), [4597](#), [4636](#), [4640](#)
`\tl_trim_spaces:n` ... [89](#), [4597](#), [4599](#),
 [4637](#), [4639](#), [5519](#), [5951](#), [9063](#), [9088](#)
`\tl_trim_spaces_aux:i:w`
 [4597](#), [4602](#), [4613](#), [4616](#), [5578](#)
`\tl_trim_spaces_aux:ii:w` [4607](#), [4621](#), [5582](#)

\tl_trim_spaces_aux_ii:w	\tl_trim_spaces_aux_ii:w	\token_if_eq_meaning:NNTF	2275
.....	4597	\token_if_eq_meaning:NNTF	52, 2290, 2952
\tl_trim_spaces_aux_iii:w	\token_if_eq_meaning_p:NN	2933, 3083
.....	4608, 4623, 4626, 4630, 5583	\token_if_expandable:N	2689, 2689
\tl_trim_spaces_aux_iv:w	4597, 4610, 4632	\token_if_expandable:NTF	53
\tl_use:c	4567, 4568, 5688	\token_if_group_begin:N	2588, 2588
\tl_use:N	88, 4567, 4567, 5687	\token_if_group_begin:NTF	51
\token_get_arg_spec:N	57, 3027, 3040	\token_if_group_end:N	2593, 2593
\token_get_prefix_arg_replacement_aux:wN	\token_if_group_end:NTF	51
.....	3027, 3028, 3035, 3044, 3053	\token_if_int_register:N	2698, 2724
\token_get_prefix_spec:N	57, 3027, 3031	\token_if_int_register:NTF	53
\token_get_replacement_spec:N	3027, 3049	\token_if_int_register_aux:w	2729, 2733
\token_get_replacement_text:N	57	\token_if_int_register_p_aux:w	2698
\token_if_active:N	2641, 2641	\token_if_letter:N	2631, 2631
\token_if_active:NF	3184	\token_if_letter:NTF	52
\token_if_active:NT	3183	\token_if_long_macro:N	2698, 2777
\token_if_active:NTF	52, 3185	\token_if_long_macro:NTF	53
\token_if_active_char:N	3169	\token_if_long_macro_aux:w	2779, 2782
\token_if_active_char:NF	3184	\token_if_long_macro_p_aux:w	2698
\token_if_active_char:NT	3183	\token_if_macro:N	2661, 2670
\token_if_active_char:NTF	3185	\token_if_macro:NTF
\token_if_active_char_p:N	3182	52, 2804, 3033, 3042, 3051
\token_if_active_p:N	3182	\token_if_macro_p_aux:w	2661, 2672, 2675
\token_if_alignment:N	2603, 2603	\token_if_math_shift:N	3169
\token_if_alignment:NF	3172	\token_if_math_shift:NF	3176
\token_if_alignment:NT	3171	\token_if_math_shift:NT	3175
\token_if_alignment:NTF	51, 3173	\token_if_math_shift:NTF	3177
\token_if_alignment_p:N	3170	\token_if_math_shift_p:N	3174
\token_if_alignment_tab:N	3169	\token_if_math_subscript:N	2621, 2621
\token_if_alignment_tab:NF	3172	\token_if_math_subscript:NTF	52
\token_if_alignment_tab:NT	3171	\token_if_math_superscript:N	2616, 2616
\token_if_alignment_tab:NTF	3173	\token_if_math_superscript:NTF	51
\token_if_alignment_tab_p:N	3170	\token_if_math_toggle:N	2598, 2598
\token_if_chardef:N	2698, 2710	\token_if_math_toggle:NF	3176
\token_if_chardef:NTF	53	\token_if_math_toggle:NT	3175
\token_if_chardef_aux:w	2712, 2715	\token_if_math_toggle:NTF	51, 3177
\token_if_chardef_p_aux:w	2698	\token_if_math_toggle_p:N	3174
\token_if_cs:N	2684, 2684	\token_if_mathchardef:N	2698, 2717
\token_if_cs:NTF	52	\token_if_mathchardef:NTF	53
\token_if_dim_register:N	2698, 2746	\token_if_mathchardef_aux:w	2719, 2722
\token_if_dim_register:NTF	53	\token_if_mathchardef_p_aux:w	2698
\token_if_dim_register_aux:w	2751, 2755	\token_if_other:N	2636, 2636
\token_if_dim_register_p_aux:w	2698	\token_if_other:NF	3180
\token_if_eq_catcode:NN	2651, 2651	\token_if_other:NT	3179
\token_if_eq_catcode:NNTF	52	\token_if_other:NTF	52, 3181
\token_if_eq_catcode_p:NN	\token_if_other_char:N	3169
.....	2931, 2932, 3081, 3082	\token_if_other_char:NF	3180
\token_if_eq_charcode:NN	2656, 2656	\token_if_other_char:NT	3179
\token_if_eq_charcode:NNTF	52	\token_if_other_char:NTF	3181
\token_if_eq_meaning:NN	2646, 2646	\token_if_other_char_p:N	3178

- \token_if_other_p:N 3178
 - \token_if_parameter:N 2608, 2610
 - \token_if_parameter:NTF 51
 - \token_if_primitive:N 2794, 2802
 - \token_if_primitive:NTF 54
 - \token_if_primitive_aux:NNw 2794, 2807, 2811
 - \token_if_primitive_aux_loop:N 2794, 2814, 2827, 2833
 - \token_if_primitive_aux_nullfont:N 2794, 2815, 2819
 - \token_if_primitive_aux_space:w 2794, 2813, 2818
 - \token_if_primitive_aux_undefined:N 2794, 2839, 2845
 - \token_if_primitive_auxii:Nw 2794, 2830, 2836
 - \token_if_protected_long_macro:N 2698, 2784
 - \token_if_protected_long_macro:NTF 53
 - \token_if_protected_long_macro_aux:w 2787, 2790
 - \token_if_protected_long_macro_p_aux:w 2698
 - \token_if_protected_macro:N 2698, 2768
 - \token_if_protected_macro:NTF 53
 - \token_if_protected_macro_aux:w 2771, 2774
 - \token_if_protected_macro_p_aux:w 2698
 - \token_if_skip_register:N 2698, 2735
 - \token_if_skip_register:NTF 53
 - \token_if_skip_register_aux:w 2740, 2744
 - \token_if_skip_register_p_aux:w 2698
 - \token_if_space:N 2626, 2626
 - \token_if_space:NTF 52
 - \token_if_toks_register:N 2698, 2757
 - \token_if_toks_register:NTF 53
 - \token_if_toks_register_aux:w 2762, 2766
 - \token_if_toks_register_p_aux:w 2698
 - \token_new:Nn 50, 2568, 2568, 2573, 2575–2577, 2579–2582
 - \token_to_meaning:N 50, 803, 803, 1190, 1200, 1213, 1832, 2673, 2713, 2720, 2730, 2741, 2752, 2763, 2772, 2780, 2788, 2808, 3036, 3045, 3054
 - \token_to_str:c 819, 820
 - \token_to_str:N 5, 51, 803, 804, 820, 1059, 1062, 1190, 1200, 1202, 1213, 1333, 1423, 1810, 2286, 4740, 5205, 5294, 5300, 5848, 5854, 6211, 6217, 6868, 6873, 7006, 7853, 7858, 8251–8255, 13173
 - \toks 659
 - \toksdef 360
 - \tolerance 570
 - \topmark 451
 - \topmarks 677
 - \topskip 581
 - \TotalHeight 7031, 7035, 7039, 7043, 7050, 7552, 7579, 7580
 - \tracingassigns 687
 - \tracingcommands 426
 - \tracinggroups 694
 - \tracingifs 690
 - \tracinglostchars 427
 - \tracingmacros 428
 - \tracingnesting 689
 - \tracingonline 429
 - \tracingoutput 430
 - \tracingpages 431
 - \tracingparagraphs 432
 - \tracingrestores 433
 - \tracingscantokens 688
 - \tracingstats 434
- U**
- \U 2705
 - \uccode 669
 - \uchyph 567
 - \underline 503
 - \unexpanded 179, 183, 682
 - \unhbox 608
 - \unhcopy 609
 - \unkern 533
 - \unless 673
 - \unpenalty 644
 - \unskip 531
 - \unvbox 610
 - \unvcopy 611
 - \uppercase 641
 - \use:c 16, 850, 850, 978, 1932, 1942, 2014, 2015, 3351, 3647, 3657, 3800, 3809, 3811, 3813, 3814, 3818, 3973, 8327, 8623, 8634, 8647, 8650, 8656, 8667, 8675, 8681, 8703, 8764, 8786, 8791, 8799, 8822, 8844, 9173, 9180, 9342, 9551, 9857, 9887, 9890, 9907, 9910, 9911, 9914, 9917, 10201, 10263, 10319, 10369,

- 11521, 11608, 11819, 12006, 12342,
12869, 12932, 12947, 12949, 13040
- `\use:n` 17, [860](#),
860, 994, 1023, 1281, 1428, 1430,
1434, 1442, 1444, 1452, 1456, 4716,
4733, 5201, 5418, 6040, 6269, 8969
- `\use:nn` .. [860](#), 861, 1535, 3027, 3968, 5810
- `\use:nnn` [860](#), 862
- `\use:nnnn` [860](#), 863
- `\use:x` 18, [851](#), 851, 8268
- `\use_i:nn` 17, [864](#), 864, 890, 1080,
1109, 1137, 1292, 1432, 1446, 1454,
9957, 10003, 10028, 10596, 10915,
11272, 11305, 12098, 12426, 12565
- `\use_i:nnn`
17, [866](#), 866, 1091, 1320, 3036, 12065
- `\use_i:nnnn` 18, [866](#), 870
- `\use_i_after_else:nw` [1507](#), 1508
- `\use_i_after_fi:nw` [1507](#), 1507
- `\use_i_after_or:nw` [1507](#), 1509
- `\use_i_after_orelse:nw` [1507](#), 1510
- `\use_i_delimit_by_q_nil:nw` . 19, [877](#), 877
- `\use_i_delimit_by_q_recursion_stop:nw`
..... 19, [45](#),
[877](#), 879, 2400, 2416, 5843, 6205, 6283
- `\use_i_delimit_by_q_stop:nw`
..... 19, [877](#), 878, 1782, 5920
- `\use_i_ii:nnn` 18, [866](#), 869, 1560
- `\use_ii:nn` 17, [864](#),
865, 892, 1082, 1111, 1139, 1294,
1429, 1435, 1443, 1457, 6032, 9033
- `\use_ii:nnn` . 17, [866](#), 867, 1093, 3045, 9081
- `\use_ii:nnnn` 18, [866](#), 871
- `\use_iii:nnn` 17, [866](#), 868, 3054
- `\use_iii:nnnn` 18, [866](#), 872
- `\use_iv:nnnn` 18, [866](#), 873
- `\use_none:n` 18, [880](#), 880,
994, 1023, 1062, 1064, 1283, 1427,
1431, 1433, 1441, 1445, 1453, 1455,
1792, 1856, 2402, 2418, 2842, 3578,
3693, 3697, 3702, 4432, 4593, 4633,
4719, 4737, 4766, 4808, 5005, 5155,
5184, 5217, 5412, 5439, 5440, 5592,
5728, 5970, 8191, 8193, 9030, 9063,
9971, 10018, 10043, 10092, 10134,
10546, 10582, 10655, 10681, 10735,
10856, 10999, 11319, 11338, 11497,
11552, 11594, 11638, 11795, 11919,
11989, 12211, 12328, 12371, 12755
- `\use_none:nn`
... [880](#), 881, 1791, 4804, 5082, 12819
- `\use_none:nnn` .. [880](#), 882, 1790, 6112, 9074
- `\use_none:nnnn` [880](#), 883, 1789, 9928
- `\use_none:nnnnn` [880](#), 884, 1788
- `\use_none:nnnnnn` [880](#), 885, 1787
- `\use_none:nnnnnnn` [880](#), 886, 1786
- `\use_none:nnnnnnnn` [880](#), 887, 1785
- `\use_none:nnnnnnnnn`
..... [880](#), 888, 1298, 1783, 1784
- `\use_none_delimit_by_q_nil:w` 18, [874](#), 874
- `\use_none_delimit_by_q_recursion_stop:w`
..... 18, [45](#), [874](#), 876, 976, 1044,
1776, 2394, 2409, 4563, 5842, 6204
- `\use_none_delimit_by_q_stop:w`
..... 18, [874](#), 875, 2305, 2309,
4395, 5715, 5907, 5913, 8357, 8371
- `\usepackage` 223
- ## V
- `\vadjust` 544
- `\valign` 379
- `\vbadness` 619
- `\vbox` 614
- `\vbox:n` 128, [6469](#), 6469
- `\vbox_gset:cn` [6474](#)
- `\vbox_gset:cw` [6490](#), 6502
- `\vbox_gset:Nn` 128, [6474](#), 6475, 6477
- `\vbox_gset:Nw` . 129, [6490](#), 6492, 6495, 6501
- `\vbox_gset_end` 129
- `\vbox_gset_end:` [6490](#), 6497, 6503
- `\vbox_gset_inline_begin:c` ... [6498](#), 6502
- `\vbox_gset_inline_begin:N` ... [6498](#), 6501
- `\vbox_gset_inline_end:` [6498](#), 6503
- `\vbox_gset_to_ht:cnn` [6484](#)
- `\vbox_gset_to_ht:Nnn` 129, [6484](#), 6486, 6489
- `\vbox_gset_top:cn` [6478](#)
- `\vbox_gset_top:Nn` . 128, [6478](#), 6480, 6483
- `\vbox_set:cn` [6474](#)
- `\vbox_set:cw` [6490](#), 6499
- `\vbox_set:Nn` .. 128, [6474](#), 6474–6476, 6917
- `\vbox_set:Nw`
129, [6490](#), 6490, 6493, 6494, 6498, 6961
- `\vbox_set_end` 129
- `\vbox_set_end:` ... [6490](#), 6496, 6500, 6967
- `\vbox_set_inline_begin:c` [6498](#), 6499
- `\vbox_set_inline_begin:N` [6498](#), 6498
- `\vbox_set_inline_end:` [6498](#), 6500
- `\vbox_set_split_to_ht:Nnn` 129, [6508](#), 6508
- `\vbox_set_to_ht:cnn` [6484](#)

<code>\vbox_set_to_ht:Nnn</code>	<code>\vtop</code>	615
..... 129 , 6484 , 6484 , 6487 , 6488		
<code>\vbox_set_top:cn</code>		W
<code>\vbox_set_top:Nn</code>	<code>\wd</code>	662
..... 128 , 6478 , 6478 , 6481 , 6482 , 6928 , 6971	<code>\widowpenalties</code>	722
<code>\vbox_to_ht:nn</code>	<code>\widowpenalty</code>	549
..... 128 , 6471 , 6471	<code>\Width</code>	7031 , 7036 ,
<code>\vbox_to_zero:n</code> 7040 , 7044 , 7051 , 7549 , 7582 , 7583	
..... 128 , 6471 , 6473	<code>\write</code>	412
<code>\vbox_top:n</code>		
..... 128 , 6469 , 6470		X
<code>\vbox_unpack:c</code>		<code>\X</code>
<code>\vbox_unpack:N</code> 2701 , 2705
..... 129 , 6504 , 6504 , 6506 , 6928 , 6971	<code>\xdef</code>	353
<code>\vbox_unpack_clear:c</code>	<code>\xetex_if_engine:</code>	1427
..... 6504	<code>\xetex_if_engine:F</code>	1434 , 1445
<code>\vbox_unpack_clear:N</code> 129 , 6504 , 6505 , 6507	<code>\xetex_if_engine:T</code>	1433 , 1444
<code>\vcenter</code>	<code>\xetex_if_engine:TF</code>	1435 , 1446
..... 465	<code>\xetex_if_engine_p:</code>	1438 , 1448 , 1506
<code>\vcoffin_set:cn</code>	<code>\xetex_if_engineTF</code>	4 , 22
..... 6913	<code>\xetex_XeTeXversion:D</code>	754 , 1439
<code>\vcoffin_set:cnw</code>	<code>\XeTeXversion</code>	754
..... 6957	<code>\xleaders</code>	538
<code>\vcoffin_set:Nnn</code> ..	<code>\xspaceskip</code>	572
..... 132 , 6913 , 6913 , 6939		
<code>\vcoffin_set:Nnw</code> ..		Y
..... 132 , 6957 , 6957 , 6986	<code>\Y</code>	2702 , 2705
<code>\vcoffin_set_end</code>	<code>\year</code>	653
..... 132		
<code>\vcoffin_set_end:</code>		Z
..... 6957 , 6964 , 6985	<code>\Z</code>	1817 , 1825 , 2703 , 2705
<code>\vfil</code>	<code>\z@</code>	4056
..... 526		
<code>\vfill</code>		
..... 528		
<code>\vfilneg</code>		
..... 527		
<code>\vfuzz</code>		
..... 621		
<code>\voffset</code>		
..... 596		
<code>\voidb@x</code>		
..... 6419		
<code>\vrule</code>		
..... 535		
<code>\vsize</code>		
..... 578		
<code>\vskip</code>		
..... 529		
<code>\vsplit</code>		
..... 607		
<code>\vss</code>		
..... 530		