

# The `unravel` package: watching TeX digest tokens\*

Bruno Le Floch

2024/01/05

## Contents

<b>1</b>	<b>unravel documentation</b>	<b>2</b>
1.1	Commands	2
1.2	Examples	3
1.3	Options	4
1.4	Differences between <code>unravel</code> and TeX's processing	5
1.5	Future perhaps	6
<b>2</b>	<b>unravel implementation</b>	<b>6</b>
2.1	Primitives, variants, and helpers	10
2.1.1	Adjustments to <code>expl3</code>	10
2.1.2	Renamed primitives	10
2.1.3	Variants	11
2.1.4	Miscellaneous helpers	11
2.1.5	String helpers	13
2.1.6	Helpers for control flow	15
2.1.7	Helpers concerning tokens	15
2.1.8	Helpers for previous input	18
2.2	Variables	20
2.2.1	User interaction	20
2.2.2	Working with tokens	22
2.2.3	Numbers and conditionals	24
2.2.4	Boxes and groups	24
2.2.5	Constants	25
2.2.6	TeX parameters	25
2.3	Numeric codes	26
2.4	Get next token	40
2.5	Manipulating the input	46
2.5.1	Elementary operations	46
2.5.2	Insert token for error recovery	52
2.5.3	Macro calls	52
2.6	Expand next token	54
2.7	Basic scanning subroutines	56

---

\*This file has version number 0.3c, last revised 2024/01/05.

2.8	Working with boxes	75
2.9	Paragraphs	79
2.10	Groups	81
2.11	Modes	84
2.12	Commands	85
2.12.1	Characters: from 0 to 15	86
2.12.2	Boxes: from 16 to 31	90
2.12.3	From 32 to 47	95
2.12.4	Maths: from 48 to 56	99
2.12.5	From 57 to 70	102
2.12.6	Extensions	104
2.12.7	Assignments	111
2.13	Expandable primitives	121
2.13.1	Conditionals	129
2.14	User interaction	138
2.14.1	Print	138
2.14.2	Prompt	144
2.14.3	Errors	148
2.15	Keys	149
2.16	Main command	151
2.17	Messages	154

## 1 unravel documentation

The aim of this L<sup>A</sup>T<sub>E</sub>X package is to help debug complicated macros. This is done by letting the user step through the execution of some T<sub>E</sub>X code, going through the details of nested expansions, performing assignments, as well as some simple typesetting commands. To use this package, one should normally run T<sub>E</sub>X in a terminal.

### 1.1 Commands

---

`\unravel` [*key-value list*] {*code*}

This command shows in the terminal the steps performed by T<sub>E</sub>X when running the *code*. By default, it pauses to let the user read the description of every step: simply press `<return>` to proceed. Typing `s<integer>` instead will go forward *integer* steps somewhat silently. In the future it will be possible to use a negative *integer* to go back a few steps. Typing `h` gives a list of various other possibilities. The available *key-value* options are described in Section 1.3.

---

`\unravelsetup` {*options*}

Sets *options* that apply to all subsequent `\unravel`. See options in Section 1.3.

---

`\unravel:nn` {*options*} {*code*}

See `\unravel`.

<code>\unravel_get:nnN</code>	<code>\unravel_get:nnN {&lt;options&gt;} {&lt;code&gt;} &lt;tl var&gt;</code>
	Performs <code>\unravel:nn</code> with the <code>&lt;options&gt;</code> and <code>&lt;code&gt;</code> then saves the output into the <code>&lt;tl var&gt;</code> . The option <code>mute</code> is useful in this case.
<code>\unravel_setup:n</code>	<code>\unravel_setup:n {&lt;options&gt;}</code>
	See <code>\unravelsetup</code> .

## 1.2 Examples

The `unravel` package is currently based on the behaviour of `pdfTeX`, but it should work in all engines supported by `expl3` (`pdfTeX`, `XYTeX`, `LuaTeX`, `epTeX`, `eupTeX`) as long as none of the primitives specific to those engines is used. Any difference between how `unravel` and `(pdf)TeX` process a given piece of code, unless described in the section 1.4, should be reported on the issue tracker (<https://github.com/blefloch/latex-unravel/issues>).

As a simple example, one can run `LATeX` on the following file.

```
\documentclass{article}
\usepackage{unravel}
\unravel
{
  \title{My title}
  \author{Me}
  \date{\today}
}
\begin{document}
\maketitle
\end{document}
```

A more elaborate example is to understand how `\newcommand` works.

```
\documentclass{article}
\usepackage{unravel}
\begin{document}
\unravel
{
  \newcommand*{\foo}[1]{bar(#1)}
  \foo{3}
}
\end{document}
```

The `unravel` package understands deeply nested expansions as can be seen for instance by unravelling functions from `l3fp`, such as with the following code (given the current default settings, this code runs for roughly 2000 steps: you can type `s1980` as a response to the prompt, then press “enter” a few times to see the last few steps of expansion).

```
\documentclass{article}
\usepackage{unravel}
\begin{document}
\ExplSyntaxOn
\unravel { \fp_eval:n { 3.45 * 2 pi } }
\ExplSyntaxOff
\end{document}
```

Given all the work that `unravel` has to do to emulate  $\TeX$ , it is not fast on very large pieces of code. For instance, running it on `\documentclass{article}` takes about thirty seconds on my machine, and finishes after somewhat less than 21000 steps.

```
\RequirePackage{unravel}
\unravel{\documentclass{article}\relax}
\usepackage{lipsum}
\begin{document}
\lipsum
\end{document}
```

The `\relax` command is needed after `\documentclass{article}` because this command tries to look for an optional argument: `\unravel` would not find any token, and would give up, as  $\TeX$  would if your file ended just after `\documentclass{article}`. After running the above through `pdf $\TeX$` , one can check that the result is identical to that without `unravel`. Note that `\unravel{\usepackage{lipsum}\relax}`, despite taking roughly as many steps to complete, is ten times slower, because `\newcommand` uses delimited arguments, which prevent some optimizations that `unravel` can otherwise obtain. For comparison, `\unravel{\lipsum[1-30]}` also takes 20000 step and is ten times faster than loading the package.

### 1.3 Options

<u>explicit-prompt</u>	Boolean option (default <code>false</code> ) determining whether to give an explicit prompt. If <code>true</code> , the text “Your input=” will appear at the beginning of lines where user input is expected.
<u>internal-debug</u>	Boolean option (default <code>false</code> ) used to debug <code>unravel</code> itself.
<u>machine</u>	Option which takes no value and makes <code>unravel</code> produce an output that is somewhat more suitable for automatic processing. In particular, it sets <code>max-action</code> , <code>max-output</code> , <code>max-input</code> to very large values, and <code>number-steps</code> to <code>false</code> .
<u>max-action</u> <u>max-output</u> <u>max-input</u>	Integer options (defaults 50, 300, 300) determining the maximum number of characters displayed for the action, the output part, and the input part.
<u>mute</u>	Make none of the steps produce any output, by setting <code>trace-assigns</code> , <code>trace-expansion</code> , <code>trace-other</code> , <code>welcome-message</code> to <code>false</code> . This is only useful with <code>\unravel_get:nnN</code> or when other options change some of these settings.
<u>number-steps</u>	Boolean option (default <code>true</code> ) determining whether to number steps.

---

`online` Integer option (default 1) determining where to write the output: terminal and log if the option is positive, log only if the option is zero, neither if the option is negative.

---

`output-file` Name of a file where the output is written instead of the log file. Setting this option also sets `online` to zero. If `online` is further modified to be positive then the output is written to the terminal as well (hence inevitably to the log file), while if it is made negative then the output is written nowhere.

---

`prompt-input` Comma-delimited list option (empty by default) whose items are used one by one as if the user typed them at the prompt. Since the key-value list is itself comma-delimited, the value here must be wrapped in braces. For instance, `prompt-input = {s10, m, u\def}` skips 10 steps, shows the first token's meaning, then continues silently until the first token is `\def`, and any subsequent prompt is treated normally with user interaction. This can be useful when repeatedly debugging complicated code when the issue is known to lie quite late in the code.

As for any `clist`, spaces are discarded around each comma and empty entries are removed, then for each item one pair of braces is removed (if any is present); to get an empty item use an empty brace group, such as in `prompt-input = {s10, {}, x}`. Category codes are those in effect when the `prompt-input` option is read.

---

`trace-assigns`  
`trace-expansion`  
`trace-other` Boolean options (default `true`) controlling what steps produce any output at all. The keys `trace-assigns`, `trace-expansion`, `trace-other` control tracing of different types of steps.

---

`welcome-message` Boolean option (default `true`) determining whether to display the welcome message.

## 1.4 Differences between `unravel` and `TEX`'s processing

Bugs are listed at <https://github.com/blefloch/latex-unravel/issues>.

Differences.

- Kerning between letters of a word is omitted, which can lead to incorrect widths.
- Some primitives are not implemented yet: alignments (`\halign`, `\valign`, `\noalign`, `\omit`, `\span`, `\cr`, `\crr`, `&`), some math mode primitives, and `\pdfprimitive`, as well as many primitives specific to engines other than pdf`TEX`. This list may sadly be incomplete!
- `\aftergroup` is only partially implemented.
- `\everyhbox`, `\everyvbox`, `\everymath`, `\everydisplay`, `\lastkern`, `\lastnodetype`, `\lastpenalty`, `\lastskip`, `\currentifbranch` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgrouptype` too.
- Setting `\globaldefs` to a non-zero value may cause problems.
- Tokens passed to `\aftergroup` are lost when `unravel` is done.

- For `unravel`, category codes are fixed when a file is read using `\input`, while `TeX` only fixes category codes when the corresponding characters are converted to tokens. Similarly, the argument of `\scantokens` is converted to the new category code regime in one go, and the result must be balanced.
- Explicit begin-group and end-group characters other than the usual left and right braces may make `unravel` choke, or may be silently replaced by the usual left and right braces.
- `\endinput` is ignored with a warning, as it is very difficult to implement it in a way similar to `TeX`'s, and as it is most often used at the very end of files, in a redundant way.
- `\outer` is not supported.
- `\unravel` cannot be nested.
- Control sequences of the form `\notexpanded:...` are reserved for use by `unravel`.

## 1.5 Future perhaps

- Use the `file-error` fatal error message: first implement `\@@_file_if_exist:nTF` and use it to determine whether `\input` will throw a fatal error in `\batchmode` and `\nonstopmode`.
- Use the `interwoven-preambles` fatal error message once alignments are implemented.
- Find out why so many input levels are used (see the log of the `unravel003` testfile for instance)

## 2 unravel implementation

Some support packages are loaded first, then we declare the package's name, date, version, and purpose.

```
1 <*package>
2 <@@=unravel>
```

Catcode settings. In a group, set `\c` to be a synonym of `\catcode` for short, set the catcode of space to be 10 (using `\fam` to avoid needing a space or an equal sign to separate the two integer arguments of `\catcode`) and that of `%` to be 14 (using `\fam` again to avoid needing the digit 7 to have catcode other: we need the digit 5 anyway in two steps). Then make `-`, `6`, `7`, `8`, `9` other (we must assume that 0 through 5 are already other), and make `:`, `_`, `h`, `j`, `k`, `q`, `s`, `w`, `x`, `y`, `z` letters (other lowercase letters already need to be letters in the rest of the code). Make sure there is no `\endlinechar`. We are finally ready to safely test whether the package has already been loaded and bail out in case it has. Expanding `\fi` before ending the group ensures that the whole line has been read by `TeX` before restoring earlier catcodes.

```
3 \begingroup\let\c\catcode\fam32\c\fam10\advance\fam5\c\fam14\c45 12 %
4 \c54 12\c55 12\c56 12\c57 12\c58 11\c95 11\c104 11\c106 11\c107 11 %
5 \c113 11\c115 11\c119 11\c120 11\c121 11\c122 11\endlinechar-1 %
6 \expandafter\ifx\c\name unravel\endcsname\relax
7 \else\endinput\expandafter\endgroup\fi
```

Set T and X to be letters for an error message. Set up braces and # for definitions, = for nicer character code assignments, > for integer comparison, + for integer expressions.

```

8 \c84 11\c88 11\c35 6\c123 1\c125 2\c62 12\c61 12\c43 12 %
   If  $\varepsilon$ -TeX's \numexpr or \protected are not available, bail out with an error.
9 \expandafter\ifx\csname numexpr\endcsname\relax
10 \errmessage{unravel requires \numexpr from eTeX}
11 \endinput\expandafter\endgroup\fi
12 \expandafter\ifx\csname protected\endcsname\relax
13 \errmessage{unravel requires \protected from eTeX}
14 \endinput\expandafter\endgroup\fi

```

If unravel is loaded within a group, bail out because expl3 would not be loaded properly.

```

15 \expandafter\ifx\csname currentgrouplevel\endcsname\relax\else
16 \ifnum\currentgrouplevel>1 \errmessage{unravel loaded in a group}
17 \endinput\expandafter\expandafter\expandafter\endgroup\fi\fi
   Make spaces ignored and make ~ a space, to prettify code.
18 \catcode 32 = 9 \relax
19 \catcode 126 = 10 \relax

```

`\l__unravel_setup_restore_tl` This token list variable will contain code to restore category codes to their value when the package was loaded.

```
20 \gdef \l__unravel_setup_restore_tl { }
```

*(End of definition for \l\_\_unravel\_setup\_restore\_tl.)*

`\__unravel_setup_restore:` Use the token list to restore catcodes to their former values, then empty the list since there is no catcode to restore anymore. This mechanism cannot be nested.

```

21 \protected \gdef \__unravel_setup_restore:
22 {
23   \l__unravel_setup_restore_tl
24   \def \l__unravel_setup_restore_tl { }
25 }

```

*(End of definition for \\_\_unravel\_setup\_restore:.)*

`\__unravel_setup_save:` This saves into `\l__unravel_setup_restore_tl` the current catcodes (from 0 to 255 only), `\endlinechar`, `\escapechar`, `\newlinechar`.

```

26 \protected \gdef \__unravel_setup_save:
27 {
28   \edef \l__unravel_setup_restore_tl
29   {
30     \__unravel_setup_save_aux:w 0 =
31     \endlinechar = \the \endlinechar
32     \escapechar = \the \escapechar
33     \newlinechar = \the \newlinechar
34     \relax
35   }
36 }
37 \long \gdef \__unravel_setup_save_aux:w #1 =
38 {
39   \catcode #1 = \the \catcode #1 ~
40   \ifnum 255 > #1 ~

```

```

41     \expandafter \_unravel_setup_save_aux:w
42     \the \numexpr #1 + 1 \expandafter =
43     \fi
44 }

```

(End of definition for `\_unravel_setup_save:` and `\_unravel_setup_save_aux:n`.)

```

\_unravel_setup_catcodes:nmn This sets all characters from #1 to #2 (inclusive) to have catcode #3.
45 \protected \long \gdef \_unravel_setup_catcodes:nmn #1 #2 #3
46 {
47   \ifnum #1 > #2 ~ \else
48     \catcode #1 = #3 ~
49     \expandafter \_unravel_setup_catcodes:nmn \expandafter
50     { \the \numexpr #1 + 1 } {#2} {#3}
51   \fi
52 }

```

(End of definition for `\_unravel_setup_catcodes:nmn`.)

`\_unravel_setup_latex:` This saves the catcodes and related parameters, then sets them to the value they normally have in a L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> package (in particular, @ is a letter).

```

53 \protected \gdef \_unravel_setup_latex:
54 {
55   \_unravel_setup_save:
56   \_unravel_setup_catcodes:nmn {0} {8} {15}
57   \catcode 9 = 10 ~
58   \catcode 10 = 12 ~
59   \catcode 11 = 15 ~
60   \catcode 12 = 13 ~
61   \catcode 13 = 5 ~
62   \_unravel_setup_catcodes:nmn {14} {31} {15}
63   \catcode 32 = 10 ~
64   \catcode 33 = 12 ~
65   \catcode 34 = 12 ~
66   \catcode 35 = 6 ~
67   \catcode 36 = 3 ~
68   \catcode 37 = 14 ~
69   \catcode 38 = 4 ~
70   \_unravel_setup_catcodes:nmn {39} {63} {12}
71   \_unravel_setup_catcodes:nmn {64} {90} {11}
72   \catcode 91 = 12 ~
73   \catcode 92 = 0 ~
74   \catcode 93 = 12 ~
75   \catcode 94 = 7 ~
76   \catcode 95 = 8 ~
77   \catcode 96 = 12 ~
78   \_unravel_setup_catcodes:nmn {97} {122} {11}
79   \catcode 123 = 1 ~
80   \catcode 124 = 12 ~
81   \catcode 125 = 2 ~
82   \catcode 126 = 13 ~
83   \catcode 127 = 15 ~
84   \_unravel_setup_catcodes:nmn {128} {255} {12}
85   \endlinechar = 13 ~

```



```

86     \escapechar = 92 ~
87     \newlinechar = 10 ~
88   }

```

(End of definition for `\_unravel_setup_latex:`)

`\_unravel_setup_unravel:` Catcodes for unravel (in particular, @ is other, : and \_ are letters, spaces are ignored, ~ is a space).

```

89 \protected \gdef \_unravel_setup_unravel:
90   {
91     \_unravel_setup_save:
92     \_unravel_setup_catcodes:nmn {0} {8} {15}
93     \catcode 9 = 9 ~
94     \catcode 10 = 12 ~
95     \catcode 11 = 15 ~
96     \catcode 12 = 13 ~
97     \catcode 13 = 5 ~
98     \_unravel_setup_catcodes:nmn {14} {31} {15}
99     \catcode 32 = 9 ~
100    \catcode 33 = 12 ~
101    \catcode 34 = 12 ~
102    \catcode 35 = 6 ~
103    \catcode 36 = 3 ~
104    \catcode 37 = 14 ~
105    \catcode 38 = 4 ~
106    \_unravel_setup_catcodes:nmn {39} {57} {12}
107    \catcode 58 = 11 ~
108    \_unravel_setup_catcodes:nmn {59} {64} {12}
109    \_unravel_setup_catcodes:nmn {65} {90} {11}
110    \catcode 91 = 12 ~
111    \catcode 92 = 0 ~
112    \catcode 93 = 12 ~
113    \catcode 94 = 7 ~
114    \catcode 95 = 11 ~
115    \catcode 96 = 12 ~
116    \_unravel_setup_catcodes:nmn {97} {122} {11}
117    \catcode 123 = 1 ~
118    \catcode 124 = 12 ~
119    \catcode 125 = 2 ~
120    \catcode 126 = 10 ~
121    \catcode 127 = 15 ~
122    \_unravel_setup_catcodes:nmn {128} {255} {12}
123    \escapechar = 92 ~
124    \endlinechar = 32 ~
125    \newlinechar = 10 ~
126   }

```

(End of definition for `\_unravel_setup_unravel:`)

End the group where all catcodes where changed, but expand `\_unravel_setup_latex:` to sanitize catcodes again outside the group. The catcodes are saved.

```

127 \expandafter \endgroup \_unravel_setup_latex:

```

Load the gtl dependency.

```

128 \RequirePackage{gtl}[2024/01/04]

```

Before loading `unravel`, restore catcodes, so that the implicit `\ExplSyntaxOn` in `\ProvidesExplPackage` picks up the correct catcodes to restore when `\ExplSyntaxOff` is run at the end of the package. The place where catcodes are restored are beyond `unravel`'s reach, which is why we cannot bypass `expl3` and simply restore the catcodes once everything is done. To avoid issues with crazy catcodes, make `TeX` read the arguments of `\ProvidesExplPackage` before restoring catcodes. Then immediately go to the catcodes we want.

```

129 \csname use:n\endcsname
130 {%
131   \csname __unravel_setup_restore:\endcsname
132   \ProvidesExplPackage
133     {unravel} {2024/01/05} {0.3c} {Watching TeX digest tokens}%
134   \csname __unravel_setup_unravel:\endcsname
135 }%
```

## 2.1 Primitives, variants, and helpers

### 2.1.1 Adjustments to `expl3`

In upcoming versions of `expl3`, the `\group_align_safe_begin:` and `\group_align_safe_end:` commands may involve an explicit end-group character token with non-standard character code, which would wrongly be normalized by `gtl` (used by `unravel`), hence break. To avoid this we change here the definitions slightly.

```

136 \cs_gset:Npn \group_align_safe_begin:
137   { \exp:w \if_false: { \fi: -' } \exp_stop_f: }
138 \cs_gset:Npn \group_align_safe_end:
139   { \if_int_compare:w '{ = \c_zero_int } \fi: }
```

### 2.1.2 Renamed primitives

Copy primitives which are used multiple times, to avoid littering the code with `:D` commands. Primitives are left as `:D` in the code when that is clearer (typically when testing the meaning of a token against that of a primitive).

```

140 \cs_new_eq:NN \__unravel_currentgrouptype: \tex_currentgrouptype:D
141 \cs_new_protected:Npn \__unravel_set_escapechar:n
142   { \int_set:Nn \tex_escapechar:D }
143 \cs_new_eq:NN \__unravel_everyeof:w \tex_everyeof:D
144 \cs_new_eq:NN \__unravel_everypar:w \tex_everypar:D
145 \cs_new_eq:NN \__unravel_hbox:w \tex_hbox:D
146 \cs_new_eq:NN \__unravel_mag: \tex_mag:D
147 \cs_new_eq:NN \__unravel_nullfont: \tex_nullfont:D
148 \cs_new_eq:NN \__unravel_the:w \tex_the:D
149 \cs_new_eq:NN \__unravel_number:w \tex_number:D
```

*(End of definition for `\__unravel_currentgrouptype:` and others.)*

`\__unravel_special_relax:` A special marker slightly different from `\relax` (its `\meaning` is `\relax` but it differs from `\relax` according to `\ifx`). In the right-hand side of our assignment, `\__unravel_special_relax:` could be replaced by any other expandable command.

```

150 \exp_after:wN \cs_new_eq:NN
151   \exp_after:wN \__unravel_special_relax:
152   \exp_not:N \__unravel_special_relax:
```

(End of definition for `\__unravel_special_relax:`.)

`\c__unravel_prompt_ior` These are not quite primitives, but are very low-level ior streams to prompt the user  
`\c__unravel_noprompt_ior` explicitly or not.

```
153 \int_const:Nn \c__unravel_prompt_ior { 16 }
154 \int_const:Nn \c__unravel_noprompt_ior { -1 }
```

(End of definition for `\c__unravel_prompt_ior` and `\c__unravel_noprompt_ior`.)

### 2.1.3 Variants

Variants that we need.

```
155 \cs_generate_variant:Nn \seq_push:Nn { Nf }
156 \cs_generate_variant:Nn \str_head:n { f }
157 \cs_generate_variant:Nn \tl_to_str:n { o }
158 \cs_generate_variant:Nn \tl_if_eq:nnTF { o }
159 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
160 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNTF { V }
161 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
162 \cs_generate_variant:Nn \gtl_gput_right:Nn { NV }
163 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
164 \cs_generate_variant:Nn \gtl_if_tl:NT { c }
165 \cs_generate_variant:Nn \gtl_to_str:N { c }
166 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
167 \cs_generate_variant:Nn \gtl_get_left:NN { c }
168 \cs_generate_variant:Nn \gtl_gset:Nn { c }
169 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
170 \cs_generate_variant:Nn \gtl_gclear:N { c }
171 \cs_generate_variant:Nn \gtl_gclear_new:N { c }
172 \cs_generate_variant:Nn \gtl_left_tl:N { c }
```

`\__unravel_tl_if_in:ooTF` Analogue of `\tl_if_in:ooTF` but with an extra group because that function redefines an auxiliary that may appear in the code being debugged (see Github issue #27).

```
173 \cs_new_protected:Npn \__unravel_tl_if_in:ooTF #1#2#3#4
174 {
175   \group_begin:
176   \exp_args:Noo \tl_if_in:nnTF {#1} {#2}
177   { \group_end: #3 } { \group_end: #4 }
178 }
```

(End of definition for `\__unravel_tl_if_in:ooTF`.)

### 2.1.4 Miscellaneous helpers

`\__unravel_tmp:w` Temporary function used to define other functions.

```
179 \cs_new_protected:Npn \__unravel_tmp:w { }
```

(End of definition for `\__unravel_tmp:w`.)

`\__unravel_file_get:nN`  
`\__unravel_file_get_aux:wN`

```
180 \cs_set_protected:Npn \__unravel_tmp:w #1
181 {
182   \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
183   {
```

```

184     \group_begin:
185     \_unravel_everyeof:w { #1 ##2 }
186     \exp_after:wN \_unravel_file_get_aux:wN
187     \exp_after:wN \prg_do_nothing:
188     \tex_input:D ##1 \scan_stop:
189   }
190 \cs_new_protected:Npn \_unravel_file_get_aux:wN ##1 #1 ##2
191 {
192   \group_end:
193   \tl_set:Ne ##2
194   { \exp_not:o {##1} \exp_not:V \_unravel_everyeof:w }
195 }
196 }
197 \exp_args:No \_unravel_tmp:w { \token_to_str:N : : }

```

(End of definition for \\_unravel\_file\_get:nN and \\_unravel\_file\_get\_aux:wN.)

\\_unravel\_tl\_first\_int:N  
 \\_unravel\_tl\_first\_int\_aux:Nn

Function that finds an explicit number in a token list. This is used for instance when implementing \read, to find the stream  $\langle number \rangle$  within the whole \read  $\langle number \rangle$  to  $\langle cs \rangle$  construction. The auxiliary initially has itself as a first argument, and once a first digit is found it has \use\_none\_delimit\_by\_q\_stop:w. That first argument is used whenever what follows is not a digit, hence initially we loop, while after the first digit is found any non-digit stops the recursion. If no integer is found, 0 is left in the token list. The surrounding \int\_eval:n lets us dump digits in the input stream while keeping the function fully expandable.

```

198 \cs_new:Npn \_unravel_tl_first_int:N #1
199 {
200   \int_eval:n
201   {
202     \exp_after:wN \_unravel_tl_first_int_aux:Nn
203     \exp_after:wN \_unravel_tl_first_int_aux:Nn
204     #1 ? 0 ? \q_stop
205   }
206 }
207 \cs_new:Npn \_unravel_tl_first_int_aux:Nn #1#2
208 {
209   \tl_if_single:nT {#2}
210   {
211     \token_if_eq_catcode:NNT + #2
212     {
213       \if_int_compare:w 1 < 1 #2 \exp_stop_f:
214       #2
215       \exp_after:wN \use_i_ii:nnn
216       \exp_after:wN \_unravel_tl_first_int_aux:Nn
217       \exp_after:wN \use_none_delimit_by_q_stop:w
218       \fi:
219     }
220   }
221   #1
222 }

```

(End of definition for \\_unravel\_tl\_first\_int:N and \\_unravel\_tl\_first\_int\_aux:Nn.)

\\_unravel\_use\_ii\_i:nn

```

223 \cs_new:Npn \_unravel_use_ii_i:nn #1#2 { #2 #1 }

```

(End of definition for `\__unravel_use_ii_i:nn`.)

```

\__unravel_prompt_input:Nn
\__unravel_prompt_input:w 224 \cs_new_protected:Npn \__unravel_prompt_input:Nn #1#2
\__unravel_prompt_input_aux:w 225 {
\q__unravel_recursion_tail:w 226 \clist_gset:Ne #1
\q__unravel_recursion_tail 227 { \__unravel_prompt_input:w \prg_do_nothing: #2 , \q__unravel_recursion_tail , }
228 }
229 \cs_new:Npn \__unravel_prompt_input:w #1 ,
230 {
231 \tl_trim_spaces_apply:oN {#1} \__unravel_use_ii_i:nn
232 \__unravel_prompt_input_aux:w ,
233 }
234 \cs_new:Npn \__unravel_prompt_input_aux:w #1 ,
235 {
236 \__unravel_use_none_delimit_by_q_recursion_tail:w #1
237 \use_none:nmmm \q__unravel_recursion_tail
238 { \tl_to_str:n {#1} } ,
239 \__unravel_prompt_input:w \prg_do_nothing:
240 }
241 \cs_new:Npn \__unravel_use_none_delimit_by_q_recursion_tail:w
242 #1 \q__unravel_recursion_tail { }
243 \quark_new:N \q__unravel_recursion_tail

```

(End of definition for `\__unravel_prompt_input:Nn` and others.)

### 2.1.5 String helpers

```

\__unravel_strip_escape:w
\__unravel_strip_escape_aux:N
\__unravel_strip_escape_aux:w

```

This is based on the 2013-07-19 (and earlier) version of `\cs_to_str:N`. There are three cases. If the escape character is printable, the charcode test is false, and `\__unravel_strip_escape_aux:N` removes one character. If the escape character is a space, the charcode test is true, and if there is no escape character, the test is unfinished after `\token_to_str:N \`. In both of those cases, `\__unravel_strip_escape_aux:w` inserts `-\@@_number:w \fi: \c_zero_int`. If the escape character was a space, the test was true, and `\int_value:w` converts `\c_zero_int` to 0, hence the leading roman numeral expansion removes a space from what follows (it is important that what follows cannot start with a digit). Otherwise, the test takes `-` as its second operand, is false, and the roman numeral expansion only sees `\c_zero_int`, thus does not remove anything from what follows.

```

244 \cs_new:Npn \__unravel_strip_escape:w
245 {
246 \tex_romannumeral:D
247 \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
248 \__unravel_strip_escape_aux:N
249 }
250 \cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero_int }
251 \cs_new:Npn \__unravel_strip_escape_aux:w #1#2
252 { - \__unravel_number:w #1 \c_zero_int }

```

(End of definition for `\__unravel_strip_escape:w`, `\__unravel_strip_escape_aux:N`, and `\__unravel_strip_escape_aux:w`.)

`\__unravel_to_str:Nn` Use the type-appropriate conversion to string.

```
253 \cs_new:Npn \__unravel_to_str:Nn #1
254 {
255   \if_meaning:w T #1
256   \exp_after:wN \tl_to_str:n
257   \else:
258     \exp_after:wN \gtl_to_str:n
259   \fi:
260 }
```

*(End of definition for \\_\_unravel\_to\_str:Nn.)*

`\__unravel_str_truncate_left:nn` Truncate the string `#1` to a maximum of `#2` characters. If it is longer, replace some characters on the left of the string by `(123~more~chars)~` with the appropriate number instead of 123. In any reasonable case, 25 is big enough to fit this extra text.

`\__unravel_str_truncate_left_aux:nnn`

```
261 \cs_new:Npn \__unravel_str_truncate_left:nn #1#2
262 {
263   \exp_args:Nf \__unravel_str_truncate_left_aux:nnn
264   { \str_count:n {#1} } {#1} {#2}
265 }
266 \cs_new:Npn \__unravel_str_truncate_left_aux:nnn #1#2#3
267 {
268   \int_compare:nNnTF {#1} > {#3}
269   {
270     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
271     \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
272   }
273   { \tl_to_str:n {#2} }
274 }
```

*(End of definition for \\_\_unravel\_str\_truncate\_left:nn and \\_\_unravel\_str\_truncate\_left\_aux:nnn.)*

`\__unravel_str_truncate_right:nn` Truncate the string `#1` to a maximum of `#2` characters. If it is longer, replace some characters on the right of the string by `~(123~more~chars)` with the appropriate number instead of 123. In any reasonable case, 25 is big enough to fit this extra text.

`\__unravel_str_truncate_right_aux:nnn`

```
275 \cs_new:Npn \__unravel_str_truncate_right:nn #1#2
276 {
277   \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
278   { \str_count:n {#1} } {#1} {#2}
279 }
280 \cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
281 {
282   \int_compare:nNnTF {#1} > {#3}
283   {
284     \str_range:nnn {#2} { 1 } { #3 - 25 } ~
285     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
286   }
287   { \tl_to_str:n {#2} }
288 }
```

*(End of definition for \\_\_unravel\_str\_truncate\_right:nn and \\_\_unravel\_str\_truncate\_right\_aux:nnn.)*

## 2.1.6 Helpers for control flow

`\__unravel_exit:w` Jump to the very end of this instance of `\unravel`.

```

289 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
290 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }
291 \cs_new:Npn \__unravel_exit_hard:w #1 \__unravel_exit_point: #2 \__unravel_exit_point: { }

```

*(End of definition for `\__unravel_exit:w`, `\__unravel_exit_hard:w`, and `\__unravel_exit_point:`.)*

`\__unravel_break:w` Useful to jump out of complicated conditionals.

```

292 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
293 \cs_new:Npn \__unravel_break:w #1 \__unravel_break_point: { }

```

*(End of definition for `\__unravel_break:w` and `\__unravel_break_point:`.)*

`\__unravel_cmd_if_internal:TF` Test whether the `\l__unravel_head_cmd_int` denotes an “internal” command, between `min_internal` and `max_internal` (see Section 2.3).

```

294 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
295 {
296   \int_compare:nNnTF
297     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
298     { \prg_return_false: }
299     {
300       \int_compare:nNnTF
301         \l__unravel_head_cmd_int
302         > { \__unravel_tex_use:n { max_internal } }
303         { \prg_return_false: }
304         { \prg_return_true: }
305     }
306 }

```

*(End of definition for `\__unravel_cmd_if_internal:TF`.)*

## 2.1.7 Helpers concerning tokens

`\__unravel_active_do:nn` Apply some code to an active character token constructed from its character code.

```

307 \cs_new_protected:Npn \__unravel_active_do:nn #1#2
308 {
309   \group_begin:
310   \char_set_active_eq:nN {#1} \scan_stop:
311   \use:e
312   {
313     \group_end:
314     \exp_not:n {#2} { \char_generate:nn {#1} { 13 } }
315   }
316 }

```

*(End of definition for `\__unravel_active_do:nn`.)*

`\__unravel_token_to_char:N` From the meaning of a character token (with arbitrary character code, except active),  
`\__unravel_meaning_to_char:n` extract the character itself (with string category codes). This is somewhat robust against  
`\__unravel_meaning_to_char:o` wrong input.

```

317 \cs_new:Npn \__unravel_meaning_to_char:n #1
318   { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
319 \cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop

```

```

320 { \_unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
321 \cs_new:Npn \_unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
322 { \tl_if_empty:nTF {#2} { ~ } {#2} }
323 \cs_generate_variant:Nn \_unravel_meaning_to_char:n { o }
324 \cs_new:Npn \_unravel_token_to_char:N #1
325 { \_unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

(End of definition for \\_unravel\_token\_to\_char:N and others.)

\\_unravel\_token\_if\_expandable\_p:N We need to cook up our own version of \token\_if\_expandable:NTF because the expl3  
\\_unravel\_token\_if\_expandable:NTF one does not think that undefined is expandable.

```

326 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
327 { p , T , F , TF }
328 {
329   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
330   \prg_return_false:
331   \else:
332   \prg_return_true:
333   \fi:
334 }

```

(End of definition for \\_unravel\_token\_if\_expandable:NTF.)

\\_unravel\_token\_if\_protected\_p:N Returns true if the token is either not expandable or is a protected macro.

```

\_unravel_token_if_protected:NTF
335 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
336 { p , T , F , TF }
337 {
338   \_unravel_token_if_expandable:NTF #1
339   {
340     \token_if_protected_macro:NTF #1
341     { \prg_return_true: }
342     {
343       \token_if_protected_long_macro:NTF #1
344       { \prg_return_true: }
345       { \prg_return_false: }
346     }
347   }
348   { \prg_return_true: }
349 }

```

(End of definition for \\_unravel\_token\_if\_protected:NTF.)

\\_unravel\_token\_if\_active\_char:NTF Lowercase the token after setting its \lccode (more precisely the \lccode of the first character in its string representation) to a known value, then compare the result with that active character.

```

350 \group_begin:
351   \char_set_catcode_active:n { 'Z }
352   \prg_new_protected_conditional:Npnn \_unravel_token_if_active_char:N #1
353   { TF }
354   {
355     \group_begin:
356     \exp_args:Ne \char_set_lccode:nn
357     { ' \exp_args:No \str_head:n { \token_to_str:N #1 } }
358     { ' Z }

```



```

359     \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
360     { \group_end: \prg_return_true: }
361     { \group_end: \prg_return_false: }
362   }
363 \group_end:

```

(End of definition for `\_unravel_token_if_active_char:NTF`.)

`\_unravel_token_if_definable:NTF` Within a group, set the escape character to a non-space value (backslash). Convert the token to a string with `\token_to_str:N`. The result is multiple characters if the token is a control sequence, and a single character otherwise (even for explicit catcode 6 character tokens which would be doubled if we used `\tl_to_str:n` instead of `\token_to_str:N`). Thus `\str_tail:n` gives a non-empty result exactly for control sequences. Those are definable (technically, not always: `\expandafter\font\csname\endcsname=cmr10\expandafter\def\the\csname\endcsname{}`). For characters just check for active characters. In both cases remember to end the group.

```

364 \group_begin:
365   \char_set_catcode_active:n { 'Z }
366   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
367     { TF }
368   {
369     \group_begin:
370     \_unravel_set_escapechar:n { 92 }
371     \tl_set:Ne \l__unravel_tmpa_tl
372       { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
373     \tl_if_empty:NTF \l__unravel_tmpa_tl
374       {
375         \_unravel_token_if_active_char:NTF #1
376         { \group_end: \prg_return_true: }
377         { \group_end: \prg_return_false: }
378       }
379     { \group_end: \prg_return_true: }
380   }
381 \group_end:

```

(End of definition for `\_unravel_token_if_definable:NTF`.)

`\_unravel_gtl_if_head_is_definable:NTF` Tests if a generalized token list is a single control sequence or a single active character. First test that it is single, then filter out the case of (explicit) begin-group, end-group, and blank space characters: those are neither control sequences nor active. Then feed the single normal token to a first auxiliary.

```

382 \prg_new_protected_conditional:Npnn \_unravel_gtl_if_head_is_definable:N #1
383   { TF , F }
384   {
385     \gtl_if_single_token:NTF #1
386     {
387       \gtl_if_head_is_N_type:NTF #1
388       {
389         \gtl_head_do:NN #1 \_unravel_token_if_definable:NTF
390         { \prg_return_true: }
391         { \prg_return_false: }
392       }
393     }
394   }

```

```

395     { \prg_return_false: }
396   }

```

(End of definition for `\__unravel_gtl_if_head_is_definable:NTF`.)

### 2.1.8 Helpers for previous input

Count prev input levels, skipping empty ones (of either tl or gtl type).

```

\__unravel_prev_input_count:
  \__unravel_prev_input_count_aux:n
  \__unravel_prev_input_count_aux:Nn
397 \cs_new:Npn \__unravel_prev_input_count:
398   {
399     \int_eval:n
400     {
401       0
402       \seq_map_function:NN \g__unravel_prev_input_seq
403       \__unravel_prev_input_count_aux:n
404     }
405   }
406 \cs_new:Npn \__unravel_prev_input_count_aux:n #1
407   { \__unravel_prev_input_count_aux:Nn #1 }
408 \cs_new:Npn \__unravel_prev_input_count_aux:Nn #1#2
409   {
410     \if_meaning:w T #1
411     \exp_after:wN \tl_if_empty:nF
412     \else:
413     \exp_after:wN \str_if_eq:onF \exp_after:wN \c_empty_gtl
414     \fi:
415     {#2} { + 1 }
416   }

```

(End of definition for `\__unravel_prev_input_count:`, `\__unravel_prev_input_count_aux:n`, and `\__unravel_prev_input_count_aux:Nn`.)

```

\__unravel_prev_input_gpush:
  \__unravel_prev_input_gpush:N
  \__unravel_prev_input_gpush_gtl:
  \__unravel_prev_input_gpush_gtl:N
  \__unravel_prev_input_gpush_aux:NN
247 \cs_new_protected:Npn \__unravel_prev_input_gpush:
248   { \seq_gput_right:Nn \g__unravel_prev_input_seq { T { } } }
249 \cs_new_protected:Npn \__unravel_prev_input_gpush:N
250   { \__unravel_prev_input_gpush_aux:NN T }
251 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
252   { \__unravel_prev_input_gpush_gtl:N \c_empty_gtl }
253 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
254   { \__unravel_prev_input_gpush_aux:NN G }
255 \cs_new_protected:Npn \__unravel_prev_input_gpush_aux:NN #1#2
256   { \seq_gput_right:Ne \g__unravel_prev_input_seq { #1 { \exp_not:o {#2} } } }

```

(End of definition for `\__unravel_prev_input_gpush:` and others.)

```

  \l__unravel_prev_aux_tl
\__unravel_prev_input_get:N
\__unravel_prev_input_gpop:N
  \__unravel_prev_input_gpop_gtl:N
  \__unravel_prev_input_aux:NNN
  \__unravel_prev_input_aux:NNNn
227 \tl_new:N \l__unravel_prev_aux_tl
228 \cs_new_protected:Npn \__unravel_prev_input_get:N
229   { \__unravel_prev_input_aux:NNN \seq_get_right:NN T }
230 \cs_new_protected:Npn \__unravel_prev_input_gpop:N
231   { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN T }
232 \cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
233   { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN G }
234 \cs_new_protected:Npn \__unravel_prev_input_aux:NNN #1#2#3

```

```

435 {
436 #1 \g__unravel_prev_input_seq \l__unravel_prev_aux_tl
437 \exp_after:wN \__unravel_prev_input_aux:NNNn
438 \exp_after:wN #2 \exp_after:wN #3 \l__unravel_prev_aux_tl
439 }
440 \cs_new_protected:Npn \__unravel_prev_input_aux:NNNn #1#2#3
441 {
442 \token_if_eq_meaning:NNTF #1#3
443 { \tl_set:Nn }
444 { \__unravel_error:nmnnn { prev-input } {#1} {#3} }
445 #2
446 }

```

(End of definition for \l\_\_unravel\_prev\_aux\_tl and others.)

```

\__unravel_prev_input_silent:n
\__unravel_prev_input_silent:V 447 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
\__unravel_prev_input_silent:e 448 {
\__unravel_prev_input:n 449 \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
\__unravel_prev_input:V 450 \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
\__unravel_prev_input:e 451 \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
452 }
453 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V , e }
454 \cs_new_protected:Npn \__unravel_prev_input:n #1
455 {
456 \__unravel_prev_input_silent:n {#1}
457 \__unravel_print_action:e { \tl_to_str:n {#1} }
458 }
459 \cs_generate_variant:Nn \__unravel_prev_input:n { V , e }

```

(End of definition for \\_\_unravel\_prev\_input\_silent:n and \\_\_unravel\_prev\_input:n.)

```

\__unravel_prev_input_gtl:N
460 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
461 {
462 \__unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
463 \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
464 \__unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
465 }

```

(End of definition for \\_\_unravel\_prev\_input\_gtl:N.)

\\_\_unravel\_prev\_input\_join\_get:nnN Pops the previous-input sequence twice to get some value in \l\_\_unravel\_head\_tl and some sign or decimal number in \l\_\_unravel\_tmpa\_tl. Combines them into a value, using the appropriate evaluation function, determined based on #1.

```

466 \cs_new_protected:Npn \__unravel_prev_input_join_get:nnN #1
467 {
468 \int_case:nnF {#1}
469 {
470 { 2 } { \__unravel_join_get_aux:NNnN \skip_eval:n \tex_glueexpr:D }
471 { 3 } { \__unravel_join_get_aux:NNnN \muskip_eval:n \tex_muexpr:D }
472 }
473 {
474 \__unravel_error:nmnnn { internal } { join-factor } { } { } { }
475 \__unravel_join_get_aux:NNnN \use:n \prg_do_nothing:

```

```

476     }
477   }
478   \cs_new_protected:Npn \__unravel_join_get_aux:NNnN #1#2#3#4
479   {
480     \__unravel_prev_input_gpop:N \l__unravel_head_tl
481     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
482     \tl_set:Ne #4 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl #3 } }
483   }

```

(End of definition for `\__unravel_prev_input_join_get:nnN` and `\__unravel_join_get_aux:NNnN`.)

## 2.2 Variables

### 2.2.1 User interaction

`\g__unravel_iow` The stream is used to implement the `output-file` option. At any given time it points to the file named `\g__unravel_current_output_file_tl`, unless that is empty, in which case the stream is closed. The idea is that we do not want to close the file in between different `\unravel` calls since file writing is not additive in T<sub>E</sub>X.

```

484 \iow_new:N \g__unravel_iow
485 \tl_new:N \g__unravel_current_output_file_tl

```

(End of definition for `\g__unravel_iow` and `\g__unravel_current_output_file_tl`.)

`\g__unravel_before_print_state_tl` Code to run before printing the state or before the prompt.

```

486 \tl_new:N \g__unravel_before_print_state_tl
487 \tl_new:N \g__unravel_before_prompt_tl

```

(End of definition for `\g__unravel_before_print_state_tl` and `\g__unravel_before_prompt_tl`.)

`\l__unravel_prompt_tmpa_int`

```

488 \int_new:N \l__unravel_prompt_tmpa_int

```

(End of definition for `\l__unravel_prompt_tmpa_int`.)

`\g__unravel_nonstop_int` The number of prompts to skip.

```

489 \int_new:N \g__unravel_nonstop_int

```

(End of definition for `\g__unravel_nonstop_int`.)

`\g__unravel_current_online_int` Temporary value replacing `\g__unravel_online_int` and that can be set through the prompt.

```

490 \int_new:N \g__unravel_current_online_int

```

(End of definition for `\g__unravel_current_online_int`.)

`\g__unravel_default_explicit_prompt_bool` Variables for the options `explicit-prompt`, `internal-debug`, `number-steps`, and so on. The `default_` booleans/integers store the default value of these options, and are affected by `\unravelsetup` or `\unravel_setup:n`.

```

491 \bool_new:N \g__unravel_default_explicit_prompt_bool
492 \bool_new:N \g__unravel_default_internal_debug_bool
493 \bool_new:N \g__unravel_default_number_steps_bool
494 \int_new:N \g__unravel_default_online_int
495 \tl_new:N \g__unravel_default_output_file_tl
496 \clist_new:N \g__unravel_default_prompt_input_clist

```

`\g__unravel_output_file_tl`

`\g__unravel_default_prompt_input_clist`

`\g__unravel_prompt_input_clist`

`\g__unravel_default_trace_assigns_bool`

`\g__unravel_trace_assigns_bool`

`\g__unravel_default_trace_expansion_bool`

`\g__unravel_trace_expansion_bool`

`\g__unravel_default_trace_other_bool`

`\g__unravel_trace_other_bool`

`\g__unravel_default_welcome_message_bool`

```

497 \bool_new:N \g__unravel_default_trace_assigns_bool
498 \bool_new:N \g__unravel_default_trace_expansion_bool
499 \bool_new:N \g__unravel_default_trace_other_bool
500 \bool_new:N \g__unravel_default_welcome_message_bool
501 \bool_gset_true:N \g__unravel_default_number_steps_bool
502 \int_gset:Nn \g__unravel_default_online_int { 1 }
503 \bool_gset_true:N \g__unravel_default_trace_assigns_bool
504 \bool_gset_true:N \g__unravel_default_trace_expansion_bool
505 \bool_gset_true:N \g__unravel_default_trace_other_bool
506 \bool_gset_true:N \g__unravel_default_welcome_message_bool
507 \bool_new:N \g__unravel_explicit_prompt_bool
508 \bool_new:N \g__unravel_internal_debug_bool
509 \bool_new:N \g__unravel_number_steps_bool
510 \int_new:N \g__unravel_online_int
511 \tl_new:N \g__unravel_output_file_tl
512 \clist_new:N \g__unravel_prompt_input_clist
513 \bool_new:N \g__unravel_trace_assigns_bool
514 \bool_new:N \g__unravel_trace_expansion_bool
515 \bool_new:N \g__unravel_trace_other_bool
516 \bool_new:N \g__unravel_welcome_message_bool

```

*(End of definition for \g\_\_unravel\_default\_explicit\_prompt\_bool and others.)*

`\g__unravel_step_int` Current expansion step.

```
517 \int_new:N \g__unravel_step_int
```

*(End of definition for \g\_\_unravel\_step\_int.)*

`\g__unravel_action_text_str` Text describing the action, displayed at each step. This should only be altered through `\__unravel_set_action_text:e`, which sets the escape character as appropriate before converting the argument to a string.

```
518 \str_new:N \g__unravel_action_text_str
```

*(End of definition for \g\_\_unravel\_action\_text\_str.)*

`\g__unravel_default_max_action_int` Maximum length of various pieces of what is shown on the terminal.

```

\g__unravel_default_max_output_int 519 \int_new:N \g__unravel_default_max_action_int
\g__unravel_default_max_input_int 520 \int_new:N \g__unravel_default_max_output_int
\g__unravel_max_action_int 521 \int_new:N \g__unravel_default_max_input_int
\g__unravel_max_output_int 522 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
\g__unravel_max_input_int 523 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
524 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
525 \int_new:N \g__unravel_max_action_int
526 \int_new:N \g__unravel_max_output_int
527 \int_new:N \g__unravel_max_input_int

```

*(End of definition for \g\_\_unravel\_default\_max\_action\_int and others.)*

`\g__unravel_speedup_macros_bool` If this boolean is true, speed up macros which have a simple parameter text. This may not be safe if very weird macros appear.

```

528 \bool_new:N \g__unravel_speedup_macros_bool
529 \bool_gset_true:N \g__unravel_speedup_macros_bool

```

*(End of definition for \g\_\_unravel\_speedup\_macros\_bool.)*

`\l__unravel_print_int` The length of one piece of the terminal output.

```
530 \int_new:N \l__unravel_print_int
```

*(End of definition for `\l__unravel_print_int`.)*

### 2.2.2 Working with tokens

`\g__unravel_input_int` The user input, at each stage of expansion, is stored in multiple `gtl` variables, from `\g__input_<n>_gtl` to `\g__unravel_input_1_gtl`. The split between variables is akin to  $\TeX$ 's input stack, and allows us to manipulate smaller token lists, speeding up processing. The total number  $\langle n \rangle$  of lists is `\g__unravel_input_int`. The highest numbered `gtl` represents input that comes to the left of lower numbered ones.

```
531 \int_new:N \g__unravel_input_int
```

*(End of definition for `\g__unravel_input_int`.)*

`\g__unravel_input_tmpa_int`

`\l__unravel_input_tmpa_tl`

```
532 \int_new:N \g__unravel_input_tmpa_int
```

```
533 \tl_new:N \l__unravel_input_tmpa_tl
```

*(End of definition for `\g__unravel_input_tmpa_int` and `\l__unravel_input_tmpa_tl`.)*

`\g__unravel_prev_input_seq`

`\l__unravel_prev_input_tl`

`\l__unravel_prev_input_gtl`

The different levels of expansion are stored in `\g__unravel_prev_input_seq`, with the innermost at the end of the sequence (otherwise the sequence would have to be reversed for display). When adding material to the last level of expansion, `\l__unravel_prev_input_tl` or `\l__unravel_prev_input_gtl` are used to temporarily store the last level of expansion.

```
534 \seq_new:N \g__unravel_prev_input_seq
```

```
535 \tl_new:N \l__unravel_prev_input_tl
```

```
536 \gtl_new:N \l__unravel_prev_input_gtl
```

*(End of definition for `\g__unravel_prev_input_seq`, `\l__unravel_prev_input_tl`, and `\l__unravel_prev_input_gtl`.)*

`\g__unravel_output_gtl`

Material that is “typeset” or otherwise sent further down  $\TeX$ 's digestion.

```
537 \gtl_new:N \g__unravel_output_gtl
```

*(End of definition for `\g__unravel_output_gtl`.)*

`\l__unravel_head_gtl`

`\l__unravel_head_tl`

`\l__unravel_head_token`

`\l__unravel_head_cmd_int`

`\l__unravel_head_char_int`

First token in the input, as a generalized token list (general case) or as a token list whenever this is possible. Also, a token set equal to it, and its command code and character code, following  $\TeX$ .

```
538 \gtl_new:N \l__unravel_head_gtl
```

```
539 \tl_new:N \l__unravel_head_tl
```

```
540 \cs_new_eq:NN \l__unravel_head_token ?
```

```
541 \int_new:N \l__unravel_head_cmd_int
```

```
542 \int_new:N \l__unravel_head_char_int
```

*(End of definition for `\l__unravel_head_gtl` and others.)*

`\l__unravel_head_meaning_tl`

```
543 \tl_new:N \l__unravel_head_meaning_tl
```

*(End of definition for `\l__unravel_head_meaning_tl`.)*

`\l__unravel_argi_tl` Token list variables used to store first/second arguments.

```
\l__unravel_argii_tl
544 \tl_new:N \l__unravel_argi_tl
545 \tl_new:N \l__unravel_argii_tl
```

*(End of definition for `\l__unravel_argi_tl` and `\l__unravel_argii_tl`.)*

`\l__unravel_tmpa_tl` Temporary storage. The `\l__unravel_unused_gtl` is only used once, to ignore some unwanted tokens.

```
\l__unravel_tmpb_gtl
\g__unravel_tmpc_tl
546 \tl_new:N \l__unravel_tmpa_tl
\l__unravel_tmpa_seq
547 \gtl_new:N \l__unravel_unused_gtl
\l__unravel_unused_gtl
548 \gtl_new:N \l__unravel_tmpb_gtl
\l__unravel_tmpb_token
549 \tl_new:N \g__unravel_tmpc_tl
550 \seq_new:N \l__unravel_tmpa_seq
551 \cs_new_eq:NN \l__unravel_tmpb_token ?
```

*(End of definition for `\l__unravel_tmpa_tl` and others.)*

`\l__unravel_defined_tl` The token that is defined by the prefixed command (such as `\chardef` or `\futurelet`), and the code to define it. We do not use the the previous-input sequence to store that code because this sequence contains a string representation of the code, which is not suitable for the definition. Using a single variable here is safe, as definitions cannot be nested. This is needed for expanding assignments, as expansion should be shown to the user, but then later should not be performed again when defining. It is also helpful in tracking some register assignments.

```
552 \tl_new:N \l__unravel_defined_tl
553 \tl_new:N \l__unravel_defining_tl
```

*(End of definition for `\l__unravel_defined_tl` and `\l__unravel_defining_tl`.)*

`\__unravel_inaccessible:w`

```
554 \cs_new_eq:NN \__unravel_inaccessible:w ?
```

*(End of definition for `\__unravel_inaccessible:w`.)*

`\g__unravel_lastnamedcs_tl` Used for LuaTeX's `\lastnamedcs` primitive.

```
555 \tl_new:N \g__unravel_lastnamedcs_tl
```

*(End of definition for `\g__unravel_lastnamedcs_tl`.)*

`\g__unravel_after_assignment_gtl` Global variables keeping track of the state of TeX. Token to insert after the next assignment. Is `\setbox` currently allowed? Should `\input` expand?

```
\g__unravel_set_box_allowed_bool
\g__unravel_name_in_progress_bool
556 \gtl_new:N \g__unravel_after_assignment_gtl
557 \bool_new:N \g__unravel_set_box_allowed_bool
558 \bool_new:N \g__unravel_name_in_progress_bool
```

*(End of definition for `\g__unravel_after_assignment_gtl`, `\g__unravel_set_box_allowed_bool`, and `\g__unravel_name_in_progress_bool`.)*

`\l__unravel_after_group_gtl` Tokens to insert after the current group ends. This variable must be emptied at the beginning of every group.

```
559 \gtl_new:N \l__unravel_after_group_gtl
```

*(End of definition for `\l__unravel_after_group_gtl`.)*

`\c__unravel_parameters_tl` Used to determine if a macro has simple parameters or not.

```
560 \group_begin:
561   \cs_set_nopar:Npe \__unravel_tmp:w #1 { \c_hash_str #1 }
562   \tl_const:Nc \c__unravel_parameters_tl
563     { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
564 \group_end:
```

*(End of definition for `\c__unravel_parameters_tl`.)*

### 2.2.3 Numbers and conditionals

`\g__unravel_val_level_int` See TeX's `cur_val_level` variable. This is set by `\__unravel_rescan_something_internal:n` to

- 0 for integer values,
- 1 for dimension values,
- 2 for glue values,
- 3 for mu glue values,
- 4 for font identifiers,
- 5 for token lists.

```
565 \int_new:N \g__unravel_val_level_int
```

*(End of definition for `\g__unravel_val_level_int`.)*

`\g__unravel_if_limit_tl` Stack for what TeX calls `if_limit`, and its depth.

```
\g__unravel_if_limit_int 566 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 567 \int_new:N \g__unravel_if_limit_int
568 \int_new:N \g__unravel_if_depth_int
```

*(End of definition for `\g__unravel_if_limit_tl`, `\g__unravel_if_limit_int`, and `\g__unravel_if_depth_int`.)*

`\l__unravel_if_nesting_int`

```
569 \int_new:N \l__unravel_if_nesting_int
```

*(End of definition for `\l__unravel_if_nesting_int`.)*

### 2.2.4 Boxes and groups

`\l__unravel_leaders_box_seq` A stack of letters: the first token in the token list is `h` if the innermost explicit box (created with `\vtop`, `\vbox`, or `\hbox`) appears in a horizontal (or math) mode leaders construction; it is `v` if the innermost explicit box appears in a vertical mode leaders construction; it is `Z` otherwise.

```
570 \seq_new:N \l__unravel_leaders_box_seq
```

*(End of definition for `\l__unravel_leaders_box_seq`.)*

`\g__unravel_ends_int` Number of times `\end` will be put back into the input in case there remains to ship some pages.

```
571 \int_new:N \g__unravel_ends_int
572 \int_gset:Nn \g__unravel_ends_int { 3 }
```

*(End of definition for `\g__unravel_ends_int`.)*



## 2.2.5 Constants

```

\c__unravel_plus_tl
\c__unravel_minus_tl 573 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 574 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 575 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 576 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 577 \tl_const:Nn \c__unravel_lq_tl { ‘ }
\c__unravel_dq_tl 578 \tl_const:Nn \c__unravel_rq_tl { ’ }
\c__unravel_lp_tl 579 \tl_const:Nn \c__unravel_dq_tl { " } %"
\c__unravel_rp_tl 580 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 581 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 582 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 583 \tl_const:Nn \c__unravel_comma_tl { , }
584 \tl_const:Nn \c__unravel_point_tl { . }

```

(End of definition for `\c__unravel_plus_tl` and others.)

```

\c__unravel_frozen_relax_gtl TeX's frozen_relax, inserted by \__unravel_insert_relax:.
585 \gtl_const:Ne \c__unravel_frozen_relax_gtl { \if_int_compare:w 0 = 0 \fi: }

```

(End of definition for `\c__unravel_frozen_relax_gtl`.)

## 2.2.6 TeX parameters

`\g__unravel_mag_set_int` The first time TeX uses the value of `\mag`, it stores it in a global parameter `mag_set` (initially 0 to denote not being set). Any time TeX needs the value of `\mag`, it checks that the value matches `mag_set`. This is done in `unravel` by `\__unravel_prepare_mag:`, storing `mag_set` in `\g__unravel_mag_set_int`.

```
586 \int_new:N \g__unravel_mag_set_int
```

(End of definition for `\g__unravel_mag_set_int`.)

`\__unravel_prepare_mag:` Used whenever TeX needs the value of `\mag`.

```

587 \cs_new_protected:Npn \__unravel_prepare_mag:
588 {
589   \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
590   {
591     \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
592     {
593       \__unravel_tex_error:nn { incompatible-mag } { }
594       \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
595     }
596   }
597   \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
598   {
599     \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
600     \int_gset:Nn \__unravel_mag: { 1000 }
601   }
602   \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
603 }

```

(End of definition for `\__unravel_prepare_mag:`.)

## 2.3 Numeric codes

First we define some numeric codes, following Section 15 of the T<sub>E</sub>X web code, then we associate a command code to each T<sub>E</sub>X primitive, and a character code, to decide what action to perform upon seeing them.

```

\__unravel_tex_const:nn
\__unravel_tex_use:n
604 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
605 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
606 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }

```

*(End of definition for \\_\_unravel\_tex\_const:nn and \\_\_unravel\_tex\_use:n.)*

```

\__unravel_tex_primitive:nnn
\__unravel_tex_primitive_pdf:nnn
607 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
608 {
609   \tl_const:ce { c__unravel_tex_#1_tl }
610   { { \__unravel_tex_use:n {#2} } {#3} }
611 }
612 \cs_new_protected:Npn \__unravel_tex_primitive_pdf:nnn #1#2#3
613 {
614   \sys_if_engine_pdftex:F
615   { \__unravel_tex_primitive:nnn {#1} {#2} {#3} }
616   \__unravel_tex_primitive:nnn { pdf #1 } {#2} {#3}
617 }

```

*(End of definition for \\_\_unravel\_tex\_primitive:nnn and \\_\_unravel\_tex\_primitive\_pdf:nnn.)*

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
618 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
619 {
620   \cs_new_protected:cpn
621   { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
622 }
623 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
624 {
625   \cs_new_eq:cc
626   { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
627   { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
628 }

```

*(End of definition for \\_\_unravel\_new\_tex\_cmd:nn and \\_\_unravel\_new\_eq\_tex\_cmd:nn.)*

```

\__unravel_new_tex_expandable:nn
629 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
630 {
631   \cs_new_protected:cpn
632   { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
633 }

```

*(End of definition for \\_\_unravel\_new\_tex\_expandable:nn.)*

Contrarily to T<sub>E</sub>X, all macros are call, no long\_call and the like.

```

634 \__unravel_tex_const:nn { relax } { 0 }
635 \__unravel_tex_const:nn { begin-group_char } { 1 }
636 \__unravel_tex_const:nn { end-group_char } { 2 }

```

```

637 \_unravel_tex_const:nn { math_char } { 3 }
638 \_unravel_tex_const:nn { tab_mark } { 4 }
639 \_unravel_tex_const:nn { alignment_char } { 4 }
640 \_unravel_tex_const:nn { car_ret } { 5 }
641 \_unravel_tex_const:nn { macro_char } { 6 }
642 \_unravel_tex_const:nn { superscript_char } { 7 }
643 \_unravel_tex_const:nn { subscript_char } { 8 }
644 \_unravel_tex_const:nn { endv } { 9 }
645 \_unravel_tex_const:nn { blank_char } { 10 }
646 \_unravel_tex_const:nn { the_char } { 11 }
647 \_unravel_tex_const:nn { other_char } { 12 }
648 \_unravel_tex_const:nn { par_end } { 13 }
649 \_unravel_tex_const:nn { stop } { 14 }
650 \_unravel_tex_const:nn { delim_num } { 15 }
651 \_unravel_tex_const:nn { max_char_code } { 15 }
652 \_unravel_tex_const:nn { char_num } { 16 }
653 \_unravel_tex_const:nn { math_char_num } { 17 }
654 \_unravel_tex_const:nn { mark } { 18 }
655 \_unravel_tex_const:nn { xray } { 19 }
656 \_unravel_tex_const:nn { make_box } { 20 }
657 \_unravel_tex_const:nn { hmove } { 21 }
658 \_unravel_tex_const:nn { vmove } { 22 }
659 \_unravel_tex_const:nn { un_hbox } { 23 }
660 \_unravel_tex_const:nn { un_vbox } { 24 }
661 \_unravel_tex_const:nn { remove_item } { 25 }
662 \_unravel_tex_const:nn { hskip } { 26 }
663 \_unravel_tex_const:nn { vskip } { 27 }
664 \_unravel_tex_const:nn { mskip } { 28 }
665 \_unravel_tex_const:nn { kern } { 29 }
666 \_unravel_tex_const:nn { mkern } { 30 }
667 \_unravel_tex_const:nn { leader_ship } { 31 }
668 \_unravel_tex_const:nn { halign } { 32 }
669 \_unravel_tex_const:nn { valign } { 33 }
670 \_unravel_tex_const:nn { no_align } { 34 }
671 \_unravel_tex_const:nn { vrule } { 35 }
672 \_unravel_tex_const:nn { hrule } { 36 }
673 \_unravel_tex_const:nn { insert } { 37 }
674 \_unravel_tex_const:nn { vadjust } { 38 }
675 \_unravel_tex_const:nn { ignore_spaces } { 39 }
676 \_unravel_tex_const:nn { after_assignment } { 40 }
677 \_unravel_tex_const:nn { after_group } { 41 }
678 \_unravel_tex_const:nn { break_penalty } { 42 }
679 \_unravel_tex_const:nn { start_par } { 43 }
680 \_unravel_tex_const:nn { ital_corr } { 44 }
681 \_unravel_tex_const:nn { accent } { 45 }
682 \_unravel_tex_const:nn { math_accent } { 46 }
683 \_unravel_tex_const:nn { discretionary } { 47 }
684 \_unravel_tex_const:nn { eq_no } { 48 }
685 \_unravel_tex_const:nn { left_right } { 49 }
686 \_unravel_tex_const:nn { math_comp } { 50 }
687 \_unravel_tex_const:nn { limit_switch } { 51 }
688 \_unravel_tex_const:nn { above } { 52 }
689 \_unravel_tex_const:nn { math_style } { 53 }
690 \_unravel_tex_const:nn { math_choice } { 54 }

```

```

691 \_unravel_tex_const:nn { non_script } { 55 }
692 \_unravel_tex_const:nn { vcenter } { 56 }
693 \_unravel_tex_const:nn { case_shift } { 57 }
694 \_unravel_tex_const:nn { message } { 58 }
695 \_unravel_tex_const:nn { extension } { 59 }
696 \_unravel_tex_const:nn { in_stream } { 60 }
697 \_unravel_tex_const:nn { begin_group } { 61 }
698 \_unravel_tex_const:nn { end_group } { 62 }
699 \_unravel_tex_const:nn { omit } { 63 }
700 \_unravel_tex_const:nn { ex_space } { 64 }
701 \_unravel_tex_const:nn { no_boundary } { 65 }
702 \_unravel_tex_const:nn { radical } { 66 }
703 \_unravel_tex_const:nn { end_cs_name } { 67 }
704 \_unravel_tex_const:nn { min_internal } { 68 }
705 \_unravel_tex_const:nn { char_given } { 68 }
706 \_unravel_tex_const:nn { math_given } { 69 }
707 \_unravel_tex_const:nn { last_item } { 70 }
708 \_unravel_tex_const:nn { max_non_prefixed_command } { 70 }
709 \_unravel_tex_const:nn { toks_register } { 71 }
710 \_unravel_tex_const:nn { assign_toks } { 72 }
711 \_unravel_tex_const:nn { assign_int } { 73 }
712 \_unravel_tex_const:nn { assign_dimen } { 74 }
713 \_unravel_tex_const:nn { assign_glue } { 75 }
714 \_unravel_tex_const:nn { assign_mu_glue } { 76 }
715 \_unravel_tex_const:nn { assign_font_dimen } { 77 }
716 \_unravel_tex_const:nn { assign_font_int } { 78 }
717 \_unravel_tex_const:nn { set_aux } { 79 }
718 \_unravel_tex_const:nn { set_prev_graf } { 80 }
719 \_unravel_tex_const:nn { set_page_dimen } { 81 }
720 \_unravel_tex_const:nn { set_page_int } { 82 }
721 \_unravel_tex_const:nn { set_box_dimen } { 83 }
722 \_unravel_tex_const:nn { set_shape } { 84 }
723 \_unravel_tex_const:nn { def_code } { 85 }
724 \_unravel_tex_const:nn { def_family } { 86 }
725 \_unravel_tex_const:nn { set_font } { 87 }
726 \_unravel_tex_const:nn { def_font } { 88 }
727 \_unravel_tex_const:nn { register } { 89 }
728 \_unravel_tex_const:nn { max_internal } { 89 }
729 \_unravel_tex_const:nn { advance } { 90 }
730 \_unravel_tex_const:nn { multiply } { 91 }
731 \_unravel_tex_const:nn { divide } { 92 }
732 \_unravel_tex_const:nn { prefix } { 93 }
733 \_unravel_tex_const:nn { let } { 94 }
734 \_unravel_tex_const:nn { shorthand_def } { 95 }
735 \_unravel_tex_const:nn { read_to_cs } { 96 }
736 \_unravel_tex_const:nn { def } { 97 }
737 \_unravel_tex_const:nn { set_box } { 98 }
738 \_unravel_tex_const:nn { hyph_data } { 99 }
739 \_unravel_tex_const:nn { set_interaction } { 100 }
740 \_unravel_tex_const:nn { letterspace_font } { 101 }
741 \_unravel_tex_const:nn { pdf_copy_font } { 102 }
742 \_unravel_tex_const:nn { max_command } { 102 }
743 \_unravel_tex_const:nn { undefined_cs } { 103 }
744 \_unravel_tex_const:nn { expand_after } { 104 }

```

```

745 \__unravel_tex_const:nn { no_expand } { 105 }
746 \__unravel_tex_const:nn { input } { 106 }
747 \__unravel_tex_const:nn { if_test } { 107 }
748 \__unravel_tex_const:nn { fi_or_else } { 108 }
749 \__unravel_tex_const:nn { cs_name } { 109 }
750 \__unravel_tex_const:nn { convert } { 110 }
751 \__unravel_tex_const:nn { the } { 111 }
752 \__unravel_tex_const:nn { top_bot_mark } { 112 }
753 \__unravel_tex_const:nn { call } { 113 }
754 \__unravel_tex_const:nn { end_template } { 117 }

```

So far we've implemented properly [71,104]; [107,113].

A few minor differences with pdf $\TeX$ 's internal numbers are as follows.

- `case_shift` is shifted by 3983.
- `assign_toks` is shifted by `local_base=3412`.
- `assign_int` is shifted by `int_base=5263`.
- `assign_dimen` is shifted by `dimen_base=5830`.
- `assign_glue` and `assign_mu_glue` are shifted by `glue_base=2882`.
- `set_shape` is shifted (in  $\varepsilon\text{-}\TeX$ ) by `local_base`.
- `def_code` and `def_family` is shifted by `cat_code_base=3983`.
- In  $\TeX$ , `inputlineno.char=3` and `badness.char=4`.

A special case for Lua $\TeX$  deals with the fact that the `\__unravel_special_relax:` has a strange meaning “[unknown command code! (0, 1)]”. For instance `\expandafter \show \noexpand \undefined` shows this.

```

755 \sys_if_engine luatex:T
756 {
757   \__unravel_tex_primitive:nnn
758     { \exp_after:wN \use_none:n \token_to_meaning:N \__unravel_special_relax: }
759     { relax } { 1 }
760 }
761 \__unravel_tex_primitive:nnn { relax } { relax } { 256 }
762 \__unravel_tex_primitive:nnn { span } { tab_mark } { 256 }
763 \__unravel_tex_primitive:nnn { cr } { car_ret } { 257 }
764 \__unravel_tex_primitive:nnn { crcr } { car_ret } { 258 }
765 \__unravel_tex_primitive:nnn { par } { par_end } { 256 }
766 \__unravel_tex_primitive:nnn { end } { stop } { 0 }
767 \__unravel_tex_primitive:nnn { dump } { stop } { 1 }
768 \__unravel_tex_primitive:nnn { delimiter } { delim_num } { 0 }
769 \__unravel_tex_primitive:nnn { char } { char_num } { 0 }
770 \__unravel_tex_primitive:nnn { mathchar } { math_char_num } { 0 }
771 \__unravel_tex_primitive:nnn { mark } { mark } { 0 }
772 \__unravel_tex_primitive:nnn { marks } { mark } { 5 }
773 \__unravel_tex_primitive:nnn { show } { xray } { 0 }
774 \__unravel_tex_primitive:nnn { showbox } { xray } { 1 }
775 \__unravel_tex_primitive:nnn { showthe } { xray } { 2 }
776 \__unravel_tex_primitive:nnn { showlists } { xray } { 3 }
777 \__unravel_tex_primitive:nnn { showgroups } { xray } { 4 }

```

```

778 \_unravel_tex_primitive:nnn { showtokens } { xray } { 5 }
779 \_unravel_tex_primitive:nnn { showifs } { xray } { 6 }
780 \_unravel_tex_primitive:nnn { box } { make_box } { 0 }
781 \_unravel_tex_primitive:nnn { copy } { make_box } { 1 }
782 \_unravel_tex_primitive:nnn { lastbox } { make_box } { 2 }
783 \_unravel_tex_primitive:nnn { vsplit } { make_box } { 3 }
784 \_unravel_tex_primitive:nnn { vtop } { make_box } { 4 }
785 \_unravel_tex_primitive:nnn { vbox } { make_box } { 5 }
786 \_unravel_tex_primitive:nnn { hbox } { make_box } { 106 }
787 \_unravel_tex_primitive:nnn { moveright } { hmove } { 0 }
788 \_unravel_tex_primitive:nnn { moveleft } { hmove } { 1 }
789 \_unravel_tex_primitive:nnn { lower } { vmove } { 0 }
790 \_unravel_tex_primitive:nnn { raise } { vmove } { 1 }
791 \_unravel_tex_primitive:nnn { unhbox } { un_hbox } { 0 }
792 \_unravel_tex_primitive:nnn { unhcopy } { un_hbox } { 1 }
793 \_unravel_tex_primitive:nnn { unvbox } { un_vbox } { 0 }
794 \_unravel_tex_primitive:nnn { unvcopy } { un_vbox } { 1 }
795 \_unravel_tex_primitive:nnn { pagediscards } { un_vbox } { 2 }
796 \_unravel_tex_primitive:nnn { splitdiscards } { un_vbox } { 3 }
797 \_unravel_tex_primitive:nnn { unpenalty } { remove_item } { 12 }
798 \_unravel_tex_primitive:nnn { unkern } { remove_item } { 11 }
799 \_unravel_tex_primitive:nnn { unskip } { remove_item } { 10 }
800 \_unravel_tex_primitive:nnn { hfil } { hskip } { 0 }
801 \_unravel_tex_primitive:nnn { hfill } { hskip } { 1 }
802 \_unravel_tex_primitive:nnn { hss } { hskip } { 2 }
803 \_unravel_tex_primitive:nnn { hfilneg } { hskip } { 3 }
804 \_unravel_tex_primitive:nnn { hskip } { hskip } { 4 }
805 \_unravel_tex_primitive:nnn { vfil } { vskip } { 0 }
806 \_unravel_tex_primitive:nnn { vfill } { vskip } { 1 }
807 \_unravel_tex_primitive:nnn { vss } { vskip } { 2 }
808 \_unravel_tex_primitive:nnn { vfilneg } { vskip } { 3 }
809 \_unravel_tex_primitive:nnn { vskip } { vskip } { 4 }
810 \_unravel_tex_primitive:nnn { mskip } { mskip } { 5 }
811 \_unravel_tex_primitive:nnn { kern } { kern } { 1 }
812 \_unravel_tex_primitive:nnn { mkern } { mkern } { 99 }
813 \_unravel_tex_primitive:nnn { shipout } { leader_ship } { 99 }
814 \_unravel_tex_primitive:nnn { leaders } { leader_ship } { 100 }
815 \_unravel_tex_primitive:nnn { cleaders } { leader_ship } { 101 }
816 \_unravel_tex_primitive:nnn { xleaders } { leader_ship } { 102 }
817 \_unravel_tex_primitive:nnn { halign } { halign } { 0 }
818 \_unravel_tex_primitive:nnn { valign } { valign } { 0 }
819 \_unravel_tex_primitive:nnn { beginL } { valign } { 4 }
820 \_unravel_tex_primitive:nnn { endL } { valign } { 5 }
821 \_unravel_tex_primitive:nnn { beginR } { valign } { 8 }
822 \_unravel_tex_primitive:nnn { endR } { valign } { 9 }
823 \_unravel_tex_primitive:nnn { noalign } { no_align } { 0 }
824 \_unravel_tex_primitive:nnn { vrule } { vrule } { 0 }
825 \_unravel_tex_primitive:nnn { hrule } { hrule } { 0 }
826 \_unravel_tex_primitive:nnn { insert } { insert } { 0 }
827 \_unravel_tex_primitive:nnn { vadjust } { vadjust } { 0 }
828 \_unravel_tex_primitive:nnn { ignorespaces } { ignore_spaces } { 0 }
829 \_unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
830 \_unravel_tex_primitive:nnn { aftergroup } { after_group } { 0 }
831 \_unravel_tex_primitive:nnn { penalty } { break_penalty } { 0 }

```

```

832 \_unravel_tex_primitive:nnn { indent           } { start_par } { 1 }
833 \_unravel_tex_primitive:nnn { noindent        } { start_par } { 0 }
834 \_unravel_tex_primitive:nnn { quitvmode      } { start_par } { 2 }
835 \_unravel_tex_primitive:nnn { /             } { ital_corr  } { 0 }
836 \_unravel_tex_primitive:nnn { accent        } { accent     } { 0 }
837 \_unravel_tex_primitive:nnn { mathaccent    } { math_accnt } { 0 }
838 \sys_if_engine_luatex:T
839 {
840   \_unravel_tex_primitive:nnn
841     { explicitdiscretionary } { discretionary } { 1 }
842 }
843 \_unravel_tex_primitive:nnn { -             } { discretionary } { 1 }
844 \_unravel_tex_primitive:nnn { discretionary } { discretionary } { 0 }
845 \_unravel_tex_primitive:nnn { eqno           } { eq_no        } { 0 }
846 \_unravel_tex_primitive:nnn { leqno        } { eq_no        } { 1 }
847 \_unravel_tex_primitive:nnn { left           } { left_right   } { 30 }
848 \_unravel_tex_primitive:nnn { right          } { left_right   } { 31 }
849 \_unravel_tex_primitive:nnn { middle         } { left_right   } { 17 }
850 \_unravel_tex_primitive:nnn { mathord        } { math_comp    } { 16 }
851 \_unravel_tex_primitive:nnn { mathop         } { math_comp    } { 17 }
852 \_unravel_tex_primitive:nnn { mathbin        } { math_comp    } { 18 }
853 \_unravel_tex_primitive:nnn { mathrel        } { math_comp    } { 19 }
854 \_unravel_tex_primitive:nnn { mathopen       } { math_comp    } { 20 }
855 \_unravel_tex_primitive:nnn { mathclose     } { math_comp    } { 21 }
856 \_unravel_tex_primitive:nnn { mathpunct     } { math_comp    } { 22 }
857 \_unravel_tex_primitive:nnn { mathinner     } { math_comp    } { 23 }
858 \_unravel_tex_primitive:nnn { underline     } { math_comp    } { 26 }
859 \_unravel_tex_primitive:nnn { overline      } { math_comp    } { 27 }
860 \_unravel_tex_primitive:nnn { displaylimits } { limit_switch } { 0 }
861 \_unravel_tex_primitive:nnn { limits        } { limit_switch } { 1 }
862 \_unravel_tex_primitive:nnn { nolimits      } { limit_switch } { 2 }
863 \_unravel_tex_primitive:nnn { above         } { above        } { 0 }
864 \_unravel_tex_primitive:nnn { over          } { above        } { 1 }
865 \_unravel_tex_primitive:nnn { atop          } { above        } { 2 }
866 \_unravel_tex_primitive:nnn { abovewithdelims } { above        } { 3 }
867 \_unravel_tex_primitive:nnn { overwithdelims } { above        } { 4 }
868 \_unravel_tex_primitive:nnn { atopwithdelims } { above        } { 5 }
869 \_unravel_tex_primitive:nnn { displaystyle  } { math_style   } { 0 }
870 \_unravel_tex_primitive:nnn { textstyle     } { math_style   } { 2 }
871 \_unravel_tex_primitive:nnn { scriptstyle   } { math_style   } { 4 }
872 \_unravel_tex_primitive:nnn { scriptscriptstyle } { math_style   } { 6 }
873 \_unravel_tex_primitive:nnn { mathchoice    } { math_choice  } { 0 }
874 \_unravel_tex_primitive:nnn { nonscript     } { non_script   } { 0 }
875 \_unravel_tex_primitive:nnn { vcenter       } { vcenter      } { 0 }
876 \_unravel_tex_primitive:nnn { lowercase     } { case_shift   } { 256 }
877 \_unravel_tex_primitive:nnn { uppercase     } { case_shift   } { 512 }
878 \_unravel_tex_primitive:nnn { message       } { message      } { 0 }
879 \_unravel_tex_primitive:nnn { errmessage    } { message      } { 1 }
880 \_unravel_tex_primitive:nnn { openout       } { extension    } { 0 }
881 \_unravel_tex_primitive:nnn { write         } { extension    } { 1 }
882 \_unravel_tex_primitive:nnn { closeout      } { extension    } { 2 }
883 \_unravel_tex_primitive:nnn { special       } { extension    } { 3 }
884 \_unravel_tex_primitive:nnn { immediate     } { extension    } { 4 }
885 \_unravel_tex_primitive:nnn { setlanguage   } { extension    } { 5 }

```

```

886 \_unravel_tex_primitive_pdf:nnn { literal      } { extension } { 6 }
887 \_unravel_tex_primitive_pdf:nnn { obj          } { extension } { 7 }
888 \_unravel_tex_primitive_pdf:nnn { refobj     } { extension } { 8 }
889 \_unravel_tex_primitive_pdf:nnn { xform      } { extension } { 9 }
890 \_unravel_tex_primitive_pdf:nnn { refxform   } { extension } { 10 }
891 \_unravel_tex_primitive_pdf:nnn { ximage     } { extension } { 11 }
892 \_unravel_tex_primitive_pdf:nnn { refximage  } { extension } { 12 }
893 \_unravel_tex_primitive_pdf:nnn { annot      } { extension } { 13 }
894 \_unravel_tex_primitive_pdf:nnn { startlink  } { extension } { 14 }
895 \_unravel_tex_primitive_pdf:nnn { endlink    } { extension } { 15 }
896 \_unravel_tex_primitive_pdf:nnn { outline    } { extension } { 16 }
897 \_unravel_tex_primitive_pdf:nnn { dest       } { extension } { 17 }
898 \_unravel_tex_primitive_pdf:nnn { thread     } { extension } { 18 }
899 \_unravel_tex_primitive_pdf:nnn { startthread } { extension } { 19 }
900 \_unravel_tex_primitive_pdf:nnn { endthread  } { extension } { 20 }
901 \_unravel_tex_primitive_pdf:nnn { savepos    } { extension } { 21 }
902 \_unravel_tex_primitive_pdf:nnn { info       } { extension } { 22 }
903 \_unravel_tex_primitive_pdf:nnn { catalog    } { extension } { 23 }
904 \_unravel_tex_primitive_pdf:nnn { names      } { extension } { 24 }
905 \_unravel_tex_primitive_pdf:nnn { fontattr   } { extension } { 25 }
906 \_unravel_tex_primitive_pdf:nnn { includechars } { extension } { 26 }
907 \_unravel_tex_primitive_pdf:nnn { mapfile    } { extension } { 27 }
908 \_unravel_tex_primitive_pdf:nnn { mapline    } { extension } { 28 }
909 \_unravel_tex_primitive_pdf:nnn { trailer    } { extension } { 29 }
910 \_unravel_tex_primitive_pdf:nnn { resettimer } { extension } { 30 }
911 \_unravel_tex_primitive_pdf:nnn { fontexpand } { extension } { 31 }
912 \_unravel_tex_primitive_pdf:nnn { setrandomseed } { extension } { 32 }
913 \_unravel_tex_primitive_pdf:nnn { snaprefpoint } { extension } { 33 }
914 \_unravel_tex_primitive_pdf:nnn { snapy      } { extension } { 34 }
915 \_unravel_tex_primitive_pdf:nnn { snapycomp  } { extension } { 35 }
916 \_unravel_tex_primitive_pdf:nnn { glyphtounicode } { extension } { 36 }
917 \_unravel_tex_primitive_pdf:nnn { colorstack } { extension } { 37 }
918 \_unravel_tex_primitive_pdf:nnn { setmatrix  } { extension } { 38 }
919 \_unravel_tex_primitive_pdf:nnn { save       } { extension } { 39 }
920 \_unravel_tex_primitive_pdf:nnn { restore    } { extension } { 40 }
921 \_unravel_tex_primitive_pdf:nnn { nobuiltintounicode } { extension } { 41 }
922 \_unravel_tex_primitive:nnn { openin      } { in_stream } { 1 }
923 \_unravel_tex_primitive:nnn { closein     } { in_stream } { 0 }
924 \_unravel_tex_primitive:nnn { begingroup  } { begin_group } { 0 }
925 \_unravel_tex_primitive:nnn { endgroup    } { end_group } { 0 }
926 \_unravel_tex_primitive:nnn { omit       } { omit } { 0 }
927 \_unravel_tex_primitive:nnn { ~          } { ex_space } { 0 }
928 \_unravel_tex_primitive:nnn { noboundary  } { no_boundary } { 0 }
929 \_unravel_tex_primitive:nnn { radical     } { radical } { 0 }
930 \_unravel_tex_primitive:nnn { endcsname   } { end_cs_name } { 0 }
931 \_unravel_tex_primitive:nnn { lastpenalty } { last_item } { 0 }
932 \_unravel_tex_primitive:nnn { lastkern    } { last_item } { 1 }
933 \_unravel_tex_primitive:nnn { lastskip    } { last_item } { 2 }
934 \_unravel_tex_primitive:nnn { lastnodetype } { last_item } { 3 }
935 \_unravel_tex_primitive:nnn { inputlineno } { last_item } { 4 }
936 \_unravel_tex_primitive:nnn { badness     } { last_item } { 5 }
937 \_unravel_tex_primitive_pdf:nnn { texversion  } { last_item } { 6 }
938 \_unravel_tex_primitive_pdf:nnn { lastobj     } { last_item } { 7 }
939 \_unravel_tex_primitive_pdf:nnn { lastxform   } { last_item } { 8 }

```



```

940 \_unravel_tex_primitive_pdf:nnn { lastximage      } { last_item } { 9 }
941 \_unravel_tex_primitive_pdf:nnn { lastximagepages } { last_item } { 10 }
942 \_unravel_tex_primitive_pdf:nnn { lastannot      } { last_item } { 11 }
943 \_unravel_tex_primitive_pdf:nnn { lastxpos      } { last_item } { 12 }
944 \_unravel_tex_primitive_pdf:nnn { lasttypos    } { last_item } { 13 }
945 \_unravel_tex_primitive_pdf:nnn { retval       } { last_item } { 14 }
946 \_unravel_tex_primitive_pdf:nnn { lastximagecolordepth } { last_item } { 15 }
947 \_unravel_tex_primitive_pdf:nnn { elapsedtime  } { last_item } { 16 }
948 \_unravel_tex_primitive_pdf:nnn { shellescape  } { last_item } { 17 }
949 \_unravel_tex_primitive_pdf:nnn { randomseed   } { last_item } { 18 }
950 \_unravel_tex_primitive_pdf:nnn { lastlink     } { last_item } { 19 }
951 \_unravel_tex_primitive:nnn { eTeXversion  } { last_item } { 20 }
952 \_unravel_tex_primitive:nnn { currentgrouplevel } { last_item } { 21 }
953 \_unravel_tex_primitive:nnn { currentgroupptype } { last_item } { 22 }
954 \_unravel_tex_primitive:nnn { currentiflevel   } { last_item } { 23 }
955 \_unravel_tex_primitive:nnn { currentiftypetype } { last_item } { 24 }
956 \_unravel_tex_primitive:nnn { currentifbranch  } { last_item } { 25 }
957 \_unravel_tex_primitive:nnn { gluestretchorder } { last_item } { 26 }
958 \_unravel_tex_primitive:nnn { glueshrinkorder  } { last_item } { 27 }
959 \_unravel_tex_primitive:nnn { fontcharwd      } { last_item } { 28 }
960 \_unravel_tex_primitive:nnn { fontcharht     } { last_item } { 29 }
961 \_unravel_tex_primitive:nnn { fontchardp     } { last_item } { 30 }
962 \_unravel_tex_primitive:nnn { fontcharic     } { last_item } { 31 }
963 \_unravel_tex_primitive:nnn { parshapelength  } { last_item } { 32 }
964 \_unravel_tex_primitive:nnn { parshapeindent  } { last_item } { 33 }
965 \_unravel_tex_primitive:nnn { parshapedimen   } { last_item } { 34 }
966 \_unravel_tex_primitive:nnn { gluestretch     } { last_item } { 35 }
967 \_unravel_tex_primitive:nnn { glueshrink      } { last_item } { 36 }
968 \_unravel_tex_primitive:nnn { mutoglupe      } { last_item } { 37 }
969 \_unravel_tex_primitive:nnn { gluetoemu      } { last_item } { 38 }
970 \_unravel_tex_primitive:nnn { numexpr        } { last_item } { 39 }
971 \_unravel_tex_primitive:nnn { dimexpr        } { last_item } { 40 }
972 \_unravel_tex_primitive:nnn { glueexpr       } { last_item } { 41 }
973 \_unravel_tex_primitive:nnn { muexpr        } { last_item } { 42 }
974 \_unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
975 \_unravel_tex_primitive:nnn { output         } { assign_toks } { 1 }
976 \_unravel_tex_primitive:nnn { everypar       } { assign_toks } { 2 }
977 \_unravel_tex_primitive:nnn { everymath      } { assign_toks } { 3 }
978 \_unravel_tex_primitive:nnn { everydisplay   } { assign_toks } { 4 }
979 \_unravel_tex_primitive:nnn { everyhbox     } { assign_toks } { 5 }
980 \_unravel_tex_primitive:nnn { everyvbox     } { assign_toks } { 6 }
981 \_unravel_tex_primitive:nnn { everyjob      } { assign_toks } { 7 }
982 \_unravel_tex_primitive:nnn { everycr       } { assign_toks } { 8 }
983 \_unravel_tex_primitive:nnn { errhelp       } { assign_toks } { 9 }
984 \_unravel_tex_primitive_pdf:nnn { pagesattr     } { assign_toks } { 10 }
985 \_unravel_tex_primitive_pdf:nnn { pageattr      } { assign_toks } { 11 }
986 \_unravel_tex_primitive_pdf:nnn { pageresources } { assign_toks } { 12 }
987 \_unravel_tex_primitive_pdf:nnn { pkmode        } { assign_toks } { 13 }
988 \_unravel_tex_primitive:nnn { everyeof      } { assign_toks } { 14 }
989 \_unravel_tex_primitive:nnn { pretolerance   } { assign_int } { 0 }
990 \_unravel_tex_primitive:nnn { tolerance     } { assign_int } { 1 }
991 \_unravel_tex_primitive:nnn { linepenalty   } { assign_int } { 2 }
992 \_unravel_tex_primitive:nnn { hyphenpenalty } { assign_int } { 3 }
993 \_unravel_tex_primitive:nnn { exhyphenpenalty } { assign_int } { 4 }

```

```

994 \_unravel_tex_primitive:nnn { clubpenalty } { assign_int } { 5 }
995 \_unravel_tex_primitive:nnn { widowpenalty } { assign_int } { 6 }
996 \_unravel_tex_primitive:nnn { displaywidowpenalty } { assign_int } { 7 }
997 \_unravel_tex_primitive:nnn { brokenpenalty } { assign_int } { 8 }
998 \_unravel_tex_primitive:nnn { binoppenalty } { assign_int } { 9 }
999 \_unravel_tex_primitive:nnn { relpenalty } { assign_int } { 10 }
1000 \_unravel_tex_primitive:nnn { predisdisplaypenalty } { assign_int } { 11 }
1001 \_unravel_tex_primitive:nnn { postdisplaypenalty } { assign_int } { 12 }
1002 \_unravel_tex_primitive:nnn { interlinepenalty } { assign_int } { 13 }
1003 \_unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int } { 14 }
1004 \_unravel_tex_primitive:nnn { finalhyphendemerits } { assign_int } { 15 }
1005 \_unravel_tex_primitive:nnn { adjdemerits } { assign_int } { 16 }
1006 \_unravel_tex_primitive:nnn { mag } { assign_int } { 17 }
1007 \_unravel_tex_primitive:nnn { delimitfactor } { assign_int } { 18 }
1008 \_unravel_tex_primitive:nnn { looseness } { assign_int } { 19 }
1009 \_unravel_tex_primitive:nnn { time } { assign_int } { 20 }
1010 \_unravel_tex_primitive:nnn { day } { assign_int } { 21 }
1011 \_unravel_tex_primitive:nnn { month } { assign_int } { 22 }
1012 \_unravel_tex_primitive:nnn { year } { assign_int } { 23 }
1013 \_unravel_tex_primitive:nnn { showboxbreadth } { assign_int } { 24 }
1014 \_unravel_tex_primitive:nnn { showboxdepth } { assign_int } { 25 }
1015 \_unravel_tex_primitive:nnn { hbadness } { assign_int } { 26 }
1016 \_unravel_tex_primitive:nnn { vbadness } { assign_int } { 27 }
1017 \_unravel_tex_primitive:nnn { pausing } { assign_int } { 28 }
1018 \_unravel_tex_primitive:nnn { tracingonline } { assign_int } { 29 }
1019 \_unravel_tex_primitive:nnn { tracingmacros } { assign_int } { 30 }
1020 \_unravel_tex_primitive:nnn { tracingstats } { assign_int } { 31 }
1021 \_unravel_tex_primitive:nnn { tracingparagraphs } { assign_int } { 32 }
1022 \_unravel_tex_primitive:nnn { tracingpages } { assign_int } { 33 }
1023 \_unravel_tex_primitive:nnn { tracingoutput } { assign_int } { 34 }
1024 \_unravel_tex_primitive:nnn { tracinglostchars } { assign_int } { 35 }
1025 \_unravel_tex_primitive:nnn { tracingcommands } { assign_int } { 36 }
1026 \_unravel_tex_primitive:nnn { tracingrestores } { assign_int } { 37 }
1027 \_unravel_tex_primitive:nnn { uchyp } { assign_int } { 38 }
1028 \_unravel_tex_primitive:nnn { outputpenalty } { assign_int } { 39 }
1029 \_unravel_tex_primitive:nnn { maxdeadcycles } { assign_int } { 40 }
1030 \_unravel_tex_primitive:nnn { hangafter } { assign_int } { 41 }
1031 \_unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
1032 \_unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
1033 \_unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
1034 \_unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
1035 \_unravel_tex_primitive:nnn { defaultthyphenchar } { assign_int } { 46 }
1036 \_unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
1037 \_unravel_tex_primitive:nnn { endliness } { assign_int } { 48 }
1038 \_unravel_tex_primitive:nnn { newlinechar } { assign_int } { 49 }
1039 \_unravel_tex_primitive:nnn { language } { assign_int } { 50 }
1040 \_unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
1041 \_unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
1042 \_unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
1043 \_unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
1044 \_unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
1045 \_unravel_tex_primitive_pdf:nnn { compresslevel } { assign_int } { 56 }
1046 \_unravel_tex_primitive_pdf:nnn { decimaldigits } { assign_int } { 57 }
1047 \_unravel_tex_primitive_pdf:nnn { movechars } { assign_int } { 58 }

```

```

1048 \_unravel_tex_primitive_pdf:nnn { imageresolution } { assign_int } { 59 }
1049 \_unravel_tex_primitive_pdf:nnn { pkresolution } { assign_int } { 60 }
1050 \_unravel_tex_primitive_pdf:nnn { uniqueresname } { assign_int } { 61 }
1051 \_unravel_tex_primitive_pdf:nnn
1052 { optionalwaysusepdfpagebox } { assign_int } { 62 }
1053 \_unravel_tex_primitive_pdf:nnn
1054 { optionpdfinclusionerrorlevel } { assign_int } { 63 }
1055 \_unravel_tex_primitive_pdf:nnn
1056 { optionpdfminorversion } { assign_int } { 64 }
1057 \_unravel_tex_primitive_pdf:nnn { minorversion } { assign_int } { 64 }
1058 \_unravel_tex_primitive_pdf:nnn { forcepagebox } { assign_int } { 65 }
1059 \_unravel_tex_primitive_pdf:nnn { pagebox } { assign_int } { 66 }
1060 \_unravel_tex_primitive_pdf:nnn
1061 { inclusionerrorlevel } { assign_int } { 67 }
1062 \_unravel_tex_primitive_pdf:nnn { gamma } { assign_int } { 68 }
1063 \_unravel_tex_primitive_pdf:nnn { imagegamma } { assign_int } { 69 }
1064 \_unravel_tex_primitive_pdf:nnn { imagehicolor } { assign_int } { 70 }
1065 \_unravel_tex_primitive_pdf:nnn { imageapplygamma } { assign_int } { 71 }
1066 \_unravel_tex_primitive_pdf:nnn { adjustspacing } { assign_int } { 72 }
1067 \_unravel_tex_primitive_pdf:nnn { protrudechars } { assign_int } { 73 }
1068 \_unravel_tex_primitive_pdf:nnn { tracingfonts } { assign_int } { 74 }
1069 \_unravel_tex_primitive_pdf:nnn { objcompresslevel } { assign_int } { 75 }
1070 \_unravel_tex_primitive_pdf:nnn
1071 { adjustinterwordglue } { assign_int } { 76 }
1072 \_unravel_tex_primitive_pdf:nnn { prependkern } { assign_int } { 77 }
1073 \_unravel_tex_primitive_pdf:nnn { appendkern } { assign_int } { 78 }
1074 \_unravel_tex_primitive_pdf:nnn { gentounicode } { assign_int } { 79 }
1075 \_unravel_tex_primitive_pdf:nnn { draftmode } { assign_int } { 80 }
1076 \_unravel_tex_primitive_pdf:nnn { inclusioncopyfonts } { assign_int } { 81 }
1077 \_unravel_tex_primitive:nnn { tracingassigns } { assign_int } { 82 }
1078 \_unravel_tex_primitive:nnn { tracinggroups } { assign_int } { 83 }
1079 \_unravel_tex_primitive:nnn { tracingifs } { assign_int } { 84 }
1080 \_unravel_tex_primitive:nnn { tracingscantokens } { assign_int } { 85 }
1081 \_unravel_tex_primitive:nnn { tracingnesting } { assign_int } { 86 }
1082 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
1083 \_unravel_tex_primitive:nnn { lastlinefit } { assign_int } { 88 }
1084 \_unravel_tex_primitive:nnn { savingvdiscards } { assign_int } { 89 }
1085 \_unravel_tex_primitive:nnn { savinghyphcodes } { assign_int } { 90 }
1086 \_unravel_tex_primitive:nnn { TeXeTstate } { assign_int } { 91 }
1087 \_unravel_tex_primitive:nnn { parindent } { assign_dimen } { 0 }
1088 \_unravel_tex_primitive:nnn { mathsurround } { assign_dimen } { 1 }
1089 \_unravel_tex_primitive:nnn { lineskiplimit } { assign_dimen } { 2 }
1090 \_unravel_tex_primitive:nnn { hsize } { assign_dimen } { 3 }
1091 \_unravel_tex_primitive:nnn { vsize } { assign_dimen } { 4 }
1092 \_unravel_tex_primitive:nnn { maxdepth } { assign_dimen } { 5 }
1093 \_unravel_tex_primitive:nnn { splitmaxdepth } { assign_dimen } { 6 }
1094 \_unravel_tex_primitive:nnn { boxmaxdepth } { assign_dimen } { 7 }
1095 \_unravel_tex_primitive:nnn { hfuzz } { assign_dimen } { 8 }
1096 \_unravel_tex_primitive:nnn { vfuzz } { assign_dimen } { 9 }
1097 \_unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
1098 \_unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
1099 \_unravel_tex_primitive:nnn { scriptspace } { assign_dimen } { 12 }
1100 \_unravel_tex_primitive:nnn { predisplaysize } { assign_dimen } { 13 }
1101 \_unravel_tex_primitive:nnn { displaywidth } { assign_dimen } { 14 }

```

```

1102 \_unravel_tex_primitive:nnn { displayindent } { assign_dimen } { 15 }
1103 \_unravel_tex_primitive:nnn { overfullrule } { assign_dimen } { 16 }
1104 \_unravel_tex_primitive:nnn { hangindent } { assign_dimen } { 17 }
1105 \_unravel_tex_primitive:nnn { hoffset } { assign_dimen } { 18 }
1106 \_unravel_tex_primitive:nnn { voffset } { assign_dimen } { 19 }
1107 \_unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
1108 \_unravel_tex_primitive_pdf:nnn { horigin } { assign_dimen } { 21 }
1109 \_unravel_tex_primitive_pdf:nnn { vorigin } { assign_dimen } { 22 }
1110 \_unravel_tex_primitive_pdf:nnn { pagewidth } { assign_dimen } { 23 }
1111 \_unravel_tex_primitive_pdf:nnn { pageheight } { assign_dimen } { 24 }
1112 \_unravel_tex_primitive_pdf:nnn { linkmargin } { assign_dimen } { 25 }
1113 \_unravel_tex_primitive_pdf:nnn { destmargin } { assign_dimen } { 26 }
1114 \_unravel_tex_primitive_pdf:nnn { threadmargin } { assign_dimen } { 27 }
1115 \_unravel_tex_primitive_pdf:nnn { firstlineheight } { assign_dimen } { 28 }
1116 \_unravel_tex_primitive_pdf:nnn { lastlinedepth } { assign_dimen } { 29 }
1117 \_unravel_tex_primitive_pdf:nnn { eachlineheight } { assign_dimen } { 30 }
1118 \_unravel_tex_primitive_pdf:nnn { eachlinedepth } { assign_dimen } { 31 }
1119 \_unravel_tex_primitive_pdf:nnn { ignoreddimen } { assign_dimen } { 32 }
1120 \_unravel_tex_primitive_pdf:nnn { pxdimen } { assign_dimen } { 33 }
1121 \_unravel_tex_primitive:nnn { lineskip } { assign_glue } { 0 }
1122 \_unravel_tex_primitive:nnn { baselineskip } { assign_glue } { 1 }
1123 \_unravel_tex_primitive:nnn { parskip } { assign_glue } { 2 }
1124 \_unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue } { 3 }
1125 \_unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue } { 4 }
1126 \_unravel_tex_primitive:nnn { abovedisplaysshortskip } { assign_glue } { 5 }
1127 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1128 \_unravel_tex_primitive:nnn { leftskip } { assign_glue } { 7 }
1129 \_unravel_tex_primitive:nnn { rightskip } { assign_glue } { 8 }
1130 \_unravel_tex_primitive:nnn { topskip } { assign_glue } { 9 }
1131 \_unravel_tex_primitive:nnn { splittopskip } { assign_glue } { 10 }
1132 \_unravel_tex_primitive:nnn { tabskip } { assign_glue } { 11 }
1133 \_unravel_tex_primitive:nnn { spaceskip } { assign_glue } { 12 }
1134 \_unravel_tex_primitive:nnn { xspaceskip } { assign_glue } { 13 }
1135 \_unravel_tex_primitive:nnn { parfillskip } { assign_glue } { 14 }
1136 \_unravel_tex_primitive:nnn { thinmuskip } { assign_mu_glue } { 15 }
1137 \_unravel_tex_primitive:nnn { medmuskip } { assign_mu_glue } { 16 }
1138 \_unravel_tex_primitive:nnn { thickmuskip } { assign_mu_glue } { 17 }
1139 \_unravel_tex_primitive:nnn { fontdimen } { assign_font_dimen } { 0 }
1140 \_unravel_tex_primitive:nnn { hyphenchar } { assign_font_int } { 0 }
1141 \_unravel_tex_primitive:nnn { skewchar } { assign_font_int } { 1 }
1142 \_unravel_tex_primitive:nnn { lpcode } { assign_font_int } { 2 }
1143 \_unravel_tex_primitive:nnn { rpcode } { assign_font_int } { 3 }
1144 \_unravel_tex_primitive:nnn { efcodes } { assign_font_int } { 4 }
1145 \_unravel_tex_primitive:nnn { tagcode } { assign_font_int } { 5 }
1146 \_unravel_tex_primitive_pdf:nnn { noligatures } { assign_font_int } { 6 }
1147 \_unravel_tex_primitive:nnn { knbscode } { assign_font_int } { 7 }
1148 \_unravel_tex_primitive:nnn { stbscode } { assign_font_int } { 8 }
1149 \_unravel_tex_primitive:nnn { shbscode } { assign_font_int } { 9 }
1150 \_unravel_tex_primitive:nnn { knbccode } { assign_font_int } { 10 }
1151 \_unravel_tex_primitive:nnn { knaccodes } { assign_font_int } { 11 }
1152 \_unravel_tex_primitive:nnn { spacefactor } { set_aux } { 102 }
1153 \_unravel_tex_primitive:nnn { prevdepth } { set_aux } { 1 }
1154 \_unravel_tex_primitive:nnn { prevgraf } { set_prev_graf } { 0 }
1155 \_unravel_tex_primitive:nnn { pagegoal } { set_page_dimen } { 0 }

```

```

1156 \_unravel_tex_primitive:nnn { pagetotal      } { set_page_dimen } { 1 }
1157 \_unravel_tex_primitive:nnn { pagestretch  } { set_page_dimen } { 2 }
1158 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 3 }
1159 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 4 }
1160 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 5 }
1161 \_unravel_tex_primitive:nnn { pageshrink     } { set_page_dimen } { 6 }
1162 \_unravel_tex_primitive:nnn { pagedepth     } { set_page_dimen } { 7 }
1163 \_unravel_tex_primitive:nnn { deadcycles    } { set_page_int   } { 0 }
1164 \_unravel_tex_primitive:nnn { insertpenalties } { set_page_int   } { 1 }
1165 \_unravel_tex_primitive:nnn { interactionmode } { set_page_int   } { 2 }
1166 \_unravel_tex_primitive:nnn { wd            } { set_box_dimen  } { 1 }
1167 \_unravel_tex_primitive:nnn { dp            } { set_box_dimen  } { 2 }
1168 \_unravel_tex_primitive:nnn { ht            } { set_box_dimen  } { 3 }
1169 \_unravel_tex_primitive:nnn { parshape      } { set_shape      } { 0 }
1170 \_unravel_tex_primitive:nnn { interlinepenalties } { set_shape      } { 1 }
1171 \_unravel_tex_primitive:nnn { clubpenalties } { set_shape      } { 2 }
1172 \_unravel_tex_primitive:nnn { widowpenalties } { set_shape      } { 3 }
1173 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape      } { 4 }
1174 \_unravel_tex_primitive:nnn { catcode       } { def_code       } { 0 }
1175 \_unravel_tex_primitive:nnn { lccode        } { def_code       } { 256 }
1176 \_unravel_tex_primitive:nnn { uccode        } { def_code       } { 512 }
1177 \_unravel_tex_primitive:nnn { sfcode        } { def_code       } { 768 }
1178 \_unravel_tex_primitive:nnn { mathcode      } { def_code       } { 1024 }
1179 \_unravel_tex_primitive:nnn { delcode       } { def_code       } { 1591 }
1180 \_unravel_tex_primitive:nnn { textfont      } { def_family     } { -48 }
1181 \_unravel_tex_primitive:nnn { scriptfont    } { def_family     } { -32 }
1182 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family     } { -16 }
1183 \_unravel_tex_primitive:nnn { nullfont      } { set_font       } { 0 }
1184 \_unravel_tex_primitive:nnn { font          } { def_font       } { 0 }
1185 \_unravel_tex_primitive:nnn { count         } { register       } { 1 000 000 }
1186 \_unravel_tex_primitive:nnn { dimen         } { register       } { 2 000 000 }
1187 \_unravel_tex_primitive:nnn { skip          } { register       } { 3 000 000 }
1188 \_unravel_tex_primitive:nnn { muskip        } { register       } { 4 000 000 }
1189 \_unravel_tex_primitive:nnn { advance       } { advance        } { 0 }
1190 \_unravel_tex_primitive:nnn { multiply      } { multiply       } { 0 }
1191 \_unravel_tex_primitive:nnn { divide        } { divide         } { 0 }
1192 \_unravel_tex_primitive:nnn { long          } { prefix         } { 1 }
1193 \_unravel_tex_primitive:nnn { outer         } { prefix         } { 2 }
1194 \_unravel_tex_primitive:nnn { global        } { prefix         } { 4 }
1195 \_unravel_tex_primitive:nnn { protected     } { prefix         } { 8 }
1196 \_unravel_tex_primitive:nnn { let           } { let            } { 0 }
1197 \_unravel_tex_primitive:nnn { futurelet     } { let            } { 1 }
1198 \_unravel_tex_primitive:nnn { chardef       } { shorthand_def  } { 0 }
1199 \_unravel_tex_primitive:nnn { mathchardef   } { shorthand_def  } { 1 }
1200 \_unravel_tex_primitive:nnn { countdef      } { shorthand_def  } { 2 }
1201 \_unravel_tex_primitive:nnn { dimendef      } { shorthand_def  } { 3 }
1202 \_unravel_tex_primitive:nnn { skipdef       } { shorthand_def  } { 4 }
1203 \_unravel_tex_primitive:nnn { muskipdef     } { shorthand_def  } { 5 }
1204 \_unravel_tex_primitive:nnn { toksdef       } { shorthand_def  } { 6 }
1205 \_unravel_tex_primitive:nnn { read          } { read_to_cs     } { 0 }
1206 \_unravel_tex_primitive:nnn { readline      } { read_to_cs     } { 1 }
1207 \_unravel_tex_primitive:nnn { def           } { def            } { 0 }
1208 \_unravel_tex_primitive:nnn { gdef         } { def            } { 1 }
1209 \_unravel_tex_primitive:nnn { edef         } { def            } { 2 }

```

```

1210 \_unravel_tex_primitive:nnn { xdef } { def } { 3 }
1211 \_unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1212 \_unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1213 \_unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1214 \_unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1215 \_unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1216 \_unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1217 \_unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1218 \_unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1219 \_unravel_tex_primitive_pdf:nnn { copyfont } { pdf_copy_font } { 0 }
1220 \_unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1221 \_unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1222 \_unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1223 \_unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1224 \_unravel_tex_primitive_pdf:nnn { primitive } { no_expand } { 1 }
1225 \_unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1226 \_unravel_tex_primitive:nnn { input } { input } { 0 }
1227 \_unravel_tex_primitive:nnn { endinput } { input } { 1 }
1228 \_unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1229 \_unravel_tex_primitive:nnn { if } { if_test } { 0 }
1230 \_unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }
1231 \_unravel_tex_primitive:nnn { ifnum } { if_test } { 2 }
1232 \_unravel_tex_primitive:nnn { ifdim } { if_test } { 3 }
1233 \_unravel_tex_primitive:nnn { ifodd } { if_test } { 4 }
1234 \_unravel_tex_primitive:nnn { ifvmode } { if_test } { 5 }
1235 \_unravel_tex_primitive:nnn { ifhmode } { if_test } { 5 }
1236 \_unravel_tex_primitive:nnn { ifmmode } { if_test } { 5 }
1237 \_unravel_tex_primitive:nnn { ifinner } { if_test } { 5 }
1238 \_unravel_tex_primitive:nnn { ifvoid } { if_test } { 9 }
1239 \_unravel_tex_primitive:nnn { ifhbox } { if_test } { 9 }
1240 \_unravel_tex_primitive:nnn { ifvbox } { if_test } { 9 }
1241 \_unravel_tex_primitive:nnn { ifx } { if_test } { 12 }
1242 \_unravel_tex_primitive:nnn { ifeof } { if_test } { 13 }
1243 \_unravel_tex_primitive:nnn { iftrue } { if_test } { 14 }
1244 \_unravel_tex_primitive:nnn { iffalse } { if_test } { 15 }
1245 \_unravel_tex_primitive:nnn { ifcase } { if_test } { 16 }
1246 \_unravel_tex_primitive:nnn { ifdefined } { if_test } { 17 }
1247 \_unravel_tex_primitive:nnn { ifcsname } { if_test } { 18 }
1248 \_unravel_tex_primitive:nnn { iffontchar } { if_test } { 19 }
1249 \_unravel_tex_primitive:nnn { ifincsname } { if_test } { 20 }
1250 \_unravel_tex_primitive:nnn { ifprimitive } { if_test } { 21 }
1251 \_unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1252 \_unravel_tex_primitive:nnn { ifabsnum } { if_test } { 22 }
1253 \_unravel_tex_primitive:nnn { ifpdfabsnum } { if_test } { 22 }
1254 \_unravel_tex_primitive:nnn { ifabsdim } { if_test } { 23 }
1255 \_unravel_tex_primitive:nnn { ifpdfabsdim } { if_test } { 23 }
1256 \bool_if:nT { \sys_if_engine_ptex_p: || \sys_if_engine_uptex_p: }
1257 {
1258   \_unravel_tex_primitive:nnn { iftdir } { if_test } { 5 }
1259   \_unravel_tex_primitive:nnn { ifydir } { if_test } { 5 }
1260   \_unravel_tex_primitive:nnn { ifddir } { if_test } { 5 }
1261   \_unravel_tex_primitive:nnn { ifmdir } { if_test } { 5 }
1262   \_unravel_tex_primitive:nnn { iftbox } { if_test } { 9 }
1263   \_unravel_tex_primitive:nnn { ifybox } { if_test } { 9 }

```

```

1264     \_unravel_tex_primitive:nnn { ifdbox           } { if_test } { 9 }
1265     \_unravel_tex_primitive:nnn { ifmbox           } { if_test } { 9 }
1266     \_unravel_tex_primitive:nnn { ifjfont          } { if_test } { 24 }
1267     \_unravel_tex_primitive:nnn { iftfont          } { if_test } { 24 }
1268   }
1269   \_unravel_tex_primitive:nnn { fi                 } { fi_or_else } { 2 }
1270   \_unravel_tex_primitive:nnn { else               } { fi_or_else } { 3 }
1271   \_unravel_tex_primitive:nnn { or                 } { fi_or_else } { 4 }
1272   \_unravel_tex_primitive:nnn { csname             } { cs_name } { 0 }
1273   \_unravel_tex_primitive:nnn { lastnamedcs        } { cs_name } { 1 }
1274   \_unravel_tex_primitive:nnn { number             } { convert } { 0 }
1275   \_unravel_tex_primitive:nnn { romannumeral        } { convert } { 1 }
1276   \_unravel_tex_primitive:nnn { string             } { convert } { 2 }
1277   \_unravel_tex_primitive:nnn { meaning            } { convert } { 3 }
1278   \_unravel_tex_primitive:nnn { fontname           } { convert } { 4 }
1279   \_unravel_tex_primitive:nnn { eTeXrevision        } { convert } { 5 }
1280   \_unravel_tex_primitive_pdf:nnn { texrevision     } { convert } { 6 }
1281   \_unravel_tex_primitive_pdf:nnn { texbanner       } { convert } { 7 }
1282   \_unravel_tex_primitive:nnn { pdffontname         } { convert } { 8 }
1283   \_unravel_tex_primitive_pdf:nnn { fontobjnum      } { convert } { 9 }
1284   \_unravel_tex_primitive_pdf:nnn { fontsize        } { convert } { 10 }
1285   \_unravel_tex_primitive_pdf:nnn { pageref         } { convert } { 11 }
1286   \_unravel_tex_primitive_pdf:nnn { xformname       } { convert } { 12 }
1287   \_unravel_tex_primitive_pdf:nnn { escapestring    } { convert } { 13 }
1288   \_unravel_tex_primitive_pdf:nnn { escapename      } { convert } { 14 }
1289   \_unravel_tex_primitive:nnn { leftmarginkern      } { convert } { 15 }
1290   \_unravel_tex_primitive:nnn { rightmarginkern     } { convert } { 16 }
1291   \_unravel_tex_primitive_pdf:nnn { strcmp          } { convert } { 17 }
1292   \_unravel_tex_primitive_pdf:nnn { colorstackinit  } { convert } { 18 }
1293   \_unravel_tex_primitive_pdf:nnn { escapehex       } { convert } { 19 }
1294   \_unravel_tex_primitive_pdf:nnn { unescapehex     } { convert } { 20 }
1295   \_unravel_tex_primitive_pdf:nnn { creationdate    } { convert } { 21 }
1296   \_unravel_tex_primitive_pdf:nnn { filemoddate     } { convert } { 22 }
1297   \_unravel_tex_primitive_pdf:nnn { filesize        } { convert } { 23 }
1298   \_unravel_tex_primitive_pdf:nnn { mdfivesum      } { convert } { 24 }
1299   \_unravel_tex_primitive_pdf:nnn { filedump        } { convert } { 25 }
1300   \_unravel_tex_primitive_pdf:nnn { match           } { convert } { 26 }
1301   \_unravel_tex_primitive_pdf:nnn { lastmatch       } { convert } { 27 }
1302   \_unravel_tex_primitive_pdf:nnn { uniformdeviate  } { convert } { 28 }
1303   \_unravel_tex_primitive_pdf:nnn { normaldeviate   } { convert } { 29 }
1304   \_unravel_tex_primitive_pdf:nnn { insertht        } { convert } { 30 }
1305   \_unravel_tex_primitive_pdf:nnn { ximagebbox      } { convert } { 31 }
1306   \_unravel_tex_primitive:nnn { jobname             } { convert } { 32 }
1307   \sys_if_engine luatex:T
1308   { \_unravel_tex_primitive:nnn { directlua         } { convert } { 33 } }
1309   \_unravel_tex_primitive:nnn { expanded            } { convert } { 34 }
1310   \sys_if_engine luatex:T
1311   { \_unravel_tex_primitive:nnn { luaescapestring   } { convert } { 35 } }
1312   \bool_if:nT { \sys_if_engine xetex_p: || \sys_if_engine ptex_p: || \sys_if_engine uptex_p: }
1313   {
1314     \_unravel_tex_primitive:nnn { Ucharcat          } { convert } { 40 }
1315   }
1316   \_unravel_tex_primitive:nnn { the                 } { the } { 0 }
1317   \_unravel_tex_primitive:nnn { unexpanded          } { the } { 1 }

```

```

1318 \__unravel_tex_primitive:nnn { detokenize          } { the } { 5 }
1319 \__unravel_tex_primitive:nnn { topmark            } { top_bot_mark } { 0 }
1320 \__unravel_tex_primitive:nnn { firstmark          } { top_bot_mark } { 1 }
1321 \__unravel_tex_primitive:nnn { botmark            } { top_bot_mark } { 2 }
1322 \__unravel_tex_primitive:nnn { splitfirstmark     } { top_bot_mark } { 3 }
1323 \__unravel_tex_primitive:nnn { splitbotmark       } { top_bot_mark } { 4 }
1324 \__unravel_tex_primitive:nnn { topmarks           } { top_bot_mark } { 5 }
1325 \__unravel_tex_primitive:nnn { firstmarks        } { top_bot_mark } { 6 }
1326 \__unravel_tex_primitive:nnn { botmarks          } { top_bot_mark } { 7 }
1327 \__unravel_tex_primitive:nnn { splitfirstmarks   } { top_bot_mark } { 8 }
1328 \__unravel_tex_primitive:nnn { splitbotmarks     } { top_bot_mark } { 9 }

```

## 2.4 Get next token

We define here two functions which fetch the next token in the token list.

- `\__unravel_get_next`: sets `\l__unravel_head_gtl`, `\l__unravel_head_token`, and if possible `\l__unravel_head_tl` (otherwise it is cleared).
- `\__unravel_get_token`: additionally sets `\l__unravel_head_cmd_int` and `\l__unravel_head_char_int`.

The latter is based on `\__unravel_set_cmd`: which derives the `\l__unravel_head_cmd_int` and `\l__unravel_head_char_int` from `\l__unravel_head_token`.

```

\__unravel_get_next:
\__unravel_get_next_aux:w

```

If the input is empty, insert a frozen `\relax` (the alternative would be either to grab a token in the input stream after `\unravel`, which is tough, or simply produce an error and exit; perhaps this should be configurable). Then remove the first token in the input, and store it in `\l__unravel_head_gtl`. Set `\l__unravel_head_token` equal in meaning to that first token. Then set `\l__unravel_head_tl` to contain the token, unless it is a begin-group or end-group character, in which case this token list is emptied.

```

1329 \cs_new_protected:Npn \__unravel_get_next:
1330   {
1331     \__unravel_input_if_empty:TF
1332     {
1333       \__unravel_error:nnnnn { runaway-unravel } { } { } { } { }
1334       \__unravel_back_input_gtl:N \c__unravel_frozen_relax_gtl
1335     }
1336     { }
1337     \__unravel_input_gpop:N \l__unravel_head_gtl
1338     \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1339     \gtl_if_tl:NNTF \l__unravel_head_gtl
1340     {
1341       \tl_set:Ne \l__unravel_head_tl
1342       { \gtl_head:N \l__unravel_head_gtl }
1343       \token_if_eq_meaning:NNT
1344         \l__unravel_head_token \__unravel_special_relax:
1345       \__unravel_get_next_notexpanded:
1346     }
1347     { \tl_clear:N \l__unravel_head_tl }
1348   }
1349 \cs_new_protected:Npn \__unravel_get_next_aux:w
1350   { \cs_set_eq:NN \l__unravel_head_token }

```



(End of definition for `\_unravel_get_next:` and `\_unravel_get_next_aux:w`.)

`\_unravel_get_next_notexpanded:`  
`\_unravel_notexpanded_test:w`  
`\_unravel_notexpanded_expand:nN`  
`\_unravel_notexpanded_expand:NN`

At this point we have likely encountered a special `\relax` marker that we use to mark cases where `\noexpand` acts on a control sequence or an active character. To make sure of that check the control sequence has the form `\notexpanded:...`. Since we don't know the escape character we must use `\cs_to_str:N`, but that function is not meant for active characters and has a runaway argument if its argument is a space (active since we know its meaning is the special `\relax`). To avoid the runaway we include an arbitrary delimiter Z. If the token in `\l_unravel_head_tl` is not `\notexpanded:...` we do nothing. Otherwise `\_unravel_notexpanded_expand:n` reconstructs the token that was hit with `\noexpand` (an active character if the argument is a single character) and do the job of `\_unravel_get_next:`, setting `\l_unravel_head_token` to the special `\relax` marker for expandable commands, as `\noexpand` would.

```

1351 \cs_set_protected:Npn \_unravel_tmp:w #1
1352   {
1353     \cs_new_protected:Npn \_unravel_get_next_notexpanded:
1354       {
1355         \tl_if_eq:onTF { \l_unravel_head_tl } { \_unravel_unravel_marker: }
1356         { \_unravel_get_next_marker: }
1357         {
1358           \exp_args:Nne \use:nn \_unravel_notexpanded_test:w
1359           { \scan_stop: \exp_after:wN \cs_to_str:N \l_unravel_head_tl Z }
1360           \q_mark \_unravel_notexpanded_expand:n
1361           #1 Z \q_mark \use_none:n
1362           \q_stop
1363         }
1364       }
1365     \cs_new_protected:Npn \_unravel_notexpanded_test:w
1366       ##1 #1 ##2 Z \q_mark ##3##4 \q_stop
1367     { ##3 {##2} }
1368   }
1369 \exp_args:Ne \_unravel_tmp:w { \scan_stop: \tl_to_str:n { notexpanded: } }
1370 \group_begin:
1371   \char_set_catcode_active:n { 0 }
1372   \cs_new_protected:Npn \_unravel_notexpanded_expand:n #1
1373     {
1374       \exp_args:Ne \tl_if_empty:nTF { \str_tail:n {#1} }
1375       {
1376         \group_begin:
1377         \char_set_lccode:nm { 0 } { '#1 }
1378         \tex_lowercase:D
1379         {
1380           \group_end:
1381           \_unravel_notexpanded_expand:N ^^@
1382         }
1383       }
1384     {
1385       \group_begin: \exp_args:Nnc \group_end:
1386       \_unravel_notexpanded_expand:N { \use_none:n #1 }
1387     }
1388   }
1389 \group_end:
1390 \cs_new_protected:Npn \_unravel_notexpanded_expand:N #1

```

```

1391 {
1392   \gtl_set:Nn \l__unravel_head_gtl {#1}
1393   \tl_set:Nn \l__unravel_head_tl {#1}
1394   \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
1395 }

```

(End of definition for `\__unravel_get_next_notexpanded:` and others.)

`\__unravel_get_next_marker:` This is used to deal with nested unravel.

```

1396 \cs_new_protected:Npn \__unravel_get_next_marker:
1397 {
1398   \__unravel_get_next:
1399   \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
1400     { \__unravel_error:neeee { nested-unravel } { } { } { } { } }
1401     { \__unravel_error:neeee { internal } { marker~unknown } { } { } { } }
1402   \__unravel_input_gpop_item:NF \l__unravel_argi_tl
1403     { \__unravel_error:neeee { internal } { marker~1 } { } { } { } }
1404   \__unravel_input_gpop_item:NF \l__unravel_argii_tl
1405     { \__unravel_error:neeee { internal } { marker~2 } { } { } { } }
1406   \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
1407   \exp_args:Ne \__unravel_back_input:n
1408     { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
1409   \__unravel_get_next:
1410 }

```

(End of definition for `\__unravel_get_next_marker:.`)

`\__unravel_get_token:` Call `\__unravel_get_next:` to set `\l__unravel_head_gtl`, `\l__unravel_head_tl` and `\l__unravel_head_token`, then call `\__unravel_set_cmd:` to set `\l__unravel_head_cmd_int` and `\l__unravel_head_char_int`.

```

1411 \cs_new_protected:Npn \__unravel_get_token:
1412 {
1413   \__unravel_get_next:
1414   \__unravel_set_cmd:
1415 }

```

(End of definition for `\__unravel_get_token:.`)

`\__unravel_set_cmd:` After the call to `\__unravel_get_next:`, we find the command code `\l__unravel_head_cmd_int` and the character code `\l__unravel_head_char_int`, based only on `\l__unravel_head_token`. First set `\l__unravel_head_meaning_tl` from the `\meaning` of the first token. If the corresponding primitive exists, use the information to set the two integers. If the token is expandable, it can either be a macro or be a primitive that we somehow do not know (e.g., an expandable `XYTEX` or `LuaTEX` primitive perhaps). Otherwise, it can be a control sequence or a character.

```

1416 \cs_new_protected:Npn \__unravel_set_cmd:
1417 {
1418   \__unravel_set_cmd_aux_meaning:
1419   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1420     { }
1421     {
1422       \__unravel_token_if_expandable:NTF \l__unravel_head_token
1423         {
1424           \token_if_macro:NTF \l__unravel_head_token

```

```

1425         { \__unravel_set_cmd_aux_macro: }
1426         { \__unravel_set_cmd_aux_unknown: }
1427     }
1428     {
1429     \token_if_cs:NTF \l__unravel_head_token
1430     { \__unravel_set_cmd_aux_cs: }
1431     { \__unravel_set_cmd_aux_char: }
1432     }
1433 }
1434 }

```

(End of definition for \\_\_unravel\_set\_cmd:.)

\\_\_unravel\_set\_cmd\_aux\_meaning: Remove the leading escape character (\\_\_unravel\_strip\_escape:w takes care of special cases there) from the \meaning of the first token, then remove anything after the first :, which is present for macros, for marks, and for that character too. For any primitive except \nullfont, this leaves the primitive's name.

```

1435 \cs_new_protected:Npn \__unravel_set_cmd_aux_meaning:
1436 {
1437     \tl_set:Ne \l__unravel_head_meaning_tl
1438     {
1439         \exp_after:wN \__unravel_strip_escape:w
1440         \token_to_meaning:N \l__unravel_head_token
1441         \tl_to_str:n { : }
1442     }
1443     \tl_set:Ne \l__unravel_head_meaning_tl
1444     {
1445         \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1446         \l__unravel_head_meaning_tl \q_stop
1447     }
1448 }
1449 \use:e
1450 {
1451     \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1452     #1 \token_to_str:N : #2 \exp_not:N \q_stop {#1}
1453 }

```

(End of definition for \\_\_unravel\_set\_cmd\_aux\_meaning: and \\_\_unravel\_set\_cmd\_aux\_meaning:w.)

\\_\_unravel\_set\_cmd\_aux\_primitive:nTF Test if there is any information about the given (cleaned-up) \meaning. If there is, use that as the command and character integers.

```

\__unravel_set_cmd_aux_primitive:oTF
\__unravel_set_cmd_aux_primitive:nn
1454 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
1455 {
1456     \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1457     {
1458         \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1459         { c__unravel_tex_#1_tl }
1460         #2
1461     }
1462 }
1463 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
1464 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
1465 {
1466     \int_set:Nn \l__unravel_head_cmd_int {#1}

```

```

1467     \int_set:Nn \l__unravel_head_char_int {#2}
1468   }

```

*(End of definition for \\_\_unravel\_set\_cmd\_aux\_primitive:nTF and \\_\_unravel\_set\_cmd\_aux\_primitive:nn.)*

`\__unravel_set_cmd_aux_macro:` The token is a macro. There is no need to determine whether the macro is long/outer.

```

1469 \cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
1470 {
1471   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1472   \int_zero:N \l__unravel_head_char_int
1473 }

```

*(End of definition for \\_\_unravel\_set\_cmd\_aux\_macro:.)*

`\__unravel_set_cmd_aux_unknown:` Complain about an unknown primitive, and consider it as if it were `\relax`.

```

1474 \sys_if_engine_luatex:TF
1475 {
1476   \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1477   {
1478     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1479     \c__unravel_tex_relax_tl
1480     \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1481     { \tl_to_str:n { xpandable~luacall } }
1482     { }
1483     {
1484       \__unravel_error:neeee { unknown-primitive }
1485       { \l__unravel_head_meaning_tl } { } { } { }
1486     }
1487   }
1488 }
1489 {
1490   \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1491   {
1492     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1493     \c__unravel_tex_relax_tl
1494     \__unravel_error:neeee { unknown-primitive }
1495     { \l__unravel_head_meaning_tl } { } { } { }
1496   }
1497 }

```

*(End of definition for \\_\_unravel\_set\_cmd\_aux\_unknown:.)*

`\__unravel_set_cmd_aux_cs:` If the `\meaning` contains `elect_ font`, the control sequence is `\nullfont` or similar (note that we do not search for `select_ font`, as the code to trim the escape character from the meaning may have removed the leading `s`). Otherwise, we expect the `\meaning` to be `\char` or `\mathchar` or similar followed by " and an uppercase hexadecimal number, or one of `\count`, `\dimen`, `\skip`, `\muskip` or `\toks` followed by a decimal number.

```

1498 \cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
1499 {
1500   \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1501   { \tl_to_str:n { elect-font } }
1502   {
1503     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1504     \c__unravel_tex_nullfont_tl

```

```

1505     }
1506     { \_unravel_set_cmd_aux_numeric: }
1507 }

```

(End of definition for \\_unravel\_set\_cmd\_aux\_cs:.)

\\_unravel\_set\_cmd\_aux\_numeric: Insert \q\_mark before the first non-letter (in fact, anything less than A) in the \meaning by looping one character at a time (skipping spaces, but there should be none). We expect the first part to be char or mathchar (or kchar or omathchar in (u)pTeX), or one of count, dimen, skip, muskip, or toks. In the first two (three) cases, the command is char\_given or math\_given. It is otherwise identical to the corresponding primitive (\count etc.). We then keep track of the associated number (part after \q\_mark) in \l\_\_unravel\_head\_char\_int. For unknown non-expandable primitives, assuming that their meaning consists solely of letters, the \q\_mark is inserted at their end, and is followed by +0, so nothing breaks.

```

1508 \cs_new_protected:Npn \_unravel_set_cmd_aux_numeric:
1509 {
1510   \tl_set:Ne \l__unravel_tmpa_tl
1511     {
1512       \exp_after:wN \_unravel_set_cmd_aux_numeric:N
1513       \l__unravel_head_meaning_tl + 0
1514     }
1515   \exp_after:wN \_unravel_set_cmd_aux_numeric:w
1516   \l__unravel_tmpa_tl \q_stop
1517 }
1518 \cs_new:Npn \_unravel_set_cmd_aux_numeric:N #1
1519 {
1520   \if_int_compare:w '#1 < 'A \exp_stop_f:
1521     \exp_not:N \q_mark
1522     \exp_after:wN \use_i:nn
1523   \fi:
1524   #1 \_unravel_set_cmd_aux_numeric:N
1525 }
1526 \cs_new_protected:Npn \_unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1527 {
1528   \str_case:nnF {#1}
1529     {
1530       { char }      { \_unravel_set_cmd_aux_given:n { char_given } }
1531       { kchar }    { \_unravel_set_cmd_aux_given:n { char_given } }
1532       { mathchar } { \_unravel_set_cmd_aux_given:n { math_given } }
1533       { omathchar } { \_unravel_set_cmd_aux_given:n { math_given } }
1534     }
1535     {
1536       \_unravel_set_cmd_aux_primitive:nTF {#1}
1537       { }
1538       { \_unravel_set_cmd_aux_unknown: }
1539       \int_add:Nn \l__unravel_head_char_int { 100 000 }
1540     }
1541   \int_add:Nn \l__unravel_head_char_int {#2}
1542 }
1543 \cs_new_protected:Npn \_unravel_set_cmd_aux_given:n #1
1544 {
1545   \int_set:Nn \l__unravel_head_cmd_int { \_unravel_tex_use:n {#1} }
1546   \int_zero:N \l__unravel_head_char_int

```

```
1547 }
```

(End of definition for `\__unravel_set_cmd_aux_numeric`: and others.)

`\__unravel_set_cmd_aux_char`: At this point, the `\meaning` token list has been shortened by the code meant to remove the escape character. We thus set it again to the `\meaning` of the leading token. The command is then the first word (delimited by a space) of the `\meaning`, followed by `_char`, except for category other, where we use `other_char`. For the character code, there is a need to expand `\__unravel_token_to_char:N` before placing ‘.

```
1548 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:
1549 {
1550   \tl_set:Ne \l__unravel_head_meaning_tl
1551     { \token_to_meaning:N \l__unravel_head_token }
1552   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1553     { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1554   \exp_after:wN \__unravel_set_cmd_aux_char:w
1555     \l__unravel_head_meaning_tl \q_stop
1556   \exp_args:NNe \int_set:Nn \l__unravel_head_char_int
1557     { ‘ \__unravel_token_to_char:N \l__unravel_head_token }
1558 }
1559 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1560 {
1561   \int_set:Nn \l__unravel_head_cmd_int
1562     { \__unravel_tex_use:n { #1_char } }
1563 }
```

(End of definition for `\__unravel_set_cmd_aux_char`: and `\__unravel_set_cmd_aux_char:w`.)

## 2.5 Manipulating the input

### 2.5.1 Elementary operations

`\__unravel_input_to_str`: Map `\gtl_to_str:c` through the input stack.

```
1564 \cs_new:Npn \__unravel_input_to_str:
1565 {
1566   \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1567     \__unravel_input_to_str_aux:n
1568 }
1569 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1570 { \gtl_to_str:c { g__unravel_input_#1_gtl } }
```

(End of definition for `\__unravel_input_to_str:`.)

`\__unravel_input_if_empty:TF` If the input stack is empty, the input contains no token. Otherwise, check the top of the stack for tokens: if there are, then the input is non-empty, and if there are none, then we get rid of the top of stack and loop.

```
1571 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1572 {
1573   \int_compare:nNnTF \g__unravel_input_int = 0
1574     { \use_i:nn }
1575     {
1576       \gtl_if_empty:cTF
1577         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1578         {
```

```

1579         \int_gdecr:N \g__unravel_input_int
1580         \__unravel_input_if_empty:TF
1581     }
1582     {
1583         \__unravel_input_split:
1584         \use_ii:nn
1585     }
1586 }
1587 }

```

(End of definition for \\_\_unravel\_input\_if\_empty:TF.)

\\_\_unravel\_input\_split: If the input is completely flat, and is a token list starting with an N-type token, try to unflatten it by splitting at each occurrence of that first token.

```

1588 \cs_new_protected:Npn \__unravel_input_split:
1589 {
1590     \int_compare:nNnT \g__unravel_input_int = 1
1591     {
1592         \exp_args:Nc \__unravel_input_split_aux:N
1593         { g__unravel_input_1_gtl }
1594     }
1595 }
1596 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1597 {
1598     \gtl_if_tl:NT #1
1599     {
1600         \gtl_if_head_is_N_type:NT #1
1601         {
1602             \tl_set:Ne \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1603             \exp_args:NNe \use:nn
1604             \__unravel_input_split_auxii:N
1605             { \tl_head:N \l__unravel_input_tmpa_tl }
1606         }
1607     }
1608 }
1609 \cs_new_protected:Npn \__unravel_input_split_auxiii:N #1
1610 {
1611     \token_if_parameter:NF #1
1612     {
1613         \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1614         { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1615         \group_begin:
1616         \cs_set:Npn \__unravel_input_split_auxiii:w
1617             ##1 \__unravel_input_split_end: { + 1 }
1618         \int_gset:Nn \g__unravel_input_int
1619             { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1620         \group_end:
1621         \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1622         \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1623     }
1624 }
1625 \cs_new:Npn \__unravel_input_split_end: { }
1626 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1627     #1 \__unravel_input_split_end:

```

```

1628 {
1629   \gtl_gclear_new:c
1630   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1631   \gtl_gset:cn
1632   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1633   \int_gdecr:N \g__unravel_input_tmpa_int
1634 }

```

(End of definition for \\_\_unravel\_input\_split:.)

\\_\_unravel\_input\_gset:n At first, all of the input is in the same gtl.

```

1635 \cs_new_protected:Npn \__unravel_input_gset:n
1636 {
1637   \int_gzero:N \g__unravel_input_int
1638   \__unravel_back_input:n
1639 }

```

(End of definition for \\_\_unravel\_input\_gset:n.)

\\_\_unravel\_input\_get:N

```

1640 \cs_new_protected:Npn \__unravel_input_get:N #1
1641 {
1642   \__unravel_input_if_empty:TF
1643   { \gtl_set:Nn #1 { \q_no_value } }
1644   {
1645     \gtl_get_left:cN
1646     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1647   }
1648 }

```

(End of definition for \\_\_unravel\_input\_get:N.)

\\_\_unravel\_input\_get\_left:N

```

\__unravel_input_get_left_aux:nN
\l__unravel_input_get_left_tl
1649 \tl_new:N \l__unravel_input_get_left_tl
1650 \cs_new_protected:Npn \__unravel_input_get_left:N #1
1651 {
1652   \tl_clear:N #1
1653   \exp_args:NV \__unravel_input_get_left_aux:nN \g__unravel_input_int #1
1654 }
1655 \cs_new_protected:Npn \__unravel_input_get_left_aux:nN #1#2
1656 {
1657   \int_compare:nNnF {#1} = 0
1658   {
1659     \tl_set:Ne \l__unravel_input_get_left_tl
1660     { \gtl_left_tl:c { g__unravel_input_#1_gtl } }
1661     \tl_concat:NNN #2 #2 \l__unravel_input_get_left_tl
1662     \gtl_if_tl:cT { g__unravel_input_#1_gtl }
1663     {
1664       \exp_args:Nf \__unravel_input_get_left_aux:nN
1665       { \int_eval:n { #1 - 1 } } #2
1666     }
1667   }
1668 }

```

(End of definition for \\_\_unravel\_input\_get\_left:N, \\_\_unravel\_input\_get\_left\_aux:nN, and \l\_\_unravel\_input\_get\_left\_tl.)



`\__unravel_input_gpop:N` Call `\__unravel_input_if_empty:TF` to remove empty levels from the input stack, then extract the first token from the left-most non-empty level.

```

1669 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1670 {
1671   \__unravel_input_if_empty:TF
1672     { \gtl_set:Nn #1 { \q_no_value } }
1673     {
1674       \gtl_gpop_left:cN
1675       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1676     }
1677 }

```

*(End of definition for `\__unravel_input_gpop:N`.)*

`\__unravel_input_merge:` Merge the top two levels of input. This requires, but does not check, that `\g__unravel_input_int` is at least 2.

```

1678 \cs_new_protected:Npn \__unravel_input_merge:
1679 {
1680   \int_gdecr:N \g__unravel_input_int
1681   \gtl_gconcat:ccc
1682     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1683     { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1684     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1685   \gtl_gclear:c
1686   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1687 }

```

*(End of definition for `\__unravel_input_merge:`.)*

`\__unravel_input_gpop_item:NNTF` If there is no input, we cannot pop an item. Otherwise, try to pop from the top of the input stack. If this succeeds, or if this failed and the top of stack has extra end-group characters, or if the input stack contains only the top-most item, then the answer given by `\gtl_gpop_left_item:NNTF` is the correct one, which we return. Otherwise, merge the top two levels and repeat.

`\__unravel_input_gpop_item_aux:NN`

```

1688 \prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
1689 {
1690   \int_compare:nNnTF \g__unravel_input_int = 0
1691     { \prg_return_false: }
1692     {
1693       \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1694       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1695     }
1696 }
1697 \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
1698 {
1699   \gtl_gpop_left_item:NNTF #1#2
1700   { \prg_return_true: }
1701   {
1702     \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
1703     { \prg_return_false: }
1704     {
1705       \int_compare:nNnTF \g__unravel_input_int = 1
1706       { \prg_return_false: }
1707       {

```

```

1708         \__unravel_input_merge:
1709         \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1710         {
1711             g__unravel_input_
1712             \int_use:N \g__unravel_input_int _gtl
1713         }
1714         #2
1715     }
1716 }
1717 }
1718 }

```

(End of definition for \\_\_unravel\_input\_gpop\_item:NTF and \\_\_unravel\_input\_gpop\_item\_aux:NN.)

\\_\_unravel\_input\_gpop\_tl:N

```

1719 \cs_new_protected:Npn \__unravel_input_gpop_tl:N #1
1720 { \tl_clear:N #1 \__unravel_input_gpop_tl_aux:N #1 }
1721 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:N #1
1722 {
1723     \int_compare:nNnF \g__unravel_input_int = 0
1724     {
1725         \exp_args:Nc \__unravel_input_gpop_tl_aux:NN
1726         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1727     }
1728 }
1729 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
1730 {
1731     \gtl_if_tl:NTF #1
1732     {
1733         \tl_put_right:Ne #2 { \gtl_left_tl:N #1 }
1734         \gtl_gclear:N #1
1735         \int_gdecr:N \g__unravel_input_int
1736         \__unravel_input_gpop_tl_aux:N #2
1737     }
1738     {
1739         \int_compare:nNnTF \g__unravel_input_int > 1
1740         { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
1741         { \use_i:nn }
1742         {
1743             \tl_put_right:Ne #2 { \gtl_left_tl:N #1 }
1744             \gtl_gpop_left_tl:N #1
1745         }
1746         {
1747             \__unravel_input_merge:
1748             \__unravel_input_gpop_tl_aux:N #2
1749         }
1750     }
1751 }

```

(End of definition for \\_\_unravel\_input\_gpop\_tl:N.)

\\_\_unravel\_input\_if\_head\_is\_group\_begin:TF

Call \\_\_unravel\_input\_if\_empty:TF to remove empty levels from the input stack, then check if the left-most non-empty level starts with an explicit begin-group character token.

```

1752 \prg_new_protected_conditional:Npnn \__unravel_input_if_head_is_group_begin: { T , F , TF }

```

```

1753 {
1754   \__unravel_input_if_empty:TF
1755   { \prg_return_false: }
1756   {
1757     \exp_args:Nc \gtl_if_head_is_group_begin:NTF
1758     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1759     { \prg_return_true: }
1760     { \prg_return_false: }
1761   }
1762 }

```

(End of definition for \\_\_unravel\_input\_if\_head\_is\_group\_begin:TF.)

\\_\_unravel\_back\_input:n Insert a token list back into the input. Use \gtl\_gclear\_new:c to define the gtl variable if necessary: this happens whenever a new largest value of \g\_\_unravel\_input\_int is reached.

\\_\_unravel\_back\_input:V  
 \\_\_unravel\_back\_input:o

```

1763 \cs_new_protected:Npn \__unravel_back_input:n
1764 {
1765   \int_gincr:N \g__unravel_input_int
1766   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1767   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1768 }
1769 \cs_generate_variant:Nn \__unravel_back_input:n { V , o }

```

(End of definition for \\_\_unravel\_back\_input:n.)

\\_\_unravel\_back\_input\_gtl:N Insert a generalized token list back into the input.

```

1770 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1771 {
1772   \gtl_if_tl:NTF #1
1773   { \exp_args:Ne \__unravel_back_input:n { \gtl_left_tl:N #1 } }
1774   {
1775     \gtl_gconcat:cNc
1776     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1777     #1
1778     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1779   }
1780 }

```

(End of definition for \\_\_unravel\_back\_input\_gtl:N.)

\\_\_unravel\_back\_input: Insert the last token read back into the input stream.

```

1781 \cs_new_protected:Npn \__unravel_back_input:
1782 { \__unravel_back_input_gtl:N \l__unravel_head_gtl }

```

(End of definition for \\_\_unravel\_back\_input:.)

\\_\_unravel\_back\_input\_tl\_o: Insert the \l\_\_unravel\_head\_tl (may or may not be the last token read) back into the input stream, after expanding it once. Then print some diagnostic information.

```

1783 \cs_new_protected:Npn \__unravel_back_input_tl_o:
1784 {
1785   \tl_set:Ne \l__unravel_tmpa_tl
1786   { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1787   \__unravel_back_input:V \l__unravel_tmpa_tl
1788   \__unravel_print_expansion:e

```

```

1789     { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1790   }

```

(End of definition for \\_\_unravel\_back\_input\_tl\_o:.)

## 2.5.2 Insert token for error recovery

\\_\_unravel\_insert\_relax: This function inserts TeX's frozen\_relax. It is called when a conditional is not done finding its condition, but hits the corresponding \fi or \or or \else, or when \input appears while \g\_\_unravel\_name\_in\_progress\_bool is true.

```

1791 \cs_new_protected:Npn \__unravel_insert_relax:
1792 {
1793   \__unravel_back_input:
1794   \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1795   \__unravel_back_input:
1796   \__unravel_print_action:
1797 }

```

(End of definition for \\_\_unravel\_insert\_relax:.)

\\_\_unravel\_insert\_group\_begin\_error:

```

1798 \cs_new_protected:Npn \__unravel_insert_group_begin_error:
1799 {
1800   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
1801   \__unravel_back_input:
1802   \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1803   \__unravel_back_input:
1804   \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
1805   \__unravel_print_action:
1806 }

```

(End of definition for \\_\_unravel\_insert\_group\_begin\_error:.)

\\_\_unravel\_insert\_dollar\_error:

```

1807 \cs_new_protected:Npn \__unravel_insert_dollar_error:
1808 {
1809   \__unravel_back_input:
1810   \__unravel_back_input:n { $ } % $
1811   \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
1812   \__unravel_print_action:
1813 }

```

(End of definition for \\_\_unravel\_insert\_dollar\_error:.)

## 2.5.3 Macro calls

\\_\_unravel\_macro\_prefix:N

\\_\_unravel\_macro\_parameter:N

\\_\_unravel\_macro\_replacement:N

```

1814 \use:e
1815 {
1816   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
1817   {
1818     \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
1819     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1820     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }

```

```

1821     \exp_not:N \q_stop
1822   }
1823   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
1824     \exp_not:n {#1} \tl_to_str:n { : } \exp_not:n { #2 -> }
1825     \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1826     { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
1827   }
1828   \cs_new:Npn \__unravel_macro_prefix:N #1
1829     { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1830   \cs_new:Npn \__unravel_macro_parameter:N #1
1831     { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1832   \cs_new:Npn \__unravel_macro_replacement:N #1
1833     { \__unravel_macro_split_do:NN #1 \use_iii:nnn }

```

(End of definition for `\__unravel_macro_prefix:N`, `\__unravel_macro_parameter:N`, and `\__unravel_macro_replacement:N`.)

Macros are simply expanded once. We cannot determine precisely which tokens a macro will need for its parameters, but we know that it must form a balanced token list. Thus we can be safe by extracting the longest balanced prefix in the input and working with that. We need to check if the argument was braced, to improve the error recovery for a non-`\long` macro receiving `\par`.

```

\__unravel_macro_call:
\__unravel_macro_call_safe:
\__unravel_macro_call_quick:
  \__unravel_macro_call_quick_loop:NNN
  \__unravel_macro_call_quick_loop:NN
  \__unravel_macro_call_quick_runaway:Nw
1834 \cs_new_protected:Npn \__unravel_macro_call:
1835   {
1836     \bool_if:NTF \g__unravel_speedup_macros_bool
1837       {
1838         \tl_set:Ne \l__unravel_tmpa_tl
1839           { ^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1840         \__unravel_tl_if_in:ooTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1841           { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1842       }
1843     { \__unravel_macro_call_safe: }
1844     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1845     \__unravel_print_expansion:
1846   }
1847 \cs_new_protected:Npn \__unravel_macro_call_safe:
1848   {
1849     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1850     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1851   }
1852 \cs_new_protected:Npn \__unravel_macro_call_quick:
1853   {
1854     \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1855     { ? \use_none_delimit_by_q_stop:w } \q_stop
1856   }
1857 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1858   {
1859     \use_none:n #2
1860     \__unravel_input_if_head_is_group_begin:TF
1861       { \__unravel_macro_call_quick_loop:NN \prg_do_nothing: }
1862       { \__unravel_macro_call_quick_loop:NN \use:n }
1863     #3
1864   }
1865 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NN #1#2

```

```

1866 {
1867   \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1868   { \__unravel_macro_call_quick_runaway:Nw #2 }
1869   \tl_put_right:Ne \l__unravel_head_tl
1870   { #1 { \exp_not:V \l__unravel_tmpa_tl } }
1871   \__unravel_macro_call_quick_loop:NNN
1872   #2
1873 }
1874 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1875 {
1876   \__unravel_error:neeee { runaway-macro-parameter }
1877   { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1878 }

```

(End of definition for `\__unravel_macro_call`: and others.)

## 2.6 Expand next token

`\__unravel_expand_do:N` The argument is a command that will almost always be run to continue a loop whose aim is to find the next non-expandable token, for various purposes. The only case where we will end up grabbing the argument is to suppress the loop by `\__unravel_noexpand:N`.

- `\__unravel_get_x_next`: when  $\TeX$  is looking for the first non-expandable token in the main loop or when looking for numbers, optional spaces etc.
- `\__unravel_get_x_or_protected`: at the start of an alignment cell.
- `\__unravel_get_token_xdef`: in the replacement text of `\edef` and `\xdef`.
- `\__unravel_get_token_x`: in the argument of `\message` and the like.
- `\prg_do_nothing`: in `\__unravel_expandafter`: namely after `\expandafter`.

We mimick  $\TeX$ 's structure, distinguishing macros from other commands because we find macro arguments very differently from primitives.

```

1879 \cs_new_protected:Npn \__unravel_expand_do:N
1880 {
1881   \__unravel_set_action_text:
1882   \bool_if:NT \g__unravel_internal_debug_bool
1883   {
1884     \__unravel_set_cmd:
1885     \exp_args:Ne \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_int } }
1886   }
1887   \token_if_macro:NTF \l__unravel_head_token
1888   { \__unravel_macro_call: }
1889   { \__unravel_expand_nonmacro: }
1890 }

```

(End of definition for `\__unravel_expand_do:N`.)

`\__unravel_expand_nonmacro:` The token is a primitive. We find its (cleaned-up) `\meaning`, and call the function implementing that expansion. If we do not recognize the meaning then it is probably an unknown primitive. Then do something similar to what we do for macros: get all tokens that are not too unlikely to appear in the arguments of the primitive and expand the resulting token list once before putting it back into the input stream.

```

1891 \cs_new_protected:Npn \__unravel_expand_nonmacro:
1892 {
1893   \__unravel_set_cmd_aux_meaning:
1894   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1895   {
1896     \cs_if_exist_use:cF
1897     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1898     { \__unravel_error:neeee { internal } { expandable } { } { } { } }
1899   }
1900   {
1901     \__unravel_set_cmd_aux_unknown:
1902     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1903     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1904     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1905     \__unravel_print_expansion:
1906   }
1907 }

```

*(End of definition for \\_\_unravel\_expand\_nonmacro:.)*

`\__unravel_get_x_next:` Get a token. If it is expandable, then expand it, and repeat. This function does not set the `cmd` and `char` integers. It is the basis of all routines that look for keywords, numbers, equal signs, filenames, optional spaces etc (in the language of L<sup>A</sup>T<sub>E</sub>X3 these are situations where T<sub>E</sub>X “f-expands”). It is also the basis of the `\__unravel_main_loop:`.

```

1908 \cs_new_protected:Npn \__unravel_get_x_next:
1909 {
1910   \__unravel_get_next:
1911   \__unravel_token_if_expandable:NT \l__unravel_head_token
1912   { \__unravel_expand_do:N \__unravel_get_x_next: }
1913 }

```

*(End of definition for \\_\_unravel\_get\_x\_next:.)*

`\__unravel_get_x_or_protected:` Get a token. If it is expandable, but not protected, then expand it, and repeat. This function does not set the `cmd` and `char` integers. This function is not used at present: it will be used at the start of alignment cells.

```

1914 \cs_new_protected:Npn \__unravel_get_x_or_protected:
1915 {
1916   \__unravel_get_next:
1917   \__unravel_token_if_protected:NF \l__unravel_head_token
1918   { \__unravel_expand_do:N \__unravel_get_x_or_protected: }
1919 }

```

*(End of definition for \\_\_unravel\_get\_x\_or\_protected:.)*

`\__unravel_get_token_xdef:` These are similar to `\__unravel_get_x_next:`, for use when reading the replacement text of `\edef`/`\xdef` or the argument of a primitive like `\message` that should be expanded as we read tokens. Loop until finding a non-expandable token (or protected macro).

```

1920 \cs_new_protected:Npn \__unravel_get_token_xdef:
1921 {
1922   \__unravel_get_next:
1923   \__unravel_token_if_protected:NF \l__unravel_head_token
1924   { \__unravel_expand_do:N \__unravel_get_token_xdef: }

```

```

1925 }
1926 \cs_new_protected:Npn \__unravel_get_token_x:
1927 {
1928   \__unravel_get_next:
1929   \__unravel_token_if_protected:NF \l__unravel_head_token
1930   { \__unravel_expand_do:N \__unravel_get_token_x: }
1931 }

```

(End of definition for \\_\_unravel\_get\_token\_xdef: and \\_\_unravel\_get\_token\_x:.)

## 2.7 Basic scanning subroutines

\\_\_unravel\_get\_x\_non\_blank: This function does not set the cmd and char integers.

```

1932 \cs_new_protected:Npn \__unravel_get_x_non_blank:
1933 {
1934   \__unravel_get_x_next:
1935   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1936   { \__unravel_get_x_non_blank: }
1937 }

```

(End of definition for \\_\_unravel\_get\_x\_non\_blank:.)

\\_\_unravel\_get\_x\_non\_relax: This function does not set the cmd and char integers.

```

1938 \cs_new_protected:Npn \__unravel_get_x_non_relax:
1939 {
1940   \__unravel_get_x_next:
1941   \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
1942   { \__unravel_get_x_non_relax: }
1943   {
1944     \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_special_relax:
1945     { \__unravel_get_x_non_relax: }
1946     {
1947       \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1948       { \__unravel_get_x_non_relax: }
1949     }
1950   }
1951 }

```

(End of definition for \\_\_unravel\_get\_x\_non\_relax:.)

\\_\_unravel\_skip\_optional\_space:

```

1952 \cs_new_protected:Npn \__unravel_skip_optional_space:
1953 {
1954   \__unravel_get_x_next:
1955   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
1956   { \__unravel_back_input: }
1957 }

```

(End of definition for \\_\_unravel\_skip\_optional\_space:.)

\\_\_unravel\_scan\_optional\_equals: See TeX's `scan_optional_equals`. In all cases we forcefully insert an equal sign in the output, because this sign is required, as `\__unravel_rescan_something_internal:n` leaves raw numbers in the previous-input sequence.

```

1958 \cs_new_protected:Npn \__unravel_scan_optional_equals:

```



```

1959 {
1960   \__unravel_get_x_non_blank:
1961   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1962     { \__unravel_prev_input:n { = } }
1963     {
1964       \__unravel_prev_input_silent:n { = }
1965       \__unravel_back_input:
1966     }
1967 }

```

(End of definition for \\_\_unravel\_scan\_optional\_equals:.)

\\_\_unravel\_scan\_left\_brace: The presence of \relax is allowed before a begin-group token. If there is no begin-group token, insert one, produce an error, and scan that begin-group using \\_\_unravel\_get\_next:.

```

1968 \cs_new_protected:Npn \__unravel_scan_left_brace:
1969 {
1970   \__unravel_get_x_non_relax:
1971   \token_if_eq_catcode:NMF \l__unravel_head_token \c_group_begin_token
1972     {
1973       \__unravel_insert_group_begin_error:
1974       \__unravel_get_next:
1975     }
1976 }

```

(End of definition for \\_\_unravel\_scan\_left\_brace:.)

\\_\_unravel\_scan\_keyword:n The details of how T<sub>E</sub>X looks for keywords are quite tricky to get right, in particular with respect to expansion, case-insensitivity, and spaces. We get rid of the case issue by requiring the keyword to be given in both cases, intertwined: for instance, \\_\_unravel\_scan\_keyword:n{TF}. Then loop through pairs of letters (which should be matching lowercase and uppercase letters). The looping auxiliary takes three arguments, the first of which is a boolean, true if spaces are allowed (no letter of the keyword has been found yet). At each iteration, get a token, with expansion, and test whether it is a non-active character equal (in character code) to either letter of the pair: this happens if the token is not “definable” (neither a control sequence nor an active character) and it has the right string representation. . . well, it could also be doubled (macro parameter character), hence we look at the first character only; spaces become an empty string, but this works out because no keyword contains a space. So, at each iteration, if the token is the correct non-active character, add it to the previous-input sequence (as a generalized token list since keywords may match begin-group or end-group characters), and otherwise break with \\_\_unravel\_scan\_keyword\_false:w, unless we are still at the beginning of the keyword and the token is a space. When the loop reaches the end of the keyword letter pairs, complain if there were an odd number of letters, and otherwise conclude the loop with \\_\_unravel\_scan\_keyword\_true:, which stores the keyword, converted to a string. Note that T<sub>E</sub>X’s skipping of leading spaces here must be intertwined with the search for keyword, as is shown by the (plain T<sub>E</sub>X) example

```

\lccode32='f \lowercase{\def\fspace{ }}
\skip0=1pt plus 1 \fspace il\relax
\message{\the\skip0} % => 1pt plus 1fil

```

```

1977 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1978 { \__unravel_scan_keyword:nTF {#1} { } { } }
1979 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1980 { T , F , TF }
1981 {
1982   \__unravel_prev_input_gpush_gtl:
1983   \__unravel_scan_keyword_loop:NNN \c_true_bool
1984   #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1985 }
1986 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1987 {
1988   \quark_if_recursion_tail_stop_do:nn {#2}
1989   { \__unravel_scan_keyword_true: }
1990   \quark_if_recursion_tail_stop_do:nn {#3}
1991   { \__unravel_error:neeee { internal } { odd-keyword-length } { } { } { } }
1992   \__unravel_get_x_next:
1993   \__unravel_scan_keyword_test:NNTF #2#3
1994   {
1995     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1996     \__unravel_scan_keyword_loop:NNN \c_false_bool
1997   }
1998   {
1999     \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
2000     { \__unravel_scan_keyword_false:w }
2001     \bool_if:NF #1
2002     { \__unravel_scan_keyword_false:w }
2003     \__unravel_scan_keyword_loop:NNN #1#2#3
2004   }
2005 }
2006 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
2007 { TF }
2008 {
2009   \__unravel_gtl_if_head_is_definable:NNTF \l__unravel_head_gtl
2010   { \prg_return_false: }
2011   {
2012     \str_if_eq:eeTF
2013     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
2014     { \prg_return_true: }
2015     {
2016       \str_if_eq:eeTF
2017       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
2018       { \prg_return_true: }
2019       { \prg_return_false: }
2020     }
2021   }
2022 }
2023 \cs_new_protected:Npn \__unravel_scan_keyword_true:
2024 {
2025   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2026   \__unravel_prev_input:e { \gtl_to_str:N \l__unravel_tmpb_gtl }
2027   \prg_return_true:
2028 }
2029 \cs_new_protected:Npn \__unravel_scan_keyword_false:w
2030 #1 \q_recursion_stop

```

```

2031 {
2032   \__unravel_back_input:
2033   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2034   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
2035   \prg_return_false:
2036 }

```

(End of definition for \\_\_unravel\_scan\_keyword:n and others.)

\\_\_unravel\_scan\_to: Used when to is mandatory: after \read or \readline and after \vsplit.

```

2037 \cs_new_protected:Npn \__unravel_scan_to:
2038 {
2039   \__unravel_scan_keyword:nF { tTo0 }
2040   {
2041     \__unravel_error:nnnnn { missing-to } { } { } { } { }
2042     \__unravel_prev_input:n { to }
2043   }
2044 }

```

(End of definition for \\_\_unravel\_scan\_to:.)

\\_\_unravel\_scan\_font\_ident: Find a font identifier.

```

2045 \cs_new_protected:Npn \__unravel_scan_font_ident:
2046 {
2047   \__unravel_get_x_non_blank:
2048   \__unravel_set_cmd:
2049   \int_case:nnF \l__unravel_head_cmd_int
2050   {
2051     { \__unravel_tex_use:n { def_font } }
2052     { \__unravel_prev_input:V \l__unravel_head_tl }
2053     { \__unravel_tex_use:n { letterspace_font } }
2054     { \__unravel_prev_input:V \l__unravel_head_tl }
2055     { \__unravel_tex_use:n { pdf_copy_font } }
2056     { \__unravel_prev_input:V \l__unravel_head_tl }
2057     { \__unravel_tex_use:n { set_font } }
2058     { \__unravel_prev_input:V \l__unravel_head_tl }
2059     { \__unravel_tex_use:n { def_family } }
2060     {
2061       \__unravel_prev_input:V \l__unravel_head_tl
2062       \__unravel_scan_int:
2063     }
2064   }
2065   {
2066     \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
2067     \__unravel_back_input:
2068     \__unravel_prev_input:n { \__unravel_nullfont: }
2069   }
2070 }

```

(End of definition for \\_\_unravel\_scan\_font\_ident:.)

\\_\_unravel\_scan\_font\_int: Find operands for one of \hyphenchar's friends (command code assign\_font\_int=78).

```

2071 \cs_new_protected:Npn \__unravel_scan_font_int:
2072 {
2073   \int_case:nnF \l__unravel_head_char_int

```

```

2074     {
2075     { 0 } { \__unravel_scan_font_ident: }
2076     { 1 } { \__unravel_scan_font_ident: }
2077     { 6 } { \__unravel_scan_font_ident: }
2078     }
2079     { \__unravel_scan_font_ident: \__unravel_scan_int: }
2080   }

```

(End of definition for \\_\_unravel\_scan\_font\_int:.)

\\_\_unravel\_scan\_font\_dimen: Find operands for \fontdimen.

```

2081 \cs_new_protected:Npn \__unravel_scan_font_dimen:
2082   {
2083     \__unravel_scan_int:
2084     \__unravel_scan_font_ident:
2085   }

```

(End of definition for \\_\_unravel\_scan\_font\_dimen:.)

\\_\_unravel\_rescan\_something\_internal:n Receives an (explicit) “level” argument:

\\_\_unravel\_scan\_something\_aux:nwn

- `int_val=0` for integer values;
- `dimen_val=1` for dimension values;
- `glue_val=2` for glue specifications;
- `mu_val=3` for math glue specifications;
- `ident_val=4` for font identifiers (this never happens);
- `tok_val=5` for token lists (after `\the` or `\showthe`).

Scans something internal, and places its value, converted to the given level, to the right of the last item of the previous-input sequence, then sets `\g__unravel_val_level_int` to the found level (level before conversion, so this may be higher than requested).

From `\__unravel_thing_case:`, get the information about what level is produced by the given token once it has received all its operands (head of `\l__unravel_tmpa_tl`), and about what to do to find those operands (tail of `\l__unravel_tmpa_tl`). If the first token may not appear after `\the` at all, `\__unravel_thing_case:` gives level 8.

If the argument (`#3` in the auxiliary) is  $< 4$  but the level that will be produced (`#1` in the auxiliary) is  $\geq 4$  (that is, 4, 5, or 8) complain about a missing number and insert a zero dimension, to get exactly  $\TeX$ 's error recovery. If the level produced is 8, complain that `\the` cannot do this.

Otherwise, scan the arguments (in a new input level). If both the argument and the level produced are  $< 4$ , then get the value with `\__unravel_thing_use_get:nnNN` which downgrades from glue to dimension to integer and produces the `incompatible-units` error if needed. The only remaining case is that the argument is 5 (since 4 is never used) and the level produced is that or less: then the value found is used with `\__unravel_the:w`.

Finally, tell the user the tokens that have been found (if there was a single token, its meaning as well) and their value. Use `=>` rather than `=` because the value displayed is the value used, not the actual value (this matters in constructions such as

`\parindent=\parskip` where a skip or a dimen is downgraded to a dimen or an int, or when there was an error).

```

2086 \cs_new_protected:Npn \__unravel_rescan_something_internal:n #1
2087 {
2088   \__unravel_set_cmd:
2089   \__unravel_set_action_text:
2090   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
2091   \exp_after:wN \__unravel_scan_something_aux:nwn
2092     \l__unravel_tmpa_tl \q_stop {#1}
2093 }
2094 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
2095 {
2096   \int_compare:nT { #3 < 4 <= #1 }
2097     {
2098       \__unravel_back_input:
2099       \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2100       \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
2101       \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
2102       \__unravel_break:w
2103     }
2104   \int_compare:nNnT {#1} = { 8 }
2105     {
2106       \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
2107       \__unravel_rescan_something_internal_auxii:nn 0 { 0 }
2108       \__unravel_break:w
2109     }
2110   \tl_if_empty:nF {#2}
2111     {
2112       \__unravel_prev_input_gpush:N \l__unravel_head_tl
2113       \__unravel_print_action:
2114       #2
2115       \__unravel_prev_input_gpop:N \l__unravel_head_tl
2116     }
2117   \int_compare:nNnTF {#3} < { 4 }
2118     { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
2119     { \tl_set:Ne \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
2120   \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
2121   \__unravel_break_point:
2122   \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
2123 }
2124 \cs_new_protected:Npn \__unravel_rescan_something_internal_auxii:nn #1#2
2125 {
2126   \__unravel_prev_input_silent:n {#1}
2127   \__unravel_set_action_text:
2128   \__unravel_set_action_text:e
2129   { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
2130   \int_gset:Nn \g__unravel_val_level_int {#2}
2131 }
2132 \cs_generate_variant:Nn \__unravel_rescan_something_internal_auxii:nn { V }

```

*(End of definition for \\_\_unravel\_rescan\_something\_internal:n and \\_\_unravel\_scan\_something\_aux:nwn.)*

`\__unravel_thing_case:` This expands to a digit (the level generated by whatever token is the current head),  
`\__unravel_thing_last_item:` followed by some code to fetch necessary operands. In most cases, this can be done by  
`\__unravel_thing_register:`

simply looking at the cmd integer, but for last\_item, set\_aux, set\_shape and register, the level of the token, or what has to be scanned, depends on the char integer. When the token is not allowed after \the (or at any other position where \\_unravel\_rescan\_something\_internal:n is called), the resulting level is 8, large enough so that the main function knows it is forbidden.

```

2133 \cs_new:Npn \_unravel_thing_case:
2134 {
2135   \int_case:nnF \l__unravel_head_cmd_int
2136     {
2137       { 68 } { 0 } % char_given
2138       { 69 } { 0 } % math_given
2139       { 70 } { \_unravel_thing_last_item: } % last_item
2140       { 71 } { 5 \_unravel_scan_toks_register: } % toks_register
2141       { 72 } { 5 } % assign_toks
2142       { 73 } { 0 } % assign_int
2143       { 74 } { 1 } % assign_dimen
2144       { 75 } { 2 } % assign_glue
2145       { 76 } { 3 } % assign_mu_glue
2146       { 77 } { 1 \_unravel_scan_font_dimen: } % assign_font_dimen
2147       { 78 } { 0 \_unravel_scan_font_int: } % assign_font_int
2148       { 79 } { \_unravel_thing_set_aux: } % set_aux
2149       { 80 } { 0 } % set_prev_graf
2150       { 81 } { 1 } % set_page_dimen
2151       { 82 } { 0 } % set_page_int
2152       { 83 } { 1 \_unravel_scan_int: } % set_box_dimen
2153       { 84 } { \_unravel_thing_set_shape: } % set_shape
2154       { 85 } { 0 \_unravel_scan_int: } % def_code
2155       { 86 } { 4 \_unravel_scan_int: } % def_family
2156       { 87 } { 4 } % set_font
2157       { 88 } { 4 } % def_font
2158       { 89 } { \_unravel_thing_register: } % register
2159       {101 } { 4 } % letterspace_font
2160       {102 } { 4 } % pdf_copy_font
2161     }
2162   { 8 }
2163 }
2164 \cs_new:Npn \_unravel_thing_set_aux:
2165 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
2166 \cs_new:Npn \_unravel_thing_set_shape:
2167 { \int_compare:nNnTF \l__unravel_head_char_int = 0 { 0 } { 0 \_unravel_scan_int: } }
2168 \cs_new:Npn \_unravel_thing_last_item:
2169 {
2170   \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
2171     {
2172       \int_case:nnF \l__unravel_head_char_int
2173         {
2174           { 1 } { 1 } % lastkern
2175           { 2 } { 2 } % lastskip
2176         }
2177         { 0 } % other integer parameters
2178       }
2179     {
2180       \int_case:nnF \l__unravel_head_char_int

```

```

2181     {
2182         { 26 } { 0 \__unravel_scan_normal_glue: } % gluestretchorder
2183         { 27 } { 0 \__unravel_scan_normal_glue: } % glueshrinkorder
2184         { 28 } % fontcharwd
2185         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2186         { 29 } % fontcharht
2187         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2188         { 30 } % fontchardp
2189         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2190         { 31 } % fontcharic
2191         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2192         { 32 } { 1 \__unravel_scan_int: } % parshapelength
2193         { 33 } { 1 \__unravel_scan_int: } % parshapeindent
2194         { 34 } { 1 \__unravel_scan_int: } % parshapedimen
2195         { 35 } { 1 \__unravel_scan_normal_glue: } % gluestretch
2196         { 36 } { 1 \__unravel_scan_normal_glue: } % glueshrink
2197         { 37 } { 2 \__unravel_scan_mu_glue: } % mutoglu
2198         { 38 } { 3 \__unravel_scan_normal_glue: } % gluetomu
2199         { 39 } % numepr
2200         { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
2201         { 40 } % dimexpr
2202         { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
2203         { 41 } % glueexpr
2204         { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
2205         { 42 } % muexpr
2206         { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
2207     }
2208     { }
2209 }
2210 }
2211 \cs_new:Npn \__unravel_thing_register:
2212 {
2213     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
2214     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
2215     { \__unravel_scan_int: }
2216 }

```

(End of definition for `\__unravel_thing_case:`, `\__unravel_thing_last_item:`, and `\__unravel_thing_register:`.)

`\__unravel_scan_toks_register:` A case where getting operands is not completely trivial.

```

2217 \cs_new_protected:Npn \__unravel_scan_toks_register:
2218 {
2219     \int_compare:nNnT \l__unravel_head_char_int = 0
2220     { \__unravel_scan_int: }
2221 }

```

(End of definition for `\__unravel_scan_toks_register:`.)

`\__unravel_thing_use_get:nnNM` Given a level found #1 and a target level #2 (both in [0,3]), turn the token list #3 into the desired level or less, and store the result in #4.

```

2222 \cs_new_protected:Npn \__unravel_thing_use_get:nnNM #1#2#3#4
2223 {
2224     \int_compare:nNnTF {#2} < { 3 }
2225     {

```

```

2226     \int_compare:nNnT {#1} = { 3 }
2227     { \__unravel_tex_error:nV { incompatible-units } #3 }
2228     \tl_set:Ne #4
2229     {
2230       \int_case:nn { \int_min:nn {#1} {#2} }
2231       {
2232         { 0 } \int_eval:n
2233         { 1 } \dim_eval:n
2234         { 2 } \skip_eval:n
2235       }
2236       { \int_compare:nNnT {#1} = { 3 } \tex_mutogluue:D #3 }
2237     }
2238   }
2239   {
2240     \int_case:nnF {#1}
2241     {
2242       { 0 } { \tl_set:Ne #4 { \int_eval:n {#3} } }
2243       { 3 } { \tl_set:Ne #4 { \muskip_eval:n {#3} } }
2244     }
2245     {
2246       \__unravel_tex_error:nV { incompatible-units } #3
2247       \tl_set:Ne #4 { \muskip_eval:n { \tex_gluetomu:D #3 } }
2248     }
2249   }
2250 }

```

*(End of definition for \\_\_unravel\_thing\_use\_get:nnNN.)*

```

\__unravel_scan_expr:N
\__unravel_scan_expr_aux:NN
\__unravel_scan_factor:N
2251 \cs_new_protected:Npn \__unravel_scan_expr:N #1
2252 { \__unravel_scan_expr_aux:NN #1 \c_false_bool }
2253 \cs_new_protected:Npn \__unravel_scan_expr_aux:NN #1#2
2254 {
2255   \__unravel_get_x_non_blank:
2256   \__unravel_scan_factor:N #1
2257   \__unravel_scan_expr_op:NN #1#2
2258 }
2259 \cs_new_protected:Npn \__unravel_scan_expr_op:NN #1#2
2260 {
2261   \__unravel_get_x_non_blank:
2262   \token_case_meaning:NnF \l__unravel_head_tl
2263   {
2264     \c__unravel_plus_tl
2265     {
2266       \__unravel_prev_input:V \l__unravel_head_tl
2267       \__unravel_scan_expr_aux:NN #1#2
2268     }
2269     \c__unravel_minus_tl
2270     {
2271       \__unravel_prev_input:V \l__unravel_head_tl
2272       \__unravel_scan_expr_aux:NN #1#2
2273     }
2274     \c__unravel_times_tl
2275     {

```



```

2276         \_unravel_prev_input:V \l_unravel_head_tl
2277         \_unravel_get_x_non_blank:
2278         \_unravel_scan_factor:N \_unravel_scan_int:
2279         \_unravel_scan_expr_op:NN #1#2
2280     }
2281 \c_unravel_over_tl
2282 {
2283     \_unravel_prev_input:V \l_unravel_head_tl
2284     \_unravel_get_x_non_blank:
2285     \_unravel_scan_factor:N \_unravel_scan_int:
2286     \_unravel_scan_expr_op:NN #1#2
2287 }
2288 \c_unravel_rp_tl
2289 {
2290     \bool_if:NTF #2
2291     { \_unravel_prev_input:V \l_unravel_head_tl }
2292     { \_unravel_back_input: }
2293 }
2294 }
2295 {
2296     \bool_if:NTF #2
2297     {
2298         \_unravel_error:nnnnn { missing-rparen } { } { } { } { }
2299         \_unravel_back_input:
2300         \_unravel_prev_input:V \c_unravel_rp_tl
2301     }
2302     {
2303         \token_if_eq_meaning:NMF \l_unravel_head_token \scan_stop:
2304         { \_unravel_back_input: }
2305     }
2306 }
2307 }
2308 \cs_new_protected:Npn \_unravel_scan_factor:N #1
2309 {
2310     \tl_if_eq:NNTF \l_unravel_head_tl \c_unravel_lp_tl
2311     {
2312         \_unravel_prev_input:V \l_unravel_head_tl
2313         \_unravel_scan_expr_aux:NN #1 \c_true_bool
2314     }
2315     {
2316         \_unravel_back_input:
2317         #1
2318     }
2319 }

```

(End of definition for \\_unravel\_scan\_expr:N, \\_unravel\_scan\_expr\_aux:NN, and \\_unravel\_scan\_factor:N.)

\\_unravel\_scan\_signs: Skips blanks, scans signs, and places them to the right of the last item of \\_unravel\_prev\_input:n.

```

2320 \cs_new_protected:Npn \_unravel_scan_signs:
2321 {
2322     \_unravel_get_x_non_blank:
2323     \tl_if_eq:NNTF \l_unravel_head_tl \c_unravel_plus_tl

```

```

2324     {
2325     \__unravel_prev_input:V \l__unravel_head_tl
2326     \__unravel_scan_signs:
2327     }
2328     {
2329     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
2330     {
2331     \__unravel_prev_input:V \l__unravel_head_tl
2332     \__unravel_scan_signs:
2333     }
2334     }
2335 }

```

(End of definition for \\_\_unravel\_scan\_signs:.)

```

\__unravel_scan_int:
\__unravel_scan_int_char: 2336 \cs_new_protected:Npn \__unravel_scan_int:
\__unravel_scan_int_lq: 2337 {
\__unravel_scan_int_explicit:n 2338 \__unravel_scan_signs:
2339 \__unravel_set_cmd:
2340 \__unravel_cmd_if_internal:TF
2341 { \__unravel_rescan_something_internal:n { 0 } }
2342 { \__unravel_scan_int_char: }
2343 }
2344 \cs_new_protected:Npn \__unravel_scan_int_char:
2345 {
2346 \token_case_meaning:NnF \l__unravel_head_tl
2347 {
2348 \c__unravel_lq_tl { \__unravel_scan_int_lq: }
2349 \c__unravel_rq_tl
2350 {
2351 \__unravel_prev_input:V \l__unravel_head_tl
2352 \__unravel_get_x_next:
2353 \__unravel_scan_int_explicit:Nn \c_false_bool { ' }
2354 }
2355 \c__unravel_dq_tl
2356 {
2357 \__unravel_prev_input:V \l__unravel_head_tl
2358 \__unravel_get_x_next:
2359 \__unravel_scan_int_explicit:Nn \c_false_bool { " } % "
2360 }
2361 }
2362 { \__unravel_scan_int_explicit:Nn \c_false_bool { } }
2363 }
2364 \cs_new_protected:Npn \__unravel_scan_int_lq:
2365 {
2366 \__unravel_get_next:
2367 \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2368 {
2369 \tl_set:Ne \l__unravel_head_tl
2370 { \__unravel_token_to_char:N \l__unravel_head_token }
2371 }
2372 \tl_set:Ne \l__unravel_tmpa_tl
2373 { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }

```

```

2374     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2375     \__unravel_print_action:e
2376     { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2377     \__unravel_skip_optional_space:
2378 }
2379 \cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
2380 {
2381     \if_int_compare:w 1
2382     < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2383     \exp_after:wN \use_i:nn
2384     \else:
2385     \exp_after:wN \use_ii:nn
2386     \fi:
2387     {
2388     \__unravel_prev_input:V \l__unravel_head_tl
2389     \__unravel_get_x_next:
2390     \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
2391     }
2392     {
2393     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2394     { \__unravel_back_input: }
2395     \bool_if:NF #1
2396     {
2397     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2398     \__unravel_prev_input:n { 0 }
2399     }
2400     }
2401 }

```

(End of definition for \\_\_unravel\_scan\_int: and others.)

\\_\_unravel\_scan\_normal\_dimen:

```

2402 \cs_new_protected:Npn \__unravel_scan_normal_dimen:
2403 { \__unravel_scan_dimen:nN { 2 } \c_false_bool }

```

(End of definition for \\_\_unravel\_scan\_normal\_dimen:.)

\\_\_unravel\_scan\_dimen:nN

The first argument is 2 if the unit may not be mu and 3 if the unit must be mu (or fil). The second argument is \c\_true\_bool if fil, fill, filll are permitted, and is otherwise false. These arguments are similar to those of TEX's own scan\_dimen procedure, in which mu is bool(#1=3) and inf is #2. The third argument of this procedure is omitted here, as the corresponding shortcut is provided as a separate function, \\_\_unravel\_scan\_dim\_unit:nN.

Ideally, \\_\_unravel\_scan\_inf\_unit\_loop: would produce an unravel error when reaching the third "L", rather than letting TeX produce the error later on.

```

2404 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2405 {
2406     \__unravel_scan_signs:
2407     \__unravel_prev_input_gpush:
2408     \__unravel_set_cmd:
2409     \__unravel_cmd_if_internal:TF
2410     {
2411     \int_compare:nNnTF {#1} = { 3 }
2412     { \__unravel_rescan_something_internal:n { 3 } }

```

```

2413         { \_unravel_rescan_something_internal:n { 1 } }
2414         \int_compare:nNnT \g_unravel_val_level_int = { 0 }
2415         { \_unravel_scan_dim_unit:nN {#1} #2 }
2416     }
2417     { \_unravel_scan_dimen_char:nN {#1} #2 }
2418     \_unravel_prev_input_gpop:N \l_unravel_head_tl
2419     \_unravel_prev_input_silent:V \l_unravel_head_tl
2420 }
2421 \cs_new_protected:Npn \_unravel_scan_dimen_char:nN #1#2
2422 {
2423     \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_comma_tl
2424     { \tl_set_eq:NN \l_unravel_head_tl \c_unravel_point_tl }
2425     \tl_if_eq:NNTF \l_unravel_head_tl \c_unravel_point_tl
2426     {
2427         \_unravel_prev_input:n { . }
2428         \_unravel_scan_decimal_loop:
2429     }
2430     {
2431         \_unravel_tl_if_in:ooTF { 0123456789 } \l_unravel_head_tl
2432         {
2433             \_unravel_back_input:
2434             \_unravel_scan_int:
2435             \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_comma_tl
2436             { \tl_set_eq:NN \l_unravel_head_tl \c_unravel_point_tl }
2437             \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_point_tl
2438             {
2439                 \_unravel_input_gpop:N \l_unravel_tmpb_gtl
2440                 \_unravel_prev_input:n { . }
2441                 \_unravel_scan_decimal_loop:
2442             }
2443         }
2444         {
2445             \_unravel_back_input:
2446             \_unravel_scan_int:
2447         }
2448     }
2449     \_unravel_scan_dim_unit:nN {#1} #2
2450 }
2451 \cs_new_protected:Npn \_unravel_scan_dim_unit:nN #1#2
2452 {
2453     \bool_if:NT #2
2454     {
2455         \_unravel_scan_keyword:nT { fFiLL }
2456         {
2457             \_unravel_scan_inf_unit_loop:
2458             \_unravel_break:w
2459         }
2460     }
2461     \_unravel_get_x_non_blank:
2462     \_unravel_set_cmd:
2463     \_unravel_cmd_if_internal:TF
2464     {
2465         \_unravel_prev_input_gpush:
2466         \_unravel_rescan_something_internal:n {#1}

```

```

2467     \int_compare:nNnTF \g__unravel_val_level_int = { 0 }
2468     { \__unravel_prev_input_join_get:nnN {#1} { sp } \l__unravel_tmpa_tl }
2469     { \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl }
2470     \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2471     \exp_after:wN \use_none:n \__unravel_break:w
2472   }
2473   { }
2474   \__unravel_back_input:
2475   \int_compare:nNnT {#1} = { 3 }
2476   {
2477     \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2478     \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
2479     \__unravel_prev_input:n { mu }
2480     \__unravel_break:w
2481   }
2482   \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2483   \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2484   \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2485   \__unravel_scan_keyword:nT { tTrRuUeE }
2486   { \__unravel_prepare_mag: }
2487   \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2488   \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2489   \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2490   \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }
2491   \__unravel_scan_keyword:nT { mMmM } { \__unravel_break:w }
2492   \__unravel_scan_keyword:nT { bBpP } { \__unravel_break:w }
2493   \__unravel_scan_keyword:nT { dDdD } { \__unravel_break:w }
2494   \__unravel_scan_keyword:nT { cCcC } { \__unravel_break:w }
2495   \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
2496   \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
2497   \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
2498   \__unravel_tex_error:nV { missing-pt } \l__unravel_head_tl
2499   \__unravel_prev_input:n { pt }
2500   \__unravel_break_point:
2501   \__unravel_skip_optional_space:
2502 }
2503 \cs_new_protected:Npn \__unravel_scan_inf_unit_loop:
2504 { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
2505 \cs_new_protected:Npn \__unravel_scan_decimal_loop:
2506 {
2507   \__unravel_get_x_next:
2508   \tl_if_empty:NTF \l__unravel_head_tl
2509     { \use_ii:nn }
2510     { \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl }
2511     {
2512       \__unravel_prev_input:V \l__unravel_head_tl
2513       \__unravel_scan_decimal_loop:
2514     }
2515     {
2516       \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
2517       { \__unravel_back_input: }
2518       \__unravel_prev_input_silent:n { ~ }
2519     }
2520 }

```

(End of definition for \\_unravel\_scan\_dimen:nN.)

```
\_unravel_scan_normal_glue:  
  \_unravel_scan_mu_glue: 2521 \cs_new_protected:Npn \_unravel_scan_normal_glue:  
    { \_unravel_scan_glue:n { 2 } }  
2522  
2523 \cs_new_protected:Npn \_unravel_scan_mu_glue:  
2524 { \_unravel_scan_glue:n { 3 } }
```

(End of definition for \\_unravel\_scan\_normal\_glue: and \\_unravel\_scan\_mu\_glue:.)

```
\_unravel_scan_glue:n  
2525 \cs_new_protected:Npn \_unravel_scan_glue:n #1  
2526 {  
2527   \_unravel_prev_input_gpush:  
2528   \_unravel_scan_signs:  
2529   \_unravel_prev_input_gpush:  
2530   \_unravel_set_cmd:  
2531   \_unravel_cmd_if_internal:TF  
2532   {  
2533     \_unravel_rescan_something_internal:n {#1}  
2534     \int_case:nnF \g__unravel_val_level_int  
2535     {  
2536       { 0 } { \_unravel_scan_dim_unit:nN {#1} \c_false_bool }  
2537       { 1 } { }  
2538     }  
2539     { \_unravel_break:w }  
2540   }  
2541   { \_unravel_back_input: \_unravel_scan_dimen:nN {#1} \c_false_bool }  
2542   \_unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl  
2543   \_unravel_prev_input_gpush:  
2544   \_unravel_prev_input_gpush:N \l__unravel_tmpa_tl  
2545   \_unravel_scan_keyword:nT { pPlLuUsS }  
2546   { \_unravel_scan_dimen:nN {#1} \c_true_bool }  
2547   \_unravel_scan_keyword:nT { mMiInNuUsS }  
2548   { \_unravel_scan_dimen:nN {#1} \c_true_bool }  
2549   \_unravel_break_point:  
2550   \_unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl  
2551   \_unravel_prev_input_silent:V \l__unravel_tmpa_tl  
2552 }
```

(End of definition for \\_unravel\_scan\_glue:n.)

```
\_unravel_scan_file_name:  
2553 \cs_new_protected:Npn \_unravel_scan_file_name:  
2554 {  
2555   \_unravel_get_x_non_relax:  
2556   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token  
2557   { \_unravel_scan_group_x:N \c_false_bool }  
2558   {  
2559     \_unravel_back_input:  
2560     \bool_gset_true:N \g__unravel_name_in_progress_bool  
2561     \bool_gset_false:N \g__unravel_quotes_bool  
2562     \_unravel_get_x_non_blank:  
2563     \_unravel_scan_file_name_loop:
```

```

2564     \bool_gset_false:N \g__unravel_name_in_progress_bool
2565     \__unravel_prev_input_silent:n { ~ }
2566   }
2567 }
2568 \cs_new_protected:Npn \__unravel_scan_file_name_loop:
2569 {
2570   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2571   { \__unravel_back_input: }
2572   {
2573     \tl_set:Ne \l__unravel_tmpa_tl
2574     { \__unravel_token_to_char:N \l__unravel_head_token }
2575     \tl_if_eq:NNT \l__unravel_tmpa_tl \c__unravel_dq_tl
2576     {
2577       \bool_if:NTF \g__unravel_quotes_bool
2578       { \bool_set_false:N } { \bool_set_true:N } \g__unravel_quotes_bool
2579     }
2580     \bool_if:NTF \g__unravel_quotes_bool
2581     { \use:n } { \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl }
2582     {
2583       \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2584       \__unravel_get_x_next:
2585       \__unravel_scan_file_name_loop:
2586     }
2587   }
2588 }
2589 \bool_new:N \g__unravel_quotes_bool

```

(End of definition for \\_\_unravel\_scan\_file\_name:.)

\\_\_unravel\_scan\_r\_token: This is analogous to TeX's `get_r_token`. We store in `\l__unravel_defined_tl` the token which we found, as this is what will be defined by the next assignment.

```

2590 \cs_new_protected:Npn \__unravel_scan_r_token:
2591 {
2592   \bool_do_while:nn
2593   { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2594   { \__unravel_get_next: }
2595   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2596   {
2597     \__unravel_error:nnnnn { missing-cs } { } { } { } { }
2598     \__unravel_back_input:
2599     \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2600   }
2601   \__unravel_prev_input_silent:V \l__unravel_head_tl
2602   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2603 }

```

(End of definition for \\_\_unravel\_scan\_r\_token:.)

\\_\_unravel\_scan\_toks\_to\_str:

```

2604 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2605 {
2606   \__unravel_prev_input_gpush:
2607   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2608   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl

```

```

2609     \__unravel_prev_input_silent:e
2610     { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2611 }

```

*(End of definition for \\_\_unravel\_scan\_toks\_to\_str:.)*

\\_\_unravel\_scan\_pdf\_ext\_toks:

```

2612 \cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
2613 {
2614     \__unravel_prev_input_gpush:
2615     \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2616     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2617     \__unravel_prev_input_silent:e
2618     { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2619 }

```

*(End of definition for \\_\_unravel\_scan\_pdf\_ext\_toks:.)*

\\_\_unravel\_scan\_toks:NN The boolean #1 is true if we are making a definition (then we start by scanning the parameter text), false if we are simply scanning a general text. The boolean #2 is true if we need to expand, false otherwise (for instance for \lowercase).

```

2620 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2621 {
2622     \bool_if:NT #1 { \__unravel_scan_param: }
2623     \__unravel_scan_left_brace:
2624     \bool_if:NTF #2
2625     { \__unravel_scan_group_x:N #1 }
2626     { \__unravel_scan_group_n:N #1 }
2627 }

```

*(End of definition for \\_\_unravel\_scan\_toks:NN.)*

\\_\_unravel\_scan\_param: Collect the parameter text into \l\_\_unravel\_tmpa\_tl, and when seeing either a begin-group or an end-group character, put it back into the input, stop looping, and put what we collected into \l\_\_unravel\_defining\_tl and into the prev\_input.

```

2628 \cs_new_protected:Npn \__unravel_scan_param:
2629 {
2630     \tl_clear:N \l__unravel_tmpa_tl
2631     \__unravel_scan_param_aux:
2632     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2633     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2634 }
2635 \cs_new_protected:Npn \__unravel_scan_param_aux:
2636 {
2637     \__unravel_get_next:
2638     \tl_concat:NNN \l__unravel_tmpa_tl
2639     \l__unravel_tmpa_tl \l__unravel_head_tl
2640     \tl_if_empty:NTF \l__unravel_head_tl
2641     { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2642 }

```

*(End of definition for \\_\_unravel\_scan\_param: and \\_\_unravel\_scan\_param\_aux:.)*



`\__unravel_scan_group_n:N` The boolean #1 is true if we are making a definition, false otherwise. In both cases put the open brace back and grab the first item. The only difference is that when making a definition we store the data into `\l__unravel_defining_tl` as well.

```

2643 \cs_new_protected:Npn \__unravel_scan_group_n:N #1
2644 {
2645   \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
2646   \__unravel_back_input:
2647   \__unravel_input_gpop_item:NF \l__unravel_head_tl
2648   {
2649     \__unravel_error:nnnnn { runaway-text } { } { } { } { }
2650     \__unravel_exit_hard:w
2651   }
2652   \tl_set:Ne \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2653   \bool_if:NT #1
2654   { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2655   \__unravel_prev_input_silent:V \l__unravel_head_tl
2656 }

```

*(End of definition for \\_\_unravel\_scan\_group\_n:N.)*

`\__unravel_scan_group_x:N` The boolean #1 is true if we are making a definition, false otherwise.

```

2657 \cs_new_protected:Npn \__unravel_scan_group_x:N #1
2658 {
2659   \__unravel_input_gpop_tl:N \l__unravel_head_tl
2660   \__unravel_back_input:V \l__unravel_head_tl
2661   \bool_if:NTF #1
2662   {
2663     \__unravel_prev_input_silent:V \c_left_brace_str
2664     \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2665     \__unravel_scan_group_xdef:n { 1 }
2666   }
2667   {
2668     \__unravel_prev_input_gpush_gtl:
2669     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2670     \__unravel_scan_group_x:n { 1 }
2671     \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2672     \__unravel_prev_input_silent:e
2673     { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2674   }
2675 }

```

*(End of definition for \\_\_unravel\_scan\_group\_x:N.)*

`\__unravel_scan_group_xdef:n` This is to scan the replacement text of an `\edef` or `\xdef`. The integer #1 counts the brace balance.

```

2676 \cs_new_protected:Npn \__unravel_scan_group_xdef:n #1
2677 {
2678   \__unravel_get_token_xdef:
2679   \tl_if_empty:NTF \l__unravel_head_tl
2680   {
2681     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2682     {
2683       \__unravel_prev_input_silent:V \c_left_brace_str
2684       \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }

```

```

2685     \_unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2686   }
2687   {
2688     \_unravel_prev_input_silent:V \c_right_brace_str
2689     \tl_put_right:Nn \l_unravel_defining_tl { \if_false: { \fi: } }
2690     \int_compare:nNnF {#1} = 1
2691     { \_unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2692   }
2693 }
2694 {
2695   \_unravel_prev_input_silent:V \l_unravel_head_tl
2696   \tl_put_right:Ne \l_unravel_defining_tl
2697   { \exp_not:N \exp_not:N \exp_not:V \l_unravel_head_tl }
2698   \_unravel_scan_group_xdef:n {#1}
2699 }
2700 }
2701 \cs_generate_variant:Nn \_unravel_scan_group_xdef:n { f }

```

(End of definition for \\_unravel\_scan\_group\_xdef:n.)

\\_unravel\_scan\_group\_x:n

```

2702 \cs_new_protected:Npn \_unravel_scan_group_x:n #1
2703   {
2704     \_unravel_get_token_x:
2705     \_unravel_prev_input_gtl:N \l_unravel_head_gtl
2706     \tl_if_empty:NTF \l_unravel_head_tl
2707     {
2708       \gtl_if_head_is_group_begin:NTF \l_unravel_head_gtl
2709       { \_unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2710       {
2711         \int_compare:nNnF {#1} = 1
2712         { \_unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2713       }
2714     }
2715     { \_unravel_scan_group_x:n {#1} }
2716   }
2717 \cs_generate_variant:Nn \_unravel_scan_group_x:n { f }

```

(End of definition for \\_unravel\_scan\_group\_x:n.)

\\_unravel\_scan\_alt\_rule:

```

2718 \cs_new_protected:Npn \_unravel_scan_alt_rule:
2719   {
2720     \_unravel_scan_keyword:nTF { wWiIdDtThH }
2721     {
2722       \_unravel_scan_normal_dimen:
2723       \_unravel_scan_alt_rule:
2724     }
2725     {
2726       \_unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2727       {
2728         \_unravel_scan_normal_dimen:
2729         \_unravel_scan_alt_rule:
2730       }
2731     }

```

```

2732         \__unravel_scan_keyword:nT { dDeEpPtThH }
2733         {
2734             \__unravel_scan_normal_dimen:
2735             \__unravel_scan_alt_rule:
2736         }
2737     }
2738 }
2739 }

```

(End of definition for \\_\_unravel\_scan\_alt\_rule:.)

\\_\_unravel\_scan\_spec: Some T<sub>E</sub>X primitives accept the keywords `to` and `spread`, followed by a dimension.

```

2740 \cs_new_protected:Npn \__unravel_scan_spec:
2741 {
2742     \__unravel_scan_keyword:nTF { tTo0 } { \__unravel_scan_normal_dimen: }
2743     {
2744         \__unravel_scan_keyword:nT { sSpPrReEaAdD }
2745         { \__unravel_scan_normal_dimen: }
2746     }
2747     \__unravel_scan_left_brace:
2748 }

```

(End of definition for \\_\_unravel\_scan\_spec:.)

## 2.8 Working with boxes

\\_\_unravel\_do\_box:N When this procedure is called, the last item in the previous-input sequence is

- empty if the box is meant to be put in the input stream,
- `\setbox<int>` if it is meant to be stored somewhere,
- `\moveright<dim>`, `\moveleft<dim>`, `\lower<dim>`, `\raise<dim>` if it is meant to be shifted,
- `\leaders` or `\cleaders` or `\xleaders`, in which case the argument is `\c_true_bool` (otherwise `\c_false_bool`).

If a `make_box` command follows, we fetch the operands. If leaders are followed by a rule, then this is also ok. In all other cases, call `\__unravel_do_box_error:` to clean up.

```

2749 \cs_new_protected:Npn \__unravel_do_box:N #1
2750 {
2751     \__unravel_get_x_non_relax:
2752     \__unravel_set_cmd:
2753     \int_compare:nNnTF
2754     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2755     { \__unravel_do_begin_box:N #1 }
2756     {
2757         \bool_if:NTF #1
2758         {
2759             \int_case:nnTF \l__unravel_head_cmd_int
2760             {
2761                 { \__unravel_tex_use:n { hrule } } { }
2762                 { \__unravel_tex_use:n { vrule } } { }
2763             }

```

```

2764             { \__unravel_do_leaders_rule: }
2765             { \__unravel_do_box_error: }
2766         }
2767     { \__unravel_do_box_error: }
2768 }
2769 }

```

(End of definition for \\_\_unravel\_do\_box:N.)

\\_\_unravel\_do\_box\_error: Put the (non-make\_box) command back into the input and complain. Then recover by throwing away the action (last item of the previous-input sequence). For some reason (this appears to be what T<sub>E</sub>X does), there is no need to remove the after assignment token here.

```

2770 \cs_new_protected:Npn \__unravel_do_box_error:
2771 {
2772     \__unravel_back_input:
2773     \__unravel_error:nnnn { missing-box } { } { } { } { }
2774     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2775     \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
2776 }

```

(End of definition for \\_\_unravel\_do\_box\_error:.)

\\_\_unravel\_do\_begin\_box:N We have just found a make\_box command and placed it into the last item of the previous-input sequence. If it is “simple” (\box<int>, \copy<int>, \lastbox, \vsplit<int> to <dim>) then we grab its operands, then call \\_\_unravel\_do\_simple\_box:N to finish up. If it is \vtop or \vbox or \hbox, we need to work harder.

```

2777 \cs_new_protected:Npn \__unravel_do_begin_box:N #1
2778 {
2779     \__unravel_prev_input:V \l__unravel_head_tl
2780     \int_case:nnTF \l__unravel_head_char_int
2781     {
2782         { 0 } { \__unravel_scan_int: } % box
2783         { 1 } { \__unravel_scan_int: } % copy
2784         { 2 } { } % lastbox
2785         { 3 } % vsplit
2786         {
2787             \__unravel_scan_int:
2788             \__unravel_scan_to:
2789             \__unravel_scan_normal_dimen:
2790         }
2791     }
2792     { \__unravel_do_simple_box:N #1 }
2793     { \__unravel_do_box_explicit:N #1 }
2794 }

```

(End of definition for \\_\_unravel\_do\_begin\_box:N.)

\\_\_unravel\_do\_simple\_box:N For leaders, we need to fetch a glue. In all cases, retrieve the box construction (such as \raise3pt\vsplit7to5em). Finally, let T<sub>E</sub>X run the code and print what we have done. In the case of \shipout, check that \mag has a value between 1 and 32768.

```

2795 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2796 {
2797     \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }

```

```

2798     {
2799       \__unravel_prev_input_gpop:N \l__unravel_head_tl
2800       \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2801         { \__unravel_prepare_mag: }
2802       \tl_use:N \l__unravel_head_tl \scan_stop:
2803       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2804       \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
2805     }
2806   }

```

(End of definition for \\_\_unravel\_do\_simple\_box:N.)

\\_\_unravel\_do\_leaders\_fetch\_skip:

```

2807 \cs_new_protected:Npn \__unravel_do_leaders_fetch_skip:
2808   {
2809     \__unravel_get_x_non_relax:
2810     \__unravel_set_cmd:
2811     \int_compare:nNnTF \l__unravel_head_cmd_int
2812       = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2813     {
2814       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2815       \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2816       \__unravel_do_append_glue:
2817     }
2818     {
2819       \__unravel_back_input:
2820       \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2821       \__unravel_prev_input_gpop:N \l__unravel_head_tl
2822       \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
2823     }
2824   }

```

(End of definition for \\_\_unravel\_do\_leaders\_fetch\_skip:.)

\\_\_unravel\_do\_box\_explicit:N

At this point, the last item in the previous-input sequence is typically `\setbox0\hbox` or `\raise 3pt\hbox`. Scan for keywords `to` and `spread` and a left brace. Install a hook in `\everyhbox` or `\everyvbox` (whichever T<sub>E</sub>X is going to insert in the box). We then retrieve all the material that led to the current box into `\l__unravel_head_tl` in order to print it, then let T<sub>E</sub>X perform the box operation (here we need to provide the begin-group token, as it was scanned but not placed in the previous-input sequence). T<sub>E</sub>X inserts `\everyhbox` or `\everyvbox` just after the begin-group token, and the hook we did is such that all that material is collected and put into the input that we will study. We must remember to find a glue for leaders, and for this we use a stack of letters `v`, `h` for vertical/horizontal leaders, and `Z` for normal boxes.

```

2825 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2826   {
2827     \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2828       { \__unravel_box_hook:N \tex_everyhbox:D }
2829       { \__unravel_box_hook:N \tex_everyvbox:D }
2830     \__unravel_scan_spec:
2831     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2832     \__unravel_set_action_text:e
2833       { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2834     \seq_push:Nf \l__unravel_leaders_box_seq

```

```

2835     { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2836 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2837 \gtl_gconcat:NNN \g__unravel_output_gtl
2838   \g__unravel_output_gtl \c_group_begin_gtl
2839 \tl_use:N \l__unravel_head_tl
2840   \c_group_begin_token \__unravel_box_hook_end:
2841 }

```

(End of definition for \\_\_unravel\_do\_box\_explicit:N.)

\\_\_unravel\_box\_hook:N Used to capture the contents of an \everyhbox or similar, without altering \everyhbox too much (just add one token at the start). The various o-expansions remove \prg\_do\_-\\_\_unravel\_box\_hook:w nothing:, used to avoid losing braces.

```

2842 \cs_new_protected:Npn \__unravel_box_hook:N #1
2843 {
2844   \tl_set:NV \l__unravel_tmpa_tl #1
2845   \str_if_eq:eeF
2846     { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2847   {
2848     \exp_args:Ne #1
2849     {
2850       \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2851       \exp_not:V #1
2852     }
2853   }
2854 \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2855 {
2856   \exp_args:No #1 {##1}
2857   \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2858   \gtl_clear:N \l__unravel_after_group_gtl
2859   \__unravel_print_action:
2860   \__unravel_back_input:o {##1}
2861   \__unravel_set_action_text:e
2862     { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2863   \tl_if_empty:oF {##1} { \__unravel_print_action: }
2864 }
2865 }
2866 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2867 \cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:

```

(End of definition for \\_\_unravel\_box\_hook:N, \\_\_unravel\_box\_hook:w, and \\_\_unravel\_box\_hook\_end:.)

\\_\_unravel\_do\_leaders\_rule: After finding a vrule or hrule command and looking for depth, heigh and width keywords, we are in the same situation as after finding a box. Fetch the required skip accordingly.

```

2868 \cs_new_protected:Npn \__unravel_do_leaders_rule:
2869 {
2870   \__unravel_prev_input:V \l__unravel_head_tl
2871   \__unravel_scan_alt_rule:
2872   \__unravel_do_leaders_fetch_skip:
2873 }

```

(End of definition for \\_\_unravel\_do\_leaders\_rule:.)

## 2.9 Paragraphs

`\_unravel_charcode_if_safe:nTF`

```

2874 \prg_new_protected_conditional:Npnm \_unravel_charcode_if_safe:n #1 { TF }
2875 {
2876   \bool_if:nTF
2877     {
2878       \int_compare_p:n { #1 = '!' }
2879       || \int_compare_p:n { ' ' <= #1 <= '[' }
2880       || \int_compare_p:n { #1 = ']' }
2881       || \int_compare_p:n { ' ' <= #1 <= 'z' }
2882     }
2883     { \prg_return_true: }
2884     { \prg_return_false: }
2885   }

```

*(End of definition for `\_unravel_charcode_if_safe:nTF`.)*

`\_unravel_char:n`  
`\_unravel_char:V`  
`\_unravel_char:e`

```

2886 \cs_new_protected:Npn \_unravel_char:n #1
2887 {
2888   \tex_char:D #1 \scan_stop:
2889   \_unravel_charcode_if_safe:nTF {#1}
2890     {
2891       \tl_set:Nc \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } }
2892       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2893       \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
2894     }
2895     {
2896       \tl_set:Nc \l__unravel_tmpa_tl
2897         { \exp_not:N \char \int_eval:n {#1} ~ }
2898       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2899       \__unravel_print_action:e
2900         { " \char_generate:nn {#1} { 12 } " = \tl_to_str:N \l__unravel_tmpa_tl }
2901     }
2902   }
2903   \cs_generate_variant:Nn \_unravel_char:n { V , e }

```

*(End of definition for `\_unravel_char:n`.)*

`\_unravel_char_in_mmode:n`  
`\_unravel_char_in_mmode:V`  
`\_unravel_char_in_mmode:e`

```

2904 \cs_new_protected:Npn \_unravel_char_in_mmode:n #1
2905 {
2906   \int_compare:nNnTF { \tex_mathcode:D #1 }
2907     = { \sys_if_engine luatex:TF { "1000000 } { "8000 } }
2908     { % math active
2909       \_unravel_active_do:nn {#1} { \gtl_set:Nn \l__unravel_head_gtl }
2910       \_unravel_back_input:
2911       \__unravel_print_action:e
2912         { \char_generate:nn {#1} { 12 } ~ active }
2913     }
2914     { \_unravel_char:n {#1} }
2915   }
2916   \cs_generate_variant:Nn \_unravel_char_in_mmode:n { V , e }

```

(End of definition for `\__unravel_char_in_mmode:n`.)

```
\__unravel_mathchar:n
\__unravel_mathchar:e 2917 \cs_new_protected:Npn \__unravel_mathchar:n #1
2918 {
2919   \tex_mathchar:D #1 \scan_stop:
2920   \tl_set:Ne \l__unravel_tmpa_tl
2921     { \exp_not:N \mathchar " \int_to_hex:n {#1} ~ } % "
2922   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2923   \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
2924 }
2925 \cs_generate_variant:Nn \__unravel_mathchar:n { e }
```

(End of definition for `\__unravel_mathchar:n`.)

`\__unravel_new_graf:N` The argument is a boolean, indicating whether the paragraph should be indented. We have much less work to do here than `TeX` itself. Our only task is to correctly position the `\everypar` tokens in the input that we will read, rather than letting `TeX` run the code right away.

```
2926 \cs_new_protected:Npn \__unravel_new_graf:N #1
2927 {
2928   \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
2929   \__unravel_everypar:w { }
2930   \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2931   \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl
2932   \__unravel_back_input:V \l__unravel_tmpa_tl
2933   \__unravel_print_action:e
2934     {
2935       \g__unravel_action_text_str \c_space_tl : ~
2936       \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2937     }
2938 }
```

(End of definition for `\__unravel_new_graf:N`.)

`\__unravel_par_if_hmode:` This is like the `end_graf` procedure in `TeX`.

```
2939 \cs_new_protected:Npn \__unravel_par_if_hmode:
2940 { \mode_if_horizontal:T { \__unravel_par: } }
```

(End of definition for `\__unravel_par_if_hmode:.`)

`\__unravel_par:`

```
2941 \cs_new_protected:Npn \__unravel_par:
2942 {
2943   \tex_par:D
2944   \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2945   \__unravel_print_action:e { Paragraph~end. }
2946 }
```

(End of definition for `\__unravel_par:.`)

`\__unravel_build_page:`

```
2947 \cs_new_protected:Npn \__unravel_build_page:
2948 {
2949 }
```

(End of definition for `\__unravel_build_page:.`)



## 2.10 Groups

`\l__unravel_choice_int` Used by `\mathchoice` etc to keep track of which argument we are currently in.

```
2950 \int_new:N \l__unravel_choice_int
```

(End of definition for `\l__unravel_choice_int`.)

`\__unravel_handle_right_brace`: When an end-group character is sensed, the result depends on the current group type. Suppress the `after_group` tokens in `\discretionary` or `\mathchoice`.

```
2951 \cs_new_protected:Npn \__unravel_handle_right_brace:
2952 {
2953   \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
2954   {
2955     \gtl_gconcat:NNN \g__unravel_output_gtl
2956     \g__unravel_output_gtl \c_group_end_gtl
2957     \int_case:nnF \__unravel_currentgrouptype:
2958     {
2959       { 10 } { } % disc
2960       { 13 } { } % math_choice
2961     }
2962     { \__unravel_back_input_gtl:N \l__unravel_after_group_gtl }
2963     \int_case:nn \__unravel_currentgrouptype:
2964     {
2965       { 1 } { \__unravel_end_simple_group: } % simple
2966       { 2 } { \__unravel_end_box_group: } % hbox
2967       { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2968       { 4 } { \__unravel_par_if_hmode: \__unravel_end_box_group: } % vbox
2969       { 5 } { \__unravel_par_if_hmode: \__unravel_end_box_group: } % vtop
2970       { 6 } { \__unravel_end_align_group: } % align
2971       { 7 } { \__unravel_end_no_align_group: } % no_align
2972       { 8 } { \__unravel_end_output_group: } % output
2973       { 9 } { \__unravel_end_simple_group: } % math
2974       { 10 } { \__unravel_end_choice_group:NN 2 \discretionary } % disc
2975       { 11 } { \__unravel_par_if_hmode: \__unravel_end_simple_group: } % insert
2976       { 12 } { \__unravel_par_if_hmode: \__unravel_end_simple_group: } % vcenter
2977       { 13 } { \__unravel_end_choice_group:NN 3 \mathchoice } % math_choice
2978     }
2979   }
2980   { % bottom_level, semi_simple, math_shift, math_left
2981     \l__unravel_head_token
2982     \__unravel_print_action:
2983   }
2984 }
```

(End of definition for `\__unravel_handle_right_brace:`.)

`\__unravel_end_simple_group`: This command is used to simply end a group, when there are no specific operations to perform.

```
2985 \cs_new_protected:Npn \__unravel_end_simple_group:
2986 {
2987   \l__unravel_head_token
2988   \__unravel_print_action:
2989 }
```

(End of definition for `\__unravel_end_simple_group:`.)

`\__unravel_end_box_group:` The end of an explicit box (generated by `\vtop`, `\vbox`, or `\hbox`) can either be simple, or can mean that we need to find a skip for a `\leaders/\cleaders/\xleaders` construction.

```

2990 \cs_new_protected:Npn \__unravel_end_box_group:
2991 {
2992   \seq_pop:NN \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2993   \exp_args:No \__unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
2994 }
2995 \cs_new_protected:Npn \__unravel_end_box_group_aux:n #1
2996 {
2997   \str_if_eq:eeTF {#1} { Z }
2998   { \__unravel_end_simple_group: }
2999   {
3000     \__unravel_get_x_non_relax:
3001     \__unravel_set_cmd:
3002     \int_compare:nNnTF \l__unravel_head_cmd_int
3003     = { \__unravel_tex_use:n { #1 skip } }
3004     {
3005       \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
3006       \__unravel_do_append_glue:
3007     }
3008     {
3009       \__unravel_back_input:
3010       \c_group_end_token \group_begin: \group_end:
3011       \__unravel_print_action:
3012     }
3013   }
3014 }

```

*(End of definition for `\__unravel_end_box_group:.`)*

`\__unravel_end_align_group:`

```

3015 \cs_new_protected:Npn \__unravel_end_align_group:
3016 {
3017   \__unravel_not_implemented:n { end_align_group }
3018   \__unravel_end_simple_group:
3019 }

```

*(End of definition for `\__unravel_end_align_group:.`)*

`\__unravel_end_no_align_group:`

```

3020 \cs_new_protected:Npn \__unravel_end_no_align_group:
3021 {
3022   \__unravel_not_implemented:n { end_no_align_group }
3023   \__unravel_end_simple_group:
3024 }

```

*(End of definition for `\__unravel_end_no_align_group:.`)*

`\__unravel_end_output_group:`

```

3025 \cs_new_protected:Npn \__unravel_end_output_group:
3026 {
3027   \__unravel_not_implemented:n { end_output_group }
3028   \__unravel_end_simple_group:
3029 }

```

(End of definition for `\_unravel_end_output_group:`.)

`\_unravel_end_choice_group:NN`

`\_unravel_end_choice_group:nN`

```
3030 \cs_new_protected:Npn \_unravel_end_choice_group:NN #1#2
3031 {
3032   \int_compare:nNnTF \l__unravel_choice_int > {#1}
3033   {
3034     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
3035     \c_group_end_token
3036     \_unravel_print_action:e
3037     { \token_to_str:N #2 \prg_replicate:nn { #1 + 1 } { { ... } } }
3038   }
3039   { \exp_args:NV \_unravel_end_choice_group:nN \l__unravel_choice_int #2 }
3040 }
3041 \cs_new_protected:Npn \_unravel_end_choice_group:nN #1#2
3042 {
3043   \_unravel_scan_left_brace:
3044   \gtl_gconcat:NNN \g__unravel_output_gtl
3045   \g__unravel_output_gtl \c_group_begin_gtl
3046   \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
3047   \use:n \c_group_end_token
3048   \use:n \c_group_begin_token
3049   \int_set:Nn \l__unravel_choice_int { #1 + 1 }
3050   \gtl_clear:N \l__unravel_after_group_gtl
3051   \_unravel_print_action:e
3052   {
3053     \token_to_str:N #2
3054     \prg_replicate:nn {#1} { { ... } }
3055     \iow_char:N \{
3056   }
3057 }
```

(End of definition for `\_unravel_end_choice_group:NN` and `\_unravel_end_choice_group:nN`.)

`\_unravel_off_save:`

```
3058 \cs_new_protected:Npn \_unravel_off_save:
3059 {
3060   \int_compare:nNnTF \_unravel_currentgrouptype: = { 0 }
3061   { % bottom-level
3062     \_unravel_error:neeee { extra-close }
3063     { \token_to_meaning:N \l__unravel_head_token } { } { } { }
3064   }
3065   {
3066     \_unravel_back_input:
3067     \int_case:nnF \_unravel_currentgrouptype:
3068     {
3069       { 14 } % semi_simple_group
3070       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
3071       { 15 } % math_shift_group
3072       { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
3073       { 16 } % math_left_group
3074       { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
3075     }
3076     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }

```

```

3077     \__unravel_back_input:
3078     \__unravel_error:neeee { off-save }
3079     { \gtl_to_str:N \l__unravel_head_gtl } { } { } { }
3080   }
3081 }

```

(End of definition for \\_\_unravel\_off\_save:.)

## 2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
3082 \cs_new_protected:Npn \__unravel_mode_math:n #1
3083 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
3084 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
3085 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
3086 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
3087 {
3088   \mode_if_math:TF
3089   { \__unravel_insert_dollar_error: }
3090   { \mode_if_horizontal:TF { \__unravel_head_for_vmde: } {#1} }
3091 }
3092 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
3093 {
3094   \mode_if_vertical:TF
3095   { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
3096   {#1}
3097 }

```

(End of definition for \\_\_unravel\_mode\_math:n, \\_\_unravel\_mode\_non\_math:n, and \\_\_unravel\_mode\_vertical:n.)

\\_\_unravel\_head\_for\_vmde: See T<sub>E</sub>X's head\_for\_vmde.

```

3098 \cs_new_protected:Npn \__unravel_head_for_vmde:
3099 {
3100   \mode_if_inner:TF
3101   {
3102     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
3103     {
3104       \__unravel_error:nnnnn { hrulerule-bad-mode } { } { } { } { }
3105       \__unravel_print_action:
3106     }
3107     { \__unravel_off_save: }
3108   }
3109   {
3110     \__unravel_back_input:
3111     \gtl_set:Nn \l__unravel_head_gtl { \par }
3112     \__unravel_back_input:
3113   }
3114 }

```

(End of definition for \\_\_unravel\_head\_for\_vmde:.)

\\_\_unravel\_goto\_inner\_math:

```

3115 \cs_new_protected:Npn \__unravel_goto_inner_math:

```

```

3116 {
3117   \__unravel_box_hook:N \tex_everymath:D
3118   $ % $
3119   \__unravel_box_hook_end:
3120 }

```

(End of definition for \\_\_unravel\_goto\_inner\_math:.)

\\_\_unravel\_goto\_display\_math:

```

3121 \cs_new_protected:Npn \__unravel_goto_display_math:
3122 {
3123   \__unravel_box_hook:N \tex_everydisplay:D
3124   $ $
3125   \__unravel_box_hook_end:
3126 }

```

(End of definition for \\_\_unravel\_goto\_display\_math:.)

\\_\_unravel\_after\_math: In display math mode, or in a group started by \eqno or \leqno (namely in inner math mode with non-zero \l\_\_unravel\_choice\_int), search for another \$; otherwise simply close the inner math.

```

3127 \cs_new_protected:Npn \__unravel_after_math:
3128 {
3129   \mode_if_inner:TF
3130   { \int_compare:nNnTF \l__unravel_choice_int > 0 }
3131   { \use_i:nn }
3132   {
3133     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3134     \__unravel_get_x_next:
3135     \token_if_eq_catcode:NMF
3136     \l__unravel_head_token \c_math_toggle_token
3137     {
3138       \__unravel_back_input:
3139       \tl_set:Nn \l__unravel_head_tl { $ } % $
3140       \__unravel_error:nmnn { missing-dollar } { } { } { } { }
3141     }
3142     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3143     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3144     $ $
3145   }
3146   {
3147     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3148     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3149     $ % $
3150   }
3151   \__unravel_print_action:
3152 }

```

(End of definition for \\_\_unravel\_after\_math:.)

## 2.12 Commands

We will implement commands in order of their command codes (some of the more elaborate commands call auxiliaries defined in other sections). Some cases are forbidden.

`\__unravel_forbidden_case:`

```
3153 \cs_new_protected:Npn \__unravel_forbidden_case:  
3154 { \__unravel_tex_error:nV { forbidden-case } \l__unravel_head_tl }
```

*(End of definition for \\_\_unravel\_forbidden\_case:.)*

### 2.12.1 Characters: from 0 to 15

This section is about command codes in the range [0, 15].

- `relax=0` for `\relax`.
- `begin-group_char=1` for begin-group characters (catcode 1).
- `end-group_char=2` for end-group characters (catcode 2).
- `math_char=3` for math shift (math toggle in `expl3`) characters (catcode 3).
- `tab_mark=4` for `\span`
- `alignment_char=4` for alignment tab characters (catcode 4).
- `car_ret=5` for `\cr` and `\crcr`.
- `macro_char=6` for macro parameter characters (catcode 6).
- `superscript_char=7` for superscript characters (catcode 7).
- `subscript_char=8` for subscript characters (catcode 8).
- `endv=9` for ?.
- `blank_char=10` for blank spaces (catcode 10).
- `the_char=11` for letters (catcode 11).
- `other_char=12` for other characters (catcode 12).
- `par_end=13` for `\par`.
- `stop=14` for `\end` and `\dump`.
- `delim_num=15` for `\delimiter`.

Not implemented at all: `endv`.

`\relax` does nothing.

```
3155 \__unravel_new_tex_cmd:nn { relax } % 0  
3156 {  
3157   \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:  
3158   {  
3159     \exp_after:wN \__unravel_token_if_expandable:NTF \l__unravel_head_tl  
3160     {  
3161       \__unravel_set_action_text:e  
3162       { \iow_char:N \notexpanded: \g__unravel_action_text_str }  
3163     }  
3164     { }  
3165   }  
3166   \__unravel_print_action:  
3167 }
```

Begin-group characters are sent to the output, as their grouping behaviour may affect the scope of font changes, for instance. They are also performed.

```

3168 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
3169 {
3170   \gtl_gconcat:NNN \g__unravel_output_gtl
3171   \g__unravel_output_gtl \c_group_begin_gtl
3172   \__unravel_print_action:
3173   \l__unravel_head_token
3174   \gtl_clear:N \l__unravel_after_group_gtl
3175 }
3176 \__unravel_new_tex_cmd:nn { end-group_char } % 2
3177 { \__unravel_handle_right_brace: }

```

Math shift characters quit vertical mode, and start math mode.

```

3178 \__unravel_new_tex_cmd:nn { math_char } % 3
3179 {
3180   \__unravel_mode_non_vertical:n
3181   {
3182     \mode_if_math:TF
3183     {
3184       \int_compare:nNnTF
3185         \__unravel_currentgrouptype: = { 15 } % math_shift_group
3186         { \__unravel_after_math: }
3187         { \__unravel_off_save: }
3188     }
3189     {
3190       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3191       \__unravel_get_next:
3192       \token_if_eq_catcode:NNTF
3193         \l__unravel_head_token \c_math_toggle_token
3194         {
3195           \mode_if_inner:TF
3196             { \__unravel_back_input: \__unravel_goto_inner_math: }
3197             {
3198               \gtl_gput_right:NV
3199                 \g__unravel_output_gtl \l__unravel_head_tl
3200                 \__unravel_goto_display_math:
3201             }
3202           }
3203         { \__unravel_back_input: \__unravel_goto_inner_math: }
3204       }
3205     }
3206 }

```

Some commands are errors when they reach T<sub>E</sub>X's stomach. Among others, `tab_mark=alignment_char`, `car_ret` and `macro_char`. We let T<sub>E</sub>X insert the proper error.

```

3207 \__unravel_new_tex_cmd:nn { alignment_char } % 4
3208 { \l__unravel_head_token \__unravel_print_action: }
3209 \__unravel_new_tex_cmd:nn { car_ret } % 5
3210 { \l__unravel_head_token \__unravel_print_action: }
3211 \__unravel_new_tex_cmd:nn { macro_char } % 6
3212 { \l__unravel_head_token \__unravel_print_action: }

```

```

3213 \__unravel_new_tex_cmd:nn { superscript_char } % 7
3214 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3215 \__unravel_new_tex_cmd:nn { subscript_char } % 8
3216 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3217 \cs_new_protected:Npn \__unravel_sub_sup:
3218 {
3219 \__unravel_prev_input_gpush:N \l__unravel_head_tl
3220 \__unravel_print_action:
3221 \__unravel_do_one_atom:
3222 }
3223 \cs_new_protected:Npn \__unravel_do_one_atom:
3224 {
3225 \__unravel_get_x_non_relax:
3226 \__unravel_set_cmd:
3227 \int_case:nnTF \l__unravel_head_cmd_int
3228 {
3229 { \__unravel_tex_use:n { the_char } }
3230 { \__unravel_prev_input:V \l__unravel_head_tl }
3231 { \__unravel_tex_use:n { other_char } }
3232 { \__unravel_prev_input:V \l__unravel_head_tl }
3233 { \__unravel_tex_use:n { char_given } }
3234 { \__unravel_prev_input:V \l__unravel_head_tl }
3235 { \__unravel_tex_use:n { char_num } }
3236 {
3237 \__unravel_prev_input:V \l__unravel_head_tl
3238 \__unravel_scan_int:
3239 }
3240 { \__unravel_tex_use:n { math_char_num } }
3241 {
3242 \__unravel_prev_input:V \l__unravel_head_tl
3243 \__unravel_scan_int:
3244 }
3245 { \__unravel_tex_use:n { math_given } }
3246 { \__unravel_prev_input:V \l__unravel_head_tl }
3247 { \__unravel_tex_use:n { delim_num } }
3248 { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
3249 }
3250 {
3251 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3252 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3253 \tl_use:N \l__unravel_head_tl \scan_stop:
3254 }
3255 {
3256 \__unravel_back_input:
3257 \__unravel_scan_left_brace:
3258 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3259 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3260 \gtl_gconcat:NNN \g__unravel_output_gtl
3261 \g__unravel_output_gtl \c_group_begin_gtl
3262 \tl_use:N \l__unravel_head_tl \c_group_begin_token
3263 }
3264 \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3265 }
3266 \__unravel_new_tex_cmd:nn { endv } % 9

```



```

3267 {
3268   \__unravel_mode_non_math:n
3269   {
3270     \__unravel_not_implemented:n { alignments }
3271   }
3272 }

```

Blank spaces are ignored in vertical and math modes in the same way as `\relax` is in all modes. In horizontal mode, add them to the output.

```

3273 \__unravel_new_tex_cmd:nn { blank_char } % 10
3274 {
3275   \mode_if_horizontal:T
3276   {
3277     \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
3278     \l__unravel_head_token
3279   }
3280   \__unravel_print_action:
3281 }

```

Letters and other characters leave vertical mode.

```

3282 \__unravel_new_tex_cmd:nn { the_char } % 11
3283 {
3284   \__unravel_mode_non_vertical:n
3285   {
3286     \tl_set:Ne \l__unravel_tmpa_tl
3287     { ' \__unravel_token_to_char:N \l__unravel_head_token }
3288     \mode_if_math:TF
3289     { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
3290     { \__unravel_char:V \l__unravel_tmpa_tl }
3291   }
3292 }

```

```

3293 \__unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12

```

```

3294 \__unravel_new_tex_cmd:nn { par_end } % 13
3295 {
3296   \__unravel_mode_non_math:n
3297   {
3298     \mode_if_vertical:TF
3299     { \__unravel_par: }
3300     {
3301       % if align_state<0 then off_save;
3302       \__unravel_par_if_hmode:
3303       \mode_if_vertical:T
3304       { \mode_if_inner:F { \__unravel_build_page: } }
3305     }
3306   }
3307 }

```

```

3308 \__unravel_new_tex_cmd:nn { stop } % 14
3309 {
3310   \__unravel_mode_vertical:n
3311   {
3312     \mode_if_inner:TF
3313     { \__unravel_forbidden_case: }
3314     {
3315       % ^^A todo: unless its_all_over

```

```

3316         \int_gdecr:N \g__unravel_ends_int
3317         \int_compare:nNnTF \g__unravel_ends_int > 0
3318         {
3319             \__unravel_back_input:
3320             \__unravel_back_input:n
3321             {
3322                 \__unravel_hbox:w to \tex_hsize:D { }
3323                 \tex_vfill:D
3324                 \tex_penalty:D - '1000000000 ~
3325             }
3326             \__unravel_build_page:
3327             \__unravel_print_action:e { End-everything! }
3328         }
3329         {
3330             \__unravel_print_outcome:
3331             \l__unravel_head_token
3332         }
3333     }
3334 }
3335 }
3336 \__unravel_new_tex_cmd:nn { delim_num } % 15
3337 {
3338     \__unravel_mode_math:n
3339     {
3340         \__unravel_prev_input_gpush:N \l__unravel_head_tl
3341         \__unravel_print_action:
3342         \__unravel_scan_int:
3343         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3344         \tl_use:N \l__unravel_head_tl \scan_stop:
3345         \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3346     }
3347 }

```

### 2.12.2 Boxes: from 16 to 31

- char\_num=16 for \char
- math\_char\_num=17 for \mathchar
- mark=18 for \mark and \marks
- xray=19 for \show, \showbox, \showthe, \showlists, \showgroups, \showtokens, \showifs.
- make\_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
- hmove=21 for \moveright and \moveleft.
- vmove=22 for \lower and \raise.
- un\_hbox=23 for \unhbox and \unhcopy.
- unvbox=24 for \unvbox, \unvcopy, \pagediscards, and \splitdiscards.
- remove\_item=25 for \unpenalty (12), \unkern (11), \unskip (10).

- hskip=26 for \hfil, \hfill, \hss, \hfilneg, \hskip.
- vskip=27 for \vfil, \vfill, \vss, \vfilneg, \vskip.
- mskip=28 for \mskip (5).
- kern=29 for \kern (1).
- mkern=30 for \mkern (99).
- leader\_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).

\char leaves vertical mode, then scans an integer operand, then calls \\_\_unravel\_char\_in\_mmode:n or \\_\_unravel\_char:n depending on the mode. See implementation of the\_char and other\_char.

```

3348 \__unravel_new_tex_cmd:nn { char_num } % 16
3349 {
3350   \__unravel_mode_non_vertical:n
3351   {
3352     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3353     \__unravel_print_action:
3354     \__unravel_scan_int:
3355     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3356     \mode_if_math:TF
3357     { \__unravel_char_in_mmode:e { \tl_tail:N \l__unravel_head_tl } }
3358     { \__unravel_char:e { \tl_tail:N \l__unravel_head_tl } }
3359   }
3360 }

```

Only allowed in math mode, \mathchar reads an integer operand, and calls \\_\_unravel\_mathchar:n, which places the corresponding math character in the \g\_\_unravel\_output\_gtl, and in the actual output.

```

3361 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3362 {
3363   \__unravel_mode_math:n
3364   {
3365     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3366     \__unravel_print_action:
3367     \__unravel_scan_int:
3368     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3369     \__unravel_mathchar:e { \tl_tail:N \l__unravel_head_tl }
3370   }
3371 }

```

```

3372 \__unravel_new_tex_cmd:nn { mark } % 18
3373 {
3374   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3375   \__unravel_print_action:
3376   \int_compare:nNnF \l__unravel_head_char_int = 0
3377   { \__unravel_scan_int: }
3378   \__unravel_prev_input_gpush:
3379   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3380   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3381   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3382   \__unravel_print_action:e

```

```

3383     { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3384     \tl_put_right:Ne \l__unravel_head_tl
3385     { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3386     \tl_use:N \l__unravel_head_tl
3387   }

```

We now implement the primitives `\show`, `\showbox`, `\showthe`, `\showlists`, `\showgroups`, `\showtokens` and `\showifs`. Those with no operand are sent to `TEX` after printing the action. Those with operands print first, then scan their operands, then are sent to `TEX`. The case of `\show` is a bit special, as its operand is a single token, which cannot easily be put into the the previous-input sequence in general. Since no expansion can occur, simply grab the token and show it.

```

3388 \__unravel_new_tex_cmd:nn { xray } % 19
3389 {
3390   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3391   \__unravel_print_action:
3392   \int_case:nnF \l__unravel_head_char_int
3393     {
3394       { 0 }
3395       { % show
3396         \__unravel_get_next:
3397         \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3398         \token_if_eq_meaning:NNTF
3399         \l__unravel_head_token \__unravel_special_relax:
3400         {
3401           \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
3402           \exp_after:wN \exp_not:N \l__unravel_head_tl
3403         }
3404         { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
3405       }
3406       { 2 }
3407       { % showthe
3408         \__unravel_get_x_next:
3409         \__unravel_rescan_something_internal:n { 5 }
3410         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3411         \exp_args:Ne \use:n % better display than \use:e
3412         { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3413       }
3414     }
3415   { % no operand for showlists, showgroups, showifs
3416     \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
3417     { \__unravel_scan_int: }
3418     \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
3419     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3420     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3421     \tl_use:N \l__unravel_head_tl \scan_stop:
3422   }
3423 }

```

`make_box=20` for `\box`, `\copy`, `\lastbox`, `\vsplit`, `\vtop`, `\vbox`, and `\hbox` (106).

```

3424 \__unravel_new_tex_cmd:nn { make_box } % 20
3425 {
3426   \__unravel_prev_input_gpush:
3427   \__unravel_back_input:

```

```

3428     \__unravel_do_box:N \c_false_bool
3429   }

```

\\_\_unravel\_do\_move: Scan a dimension and a box, and perform the shift, printing the appropriate action.

```

3430 \cs_new_protected:Npn \__unravel_do_move:
3431 {
3432   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3433   \__unravel_print_action:
3434   \__unravel_scan_normal_dimen:
3435   \__unravel_do_box:N \c_false_bool
3436 }

```

(End of definition for \\_\_unravel\_do\_move:.)

hmove=21 for \moveright and \moveleft.

```

3437 \__unravel_new_tex_cmd:nn { hmove } % 21
3438 {
3439   \mode_if_vertical:TF
3440     { \__unravel_do_move: } { \__unravel_forbidden_case: }
3441 }

```

vmove=22 for \lower and \raise.

```

3442 \__unravel_new_tex_cmd:nn { vmove } % 22
3443 {
3444   \mode_if_vertical:TF
3445     { \__unravel_forbidden_case: } { \__unravel_do_move: }
3446 }

```

\\_\_unravel\_do\_unpackage:

```

3447 \cs_new_protected:Npn \__unravel_do_unpackage:
3448 {
3449   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3450   \__unravel_print_action:
3451   \__unravel_scan_int:
3452   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3453   \tl_use:N \l__unravel_head_tl \scan_stop:
3454   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3455 }

```

(End of definition for \\_\_unravel\_do\_unpackage:.)

un\_hbox=23 for \unhbox and \unhcopy.

```

3456 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3457 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

```

unvbox=24 for \unvbox, \unvcopy, \pagediscards, and \splitdiscards. The latter two take no operands, so we just let T<sub>E</sub>X do its thing, then we show the action.

```

3458 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3459 {
3460   \__unravel_mode_vertical:n
3461   {
3462     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3463       { \l__unravel_head_token \__unravel_print_action: }
3464       { \__unravel_do_unpackage: }
3465   }
3466 }

```

remove\_item=25 for \unpenalty (12), \unkern (11), \unskip (10). Those commands only act on T<sub>E</sub>X's box/glue data structures, which unravel does not (and cannot) care about.

```
3467 \_unravel_new_tex_cmd:nn { remove_item } % 25
3468 { \l_unravel_head_token \_unravel_print_action: }
```

\\_unravel\_do\_append\_glue: For \hfil, \hfill, \hss, \hfilneg and their vertical analogs, simply call the primitive then print the action. For \hskip, \vskip and \mskip, read a normal glue or a mu glue (\l\_unravel\_head\_char\_int is 4 or 5), then call the primitive with that operand, and print the whole thing as an action.

```
3469 \cs_new_protected:Npn \_unravel_do_append_glue:
3470 {
3471   \int_compare:nNnTF \l_unravel_head_char_int < { 4 }
3472   { \tl_use:N \l_unravel_head_tl \_unravel_print_action: }
3473   {
3474     \_unravel_prev_input_gpush:N \l_unravel_head_tl
3475     \_unravel_print_action:
3476     \exp_args:Nf \_unravel_scan_glue:n
3477     { \int_eval:n { \l_unravel_head_char_int - 2 } }
3478     \_unravel_prev_input_gpop:N \l_unravel_head_tl
3479     \tl_use:N \l_unravel_head_tl \scan_stop:
3480     \_unravel_print_action:e { \tl_to_str:N \l_unravel_head_tl }
3481   }
3482 }
```

(End of definition for \\_unravel\_do\_append\_glue:.)

hskip=26 for \hfil, \hfill, \hss, \hfilneg, \hskip.

```
3483 \_unravel_new_tex_cmd:nn { hskip } % 26
3484 { \_unravel_mode_non_vertical:n { \_unravel_do_append_glue: } }
```

vskip=27 for \vfil, \vfill, \vss, \vfilneg, \vskip.

```
3485 \_unravel_new_tex_cmd:nn { vskip } % 27
3486 { \_unravel_mode_vertical:n { \_unravel_do_append_glue: } }
```

mskip=28 for \mskip (5).

```
3487 \_unravel_new_tex_cmd:nn { mskip } % 28
3488 { \_unravel_mode_math:n { \_unravel_do_append_glue: } }
```

\\_unravel\_do\_append\_kern: See \\_unravel\_do\_append\_glue:. This function is used for the primitives \kern and \mkern only.

```
3489 \cs_new_protected:Npn \_unravel_do_append_kern:
3490 {
3491   \_unravel_prev_input_gpush:N \l_unravel_head_tl
3492   \_unravel_print_action:
3493   \token_if_eq_meaning:NNTF \l_unravel_head_token \tex_kern:D
3494   { \_unravel_scan_dimen:nN { 2 } \c_false_bool }
3495   { \_unravel_scan_dimen:nN { 3 } \c_false_bool }
3496   \_unravel_prev_input_gpop:N \l_unravel_head_tl
3497   \tl_use:N \l_unravel_head_tl \scan_stop:
3498   \_unravel_print_action:e { \tl_to_str:N \l_unravel_head_tl }
3499 }
```

(End of definition for `\_unravel\_do\_append\_kern:`)

```
kern=29 for \kern (1).
3500 \_unravel\_new\_tex\_cmd:nn { kern } % 29
3501 { \_unravel\_do\_append\_kern: }
mkern=30 for \mkern (99).
3502 \_unravel\_new\_tex\_cmd:nn { mkern } % 30
3503 { \_unravel\_mode\_math:n { \_unravel\_do\_append\_kern: } }
leader\_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).
3504 \_unravel\_new\_tex\_cmd:nn { leader\_ship } % 31
3505 {
3506 \_unravel\_prev\_input\_gpush:N \l\_unravel\_head\_tl
3507 \_unravel\_print\_action:
3508 \tl\_if\_head\_eq\_meaning:VNTF \l\_unravel\_head\_tl \tex\_shipout:D
3509 { \_unravel\_do\_box:N \c\_false\_bool }
3510 { \_unravel\_do\_box:N \c\_true\_bool }
3511 }
```

### 2.12.3 From 32 to 47

- `halign=32`
- `valign=33`
- `no\_align=34`
- `vrule=35`
- `hrule=36`
- `insert=37`
- `vadjust=38`
- `ignore\_spaces=39`
- `after\_assignment=40`
- `after\_group=41`
- `break\_penalty=42`
- `start\_par=43`
- `ital\_corr=44`
- `accent=45`
- `math\_accent=46`
- `discretionary=47`

```
3512 \_unravel\_new\_tex\_cmd:nn { halign } % 32
3513 { \_unravel\_not\_implemented:n { halign } }
3514 \_unravel\_new\_tex\_cmd:nn { valign } % 33
3515 { \_unravel\_not\_implemented:n { valign } }
3516 \_unravel\_new\_tex\_cmd:nn { no\_align } % 34
3517 { \l\_unravel\_head\_token \_unravel\_print\_action: }
```

```

3518 \__unravel_new_tex_cmd:nn { vrule } % 35
3519 { \__unravel_mode_non_vertical:n { \__unravel_do_rule: } }
3520 \__unravel_new_tex_cmd:nn { hrule } % 36
3521 { \__unravel_mode_vertical:n { \__unravel_do_rule: } }
3522 \cs_new_protected:Npn \__unravel_do_rule:
3523 {
3524   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3525   \__unravel_print_action:
3526   \__unravel_scan_alt_rule:
3527   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3528   \tl_use:N \l__unravel_head_tl \scan_stop:
3529   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3530 }
3531 \__unravel_new_tex_cmd:nn { insert } % 37
3532 {
3533   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3534   \__unravel_print_action:
3535   \__unravel_scan_int:
3536   \__unravel_begin_insert_or_adjust:
3537 }
3538 \__unravel_new_tex_cmd:nn { vadjust } % 38
3539 {
3540   \mode_if_vertical:TF
3541   { \__unravel_forbidden_case: }
3542   {
3543     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3544     \__unravel_print_action:
3545     \__unravel_scan_keyword:nTF { pPrReE }
3546     \__unravel_begin_insert_or_adjust:
3547   }
3548 }
3549 \cs_new_protected:Npn \__unravel_begin_insert_or_adjust:
3550 {
3551   \__unravel_scan_left_brace:
3552   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3553   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3554   \gtl_gconcat:NNN \g__unravel_output_gtl
3555   \g__unravel_output_gtl \c_group_begin_gtl
3556   \tl_use:N \l__unravel_head_tl \c_group_begin_token
3557   \__unravel_print_action:e
3558   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
3559 }
3560 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
3561 {
3562   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3563   {
3564     \__unravel_print_action:
3565     \__unravel_get_x_non_blank:
3566     \__unravel_set_cmd:
3567     \__unravel_do_step:
3568   }
3569   { \__unravel_not_implemented:n { pdfprimitive } }
3570 }

```



```

3571 \__unravel_new_tex_cmd:nn { after_assignment } % 40
3572 {
3573   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3574   \__unravel_get_next:
3575   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3576   \__unravel_print_action:e
3577   {
3578     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3579     \gtl_to_str:N \l__unravel_head_gtl
3580   }
3581 }

```

Save the next token at the end of `\l__unravel_after_group_gtl`, unless we are at the bottom group level, in which case, the token is ignored completely.

```

3582 \__unravel_new_tex_cmd:nn { after_group } % 41
3583 {
3584   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3585   \__unravel_get_next:
3586   \int_compare:nNnTF \__unravel_currentgrouptype: = 0
3587   {
3588     \__unravel_print_action:e
3589     {
3590       Aftergroup~(level~0~=>~dropped):~
3591       \tl_to_str:N \l__unravel_tmpa_tl
3592       \gtl_to_str:N \l__unravel_head_gtl
3593     }
3594   }
3595   {
3596     \gtl_concat:NNN \l__unravel_after_group_gtl
3597     \l__unravel_after_group_gtl \l__unravel_head_gtl
3598     \__unravel_print_action:e
3599     {
3600       Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3601       \gtl_to_str:N \l__unravel_head_gtl
3602     }
3603   }
3604 }

```

See `\__unravel_do_append_glue:`.

```

3605 \__unravel_new_tex_cmd:nn { break_penalty } % 42
3606 {
3607   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3608   \__unravel_print_action:
3609   \__unravel_scan_int:
3610   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3611   \tl_use:N \l__unravel_head_tl \scan_stop:
3612   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3613 }

```

```

3614 \__unravel_new_tex_cmd:nn { start_par } % 43
3615 {
3616   \mode_if_vertical:TF
3617   {
3618     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3619     { \__unravel_new_graf:N \c_false_bool }

```

```

3620         { \_unravel_new_graf:N \c_true_bool }
3621     }
3622     {
3623     \int_compare:nNnT \l_unravel_head_char_int = { 1 } % indent
3624     {
3625         \_unravel_hbox:w width \tex_parindent:D { }
3626         \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3627     }
3628     \_unravel_print_action:
3629     }
3630 }
3631 \_unravel_new_tex_cmd:nn { ital_corr } % 44
3632 {
3633     \mode_if_vertical:TF { \_unravel_forbidden_case: }
3634     { \l_unravel_head_token \_unravel_print_action: }
3635 }

```

\\_unravel\_do\_accent:

```

3636 \cs_new_protected:Npn \_unravel_do_accent:
3637 {
3638     \_unravel_prev_input_gpush:N \l_unravel_head_tl
3639     \_unravel_print_action:
3640     \_unravel_scan_int:
3641     \_unravel_do_assignments:
3642     \bool_if:nTF
3643     {
3644         \token_if_eq_catcode_p:NN
3645         \l_unravel_head_token \c_catcode_letter_token
3646         ||
3647         \token_if_eq_catcode_p:NN
3648         \l_unravel_head_token \c_catcode_other_token
3649         ||
3650         \int_compare_p:nNn
3651         \l_unravel_head_cmd_int = { \_unravel_tex_use:n { char_given } }
3652     }
3653     { \_unravel_prev_input:V \l_unravel_head_tl }
3654     {
3655         \token_if_eq_meaning:NNTF \l_unravel_head_token \tex_char:D
3656         {
3657             \_unravel_prev_input:V \l_unravel_head_tl
3658             \_unravel_scan_int:
3659         }
3660         { \_unravel_break:w }
3661     }
3662     \_unravel_prev_input_gpop:N \l_unravel_head_tl
3663     \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3664     \tl_use:N \l_unravel_head_tl \scan_stop:
3665     \_unravel_print_action:e { \tl_to_str:N \l_unravel_head_tl }
3666     \_unravel_break_point:
3667 }

```

*(End of definition for \\_unravel\_do\_accent:.)*

`\__unravel_do_math_accent:` T<sub>E</sub>X will complain if `\l__unravel_head_tl` happens to start with `\accent` (the user used `\accent` in math mode).

```

3668 \cs_new_protected:Npn \__unravel_do_math_accent:
3669 {
3670   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3671   \__unravel_print_action:
3672   \__unravel_scan_int:
3673   \__unravel_do_one_atom:
3674 }

(End of definition for \__unravel_do_math_accent:.)

3675 \__unravel_new_tex_cmd:nn { accent } % 45
3676 {
3677   \__unravel_mode_non_vertical:n
3678   {
3679     \mode_if_math:TF
3680     { \__unravel_do_math_accent: } { \__unravel_do_accent: }
3681   }
3682 }

3683 \__unravel_new_tex_cmd:nn { math_accent } % 46
3684 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }

3685 \__unravel_new_tex_cmd:nn { discretionary } % 47
3686 {
3687   \__unravel_mode_non_vertical:n
3688   {
3689     \int_compare:nNnTF \l__unravel_head_char_int = { 1 }
3690     { \__unravel_output_head_token: }
3691     { \__unravel_do_choice: }
3692   }
3693 }

```

#### 2.12.4 Maths: from 48 to 56

- eq\_no=48
- left\_right=49
- math\_comp=50
- limit\_switch=51
- above=52
- math\_style=53
- math\_choice=54
- non\_script=55
- vcenter=56

```

3694 \__unravel_new_tex_cmd:nn { eq_no } % 48
3695 {
3696   \mode_if_math:TF
3697   {
3698     \mode_if_inner:TF
3699     { \__unravel_off_save: }
3700     {
3701       \int_compare:nNnTF \tex_currentgrouptype:D = { 15 }
3702       {
3703         \__unravel_box_hook:N \tex_everymath:D
3704         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3705         \l__unravel_head_token
3706         \__unravel_box_hook_end:
3707         \int_set:Nn \l__unravel_choice_int { 1 }
3708       }
3709       { \__unravel_off_save: }
3710     }
3711   }
3712   { \__unravel_forbidden_case: }
3713 }
3714 \__unravel_new_tex_cmd:nn { left_right } % 49
3715 {
3716   \__unravel_mode_math:n
3717   {
3718     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3719     \__unravel_print_action:
3720     \__unravel_scan_delimiter:
3721     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3722     \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_left:D
3723     {
3724       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3725       \tl_use:N \l__unravel_head_tl \scan_stop:
3726       \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3727     }
3728     {
3729       \int_case:nnF \tex_currentgrouptype:D
3730       {
3731         { 16 }
3732         {
3733           \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3734           \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3735           \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_middle:D
3736           {
3737             \tl_use:N \l__unravel_head_tl \scan_stop:
3738             \gtl_clear:N \l__unravel_after_group_gtl
3739           }
3740           { \tl_use:N \l__unravel_head_tl \scan_stop: }
3741           \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3742         }
3743         { 15 }
3744         { % todo: this is a TeX error
3745           \tl_use:N \l__unravel_head_tl \scan_stop:
3746         }
3747       }
3748     }
3749   }

```

```

3748         { \_unravel_off_save: }
3749     }
3750 }
3751 }
3752 \cs_new_protected:Npn \_unravel_scan_delimiter:
3753 {
3754     \_unravel_get_x_non_relax:
3755     \_unravel_set_cmd:
3756     \int_case:nnF \l_unravel_head_cmd_int
3757     {
3758         { \_unravel_tex_use:n { the_char } }
3759         { \_unravel_prev_input:V \l_unravel_head_tl }
3760         { \_unravel_tex_use:n { other_char } }
3761         { \_unravel_prev_input:V \l_unravel_head_tl }
3762         { \_unravel_tex_use:n { delim_num } }
3763         {
3764             \_unravel_prev_input:V \l_unravel_head_tl
3765             \_unravel_scan_int:
3766         }
3767     }
3768     {
3769         \_unravel_back_input:
3770         \_unravel_tex_error:nV { missing-delim } \l_unravel_head_tl
3771         \_unravel_prev_input:n { . }
3772     }
3773 }
3774 \_unravel_new_tex_cmd:nn { math_comp } % 50
3775 { \_unravel_mode_math:n { \_unravel_sub_sup: } }
3776 \_unravel_new_tex_cmd:nn { limit_switch } % 51
3777 { \_unravel_mode_math:n { \_unravel_output_head_token: } }
3778 \cs_new_protected:Npn \_unravel_output_head_token:
3779 {
3780     \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3781     \l_unravel_head_token
3782     \_unravel_print_action:
3783 }
3784 \_unravel_new_tex_cmd:nn { above } % 52
3785 { \_unravel_mode_math:n { \_unravel_not_implemented:n { above } } }
3786 \_unravel_new_tex_cmd:nn { math_style } % 53
3787 { \_unravel_mode_math:n { \_unravel_output_head_token: } }
3788 \_unravel_new_tex_cmd:nn { math_choice } % 54
3789 { \_unravel_mode_math:n { \_unravel_do_choice: } }
3790 \cs_new_protected:Npn \_unravel_do_choice:
3791 {
3792     \_unravel_prev_input_gpush:N \l_unravel_head_tl
3793     \_unravel_print_action:
3794     \_unravel_scan_left_brace:
3795     \_unravel_prev_input_gpop:N \l_unravel_head_tl
3796     \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3797     \gtl_gconcat:NNN \g_unravel_output_gtl
3798     \g_unravel_output_gtl \c_group_begin_gtl
3799     \tl_use:N \l_unravel_head_tl \c_group_begin_token

```

```

3800 \gtl_clear:N \l__unravel_after_group_gtl
3801 \int_set:Nn \l__unravel_choice_int { 1 }
3802 \__unravel_print_action:e
3803 { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
3804 }
3805 \__unravel_new_tex_cmd:nn { non_script } % 55
3806 { \__unravel_mode_math:n { \__unravel_output_head_token: } }
3807 \__unravel_new_tex_cmd:nn { vcenter } % 56
3808 { \__unravel_mode_math:n { \__unravel_not_implemented:n { vcenter } } }

```

### 2.12.5 From 57 to 70

- case\_shift=57
- message=58
- extension=59
- in\_stream=60
- begin\_group=61
- end\_group=62
- omit=63
- ex\_space=64
- no\_boundary=65
- radical=66
- end\_cs\_name=67
- char\_given=68
- math\_given=69
- last\_item=70

```

3809 \__unravel_new_tex_cmd:nn { case_shift } % 57
3810 {
3811 \__unravel_prev_input_gpush:N \l__unravel_head_tl
3812 \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3813 \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3814 \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
3815 }
3816 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
3817 {
3818 #1 { \__unravel_back_input:n {#2} }
3819 \__unravel_print_action:e
3820 { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3821 }

```

```

3822 \__unravel_new_tex_cmd:nn { message } % 58
3823 {
3824   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3825   \__unravel_print_action:
3826   \__unravel_scan_toks_to_str:
3827   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3828   \tl_use:N \l__unravel_head_tl
3829   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3830 }

Extensions are implemented in a later section.

3831 \__unravel_new_tex_cmd:nn { extension } % 59
3832 {
3833   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3834   \__unravel_print_action:
3835   \__unravel_scan_extension_operands:
3836   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3837   \tl_use:N \l__unravel_head_tl \scan_stop:
3838   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3839 }

3840 \__unravel_new_tex_cmd:nn { in_stream } % 60
3841 {
3842   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3843   \__unravel_print_action:
3844   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3845   {
3846     \__unravel_scan_int:
3847     \__unravel_scan_optional_equals:
3848     \__unravel_scan_file_name:
3849   }
3850   { \__unravel_scan_int: }
3851   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3852   \tl_use:N \l__unravel_head_tl \scan_stop:
3853   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
3854 }

3855 \__unravel_new_tex_cmd:nn { begin_group } % 61
3856 {
3857   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3858   \l__unravel_head_token
3859   \gtl_clear:N \l__unravel_after_group_gtl
3860   \__unravel_print_action:
3861 }

3862 \__unravel_new_tex_cmd:nn { end_group } % 62
3863 {
3864   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3865   \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3866   \l__unravel_head_token
3867   \__unravel_print_action:
3868 }

3869 \__unravel_new_tex_cmd:nn { omit } % 63
3870 { \l__unravel_head_token \__unravel_print_action: }

3871 \__unravel_new_tex_cmd:nn { ex_space } % 64
3872 {

```

```

3873   \__unravel_mode_non_vertical:n
3874   { \l__unravel_head_token \__unravel_print_action: }
3875   }
3876   \__unravel_new_tex_cmd:nn { no_boundary } % 65
3877   {
3878     \__unravel_mode_non_vertical:n
3879     { \l__unravel_head_token \__unravel_print_action: }
3880   }
3881   \__unravel_new_tex_cmd:nn { radical } % 66
3882   { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
3883   \__unravel_new_tex_cmd:nn { end_cs_name } % 67
3884   {
3885     \__unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
3886     \__unravel_print_action:
3887   }
3888   See the_char and other_char.
3889   \__unravel_new_tex_cmd:nn { char_given } % 68
3890   {
3891     \__unravel_mode_non_vertical:n
3892     {
3893       \mode_if_math:TF
3894       { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
3895       { \__unravel_char:V \l__unravel_head_char_int }
3896     }
3897   }
3898   See math_char_num.
3899   \__unravel_new_tex_cmd:nn { math_given } % 69
3900   {
3901     \__unravel_mode_math:n
3902     { \__unravel_mathchar:e { \int_use:N \l__unravel_head_char_int } }
3903   }
3904   \__unravel_new_tex_cmd:nn { last_item } % 70
3905   { \__unravel_forbidden_case: }

```

### 2.12.6 Extensions

\\_\_unravel\_scan\_extension\_operands:

```

3904   \cs_new_protected:Npn \__unravel_scan_extension_operands:
3905   {
3906     \int_case:nnF \l__unravel_head_char_int
3907     {
3908       { 0 } % openout
3909       {
3910         \__unravel_scan_int:
3911         \__unravel_scan_optional_equals:
3912         \__unravel_scan_file_name:
3913       }
3914       { 1 } % write
3915       {
3916         \__unravel_scan_int:
3917         \__unravel_scan_toks:NN \c_false_bool \c_false_bool

```



```

3918     }
3919 { 2 } % closeout
3920     { \_unravel_scan_int: }
3921 { 3 } % special
3922     { \_unravel_scan_toks_to_str: }
3923 { 4 } % immediate
3924     { \_unravel_scan_immediate_operands: }
3925 { 5 } % setlanguage
3926     {
3927         \mode_if_horizontal:TF
3928         { \_unravel_scan_int: }
3929         { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3930     }
3931 { 6 } % pdfliteral
3932     {
3933         \_unravel_scan_keyword:nF { dDiIrReEcCtT }
3934         { \_unravel_scan_keyword:n { pPaAgGeE } }
3935         \_unravel_scan_pdf_ext_toks:
3936     }
3937 { 7 } % pdfobj
3938     {
3939         \_unravel_scan_keyword:nTF
3940         { rReEsSeErRvVeEoObBjJnNuUmM }
3941         { \_unravel_skip_optional_space: }
3942         {
3943             \_unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3944             { \_unravel_scan_int: }
3945             \_unravel_scan_keyword:nT { sStTrReEaAmM }
3946             {
3947                 \_unravel_scan_keyword:nT { aAtTtTrR }
3948                 { \_unravel_scan_pdf_ext_toks: }
3949             }
3950             \_unravel_scan_keyword:n { fFiIlLeE }
3951             \_unravel_scan_pdf_ext_toks:
3952         }
3953     }
3954 { 8 } % pdfrefobj
3955     { \_unravel_scan_int: }
3956 { 9 } % pdfxform
3957     {
3958         \_unravel_scan_keyword:nT { aAtTtTrR }
3959         { \_unravel_scan_pdf_ext_toks: }
3960         \_unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3961         { \_unravel_scan_pdf_ext_toks: }
3962         \_unravel_scan_int:
3963     }
3964 { 10 } % pdfrefxform
3965     { \_unravel_scan_int: }
3966 { 11 } % pdfximage
3967     { \_unravel_scan_image: }
3968 { 12 } % pdfrefximage
3969     { \_unravel_scan_int: }
3970 { 13 } % pdfannot
3971     {

```

```

3972     \_unravel_scan_keyword:nTF
3973     { rReEsSeErRvVeEoObBjJnNuUmM }
3974     { \_unravel_scan_optional_space: }
3975     {
3976         \_unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3977         { \_unravel_scan_int: }
3978         \_unravel_scan_alt_rule:
3979         \_unravel_scan_pdf_ext_toks:
3980     }
3981 }
3982 { 14 } % pdfstartlink
3983 {
3984     \mode_if_vertical:TF
3985     { \_unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3986     {
3987         \_unravel_scan_rule_attr:
3988         \_unravel_scan_action:
3989     }
3990 }
3991 { 15 } % pdfendlink
3992 {
3993     \mode_if_vertical:T
3994     { \_unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3995 }
3996 { 16 } % pdfoutline
3997 {
3998     \_unravel_scan_keyword:nT { aAtTtTrR }
3999     { \_unravel_scan_pdf_ext_toks: }
4000     \_unravel_scan_action:
4001     \_unravel_scan_keyword:nT { cCoOuUnNtT }
4002     { \_unravel_scan_int: }
4003     \_unravel_scan_pdf_ext_toks:
4004 }
4005 { 17 } % pdfdest
4006 { \_unravel_scan_pdfdest_operands: }
4007 { 18 } % pdfthread
4008 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
4009 { 19 } % pdfstartthread
4010 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
4011 { 20 } % pdfendthread
4012 { }
4013 { 21 } % pdfsavepos
4014 { }
4015 { 22 } % pdfinfo
4016 { \_unravel_scan_pdf_ext_toks: }
4017 { 23 } % pdfcatalog
4018 {
4019     \_unravel_scan_pdf_ext_toks:
4020     \_unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
4021     { \_unravel_scan_action: }
4022 }
4023 { 24 } % pdfnames
4024 { \_unravel_scan_pdf_ext_toks: }
4025 { 25 } % pdffontattr

```

```

4026     {
4027         \_unravel_scan_font_ident:
4028         \_unravel_scan_pdf_ext_toks:
4029     }
4030 { 26 } % pdfincludechars
4031     {
4032         \_unravel_scan_font_ident:
4033         \_unravel_scan_pdf_ext_toks:
4034     }
4035 { 27 } % pdfmapfile
4036     { \_unravel_scan_pdf_ext_toks: }
4037 { 28 } % pdfmapline
4038     { \_unravel_scan_pdf_ext_toks: }
4039 { 29 } % pdftrailer
4040     { \_unravel_scan_pdf_ext_toks: }
4041 { 30 } % pdfresettimer
4042     { }
4043 { 31 } % pdffontexpand
4044     {
4045         \_unravel_scan_font_ident:
4046         \_unravel_scan_optional_equals:
4047         \_unravel_scan_int:
4048         \_unravel_scan_int:
4049         \_unravel_scan_int:
4050         \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
4051         { \_unravel_skip_optional_space: }
4052     }
4053 { 32 } % pdfsetrandomseed
4054     { \_unravel_scan_int: }
4055 { 33 } % pdfsnaprefpoint
4056     { }
4057 { 34 } % pdfsnapy
4058     { \_unravel_scan_normal_glue: }
4059 { 35 } % pdfsnapycomp
4060     { \_unravel_scan_int: }
4061 { 36 } % pdfglyphfontunicode
4062     {
4063         \_unravel_scan_pdf_ext_toks:
4064         \_unravel_scan_pdf_ext_toks:
4065     }
4066 { 37 } % pdfcolorstack
4067     { \_unravel_scan_pdfcolorstack_operands: }
4068 { 38 } % pdfsetmatrix
4069     { \_unravel_scan_pdf_ext_toks: }
4070 { 39 } % pdfsave
4071     { }
4072 { 40 } % pdfrestore
4073     { }
4074 { 41 } % pdfnubuiltintfontunicode
4075     { \_unravel_scan_font_ident: }
4076 }
4077 { } % no other cases.
4078 }

```

(End of definition for \\_unravel\_scan\_extension\_operands:.)

\\_unravel\_scan\_pdfcolorstack\_operands:

```
4079 \cs_new_protected:Npn \_unravel_scan_pdfcolorstack_operands:
4080 {
4081   \_unravel_scan_int:
4082   \_unravel_scan_keyword:nF { sSeEtT }
4083   {
4084     \_unravel_scan_keyword:nF { pPuUsShH }
4085     {
4086       \_unravel_scan_keyword:nF { pPoOpP }
4087       {
4088         \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
4089         {
4090           \_unravel_error:nnnnn { color-stack-action-missing }
4091           { } { } { } { }
4092         }
4093       }
4094     }
4095   }
4096 }
```

*(End of definition for \\_unravel\_scan\_pdfcolorstack\_operands:.)*

\\_unravel\_scan\_rule\_attr:

```
4097 \cs_new_protected:Npn \_unravel_scan_rule_attr:
4098 {
4099   \_unravel_scan_alt_rule:
4100   \_unravel_scan_keyword:nT { aAtTtTrR }
4101   { \_unravel_scan_pdf_ext_toks: }
4102 }
```

*(End of definition for \\_unravel\_scan\_rule\_attr:.)*

\\_unravel\_scan\_action:

```
4103 \cs_new_protected:Npn \_unravel_scan_action:
4104 {
4105   \_unravel_scan_keyword:nTF { uUsSeErR }
4106   { \_unravel_scan_pdf_ext_toks: }
4107   {
4108     \_unravel_scan_keyword:nF { gGoOtToO }
4109     {
4110       \_unravel_scan_keyword:nF { tThHrReEaAdD }
4111       { \_unravel_error:nnnnn { action-type-missing } { } { } { } { } }
4112     }
4113   }
4114   \_unravel_scan_keyword:nT { fFiIlLeE }
4115   { \_unravel_scan_pdf_ext_toks: }
4116   \_unravel_scan_keyword:nTF { pPaAgGeE }
4117   {
4118     \_unravel_scan_int:
4119     \_unravel_scan_pdf_ext_toks:
4120   }
4121   {
4122     \_unravel_scan_keyword:nTF { nNaAmMeE }
4123     { \_unravel_scan_pdf_ext_toks: }
```

```

4124     {
4125         \_unravel_scan_keyword:nTF { nNuUmM }
4126         { \_unravel_scan_int: }
4127         { \_unravel_error:nmmnn { identifier-type-missing } { } { } { } { } }
4128     }
4129 }
4130 \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
4131 { \_unravel_skip_optional_space: }
4132 {
4133     \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
4134     { \_unravel_skip_optional_space: }
4135 }
4136 }

```

(End of definition for \\_unravel\_scan\_action:.)

\\_unravel\_scan\_image: Used by \pdfximage.

```

4137 \cs_new_protected:Npn \_unravel_scan_image:
4138 {
4139     \_unravel_scan_rule_attr:
4140     \_unravel_scan_keyword:nTF { nNaAmMeEdD }
4141     { \_unravel_scan_pdf_ext_toks: }
4142     {
4143         \_unravel_scan_keyword:nT { pPaAgGeE }
4144         { \_unravel_scan_int: }
4145     }
4146     \_unravel_scan_keyword:nT { cCoO1LoOrRsSpPaAcCeE }
4147     { \_unravel_scan_int: }
4148     \_unravel_scan_pdf_ext_toks:
4149 }

```

(End of definition for \\_unravel\_scan\_image:.)

\\_unravel\_scan\_immediate\_operands:

```

4150 \cs_new_protected:Npn \_unravel_scan_immediate_operands:
4151 {
4152     \_unravel_get_x_next:
4153     \_unravel_set_cmd:
4154     \int_compare:nNnTF
4155     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { extension } }
4156     {
4157         \int_compare:nNnTF
4158         \l__unravel_head_char_int < { 3 } % openout, write, closeout
4159         { \_unravel_scan_immediate_operands_aux: }
4160         {
4161             \int_case:nnF \l__unravel_head_char_int
4162             {
4163                 { 7 } { \_unravel_scan_extension_operands_aux: } % pdfobj
4164                 { 9 }
4165                 {
4166                     \_unravel_prepare_mag:
4167                     \_unravel_scan_extension_operands_aux:
4168                 } % pdfxform
4169                 { 11 } { \_unravel_scan_extension_operands_aux: } %pdfximage
4170             }
4171         }
4172     }

```

```

4171         { \_unravel_scan_immediate_operands_bad: }
4172     }
4173 }
4174 { \_unravel_scan_immediate_operands_bad: }
4175 }
4176 \cs_new_protected:Npn \_unravel_scan_immediate_operands_aux:
4177 {
4178     \_unravel_prev_input:V \l_unravel_head_tl
4179     \_unravel_scan_extension_operands:
4180 }
4181 \cs_new_protected:Npn \_unravel_scan_immediate_operands_bad:
4182 {
4183     \_unravel_back_input:
4184     \_unravel_prev_input_gpop:N \l_unravel_head_tl
4185     \_unravel_print_action:e { \tl_to_str:N \l_unravel_head_tl ignored }
4186     \_unravel_prev_input_gpush:
4187 }
4188

```

(End of definition for \\_unravel\_scan\_immediate\_operands:.)

\\_unravel\_scan\_pdfdest\_operands:

```

4189 \cs_new_protected:Npn \_unravel_scan_pdfdest_operands:
4190 {
4191     \_unravel_scan_keyword:nTF { nNuUmM }
4192     { \_unravel_scan_int: }
4193     {
4194         \_unravel_scan_keyword:nTF { nNaAmMeE }
4195         { \_unravel_scan_pdf_ext_toks: }
4196         { \_unravel_error:nmnnn { identifier-type-missing } { } { } { } { } }
4197     }
4198     \_unravel_scan_keyword:nTF { xXyYzZ }
4199     {
4200         \_unravel_scan_keyword:nT { zZoOoOmM }
4201         { \_unravel_scan_int: }
4202     }
4203     {
4204         \_unravel_scan_keyword:nF { fFiItTbBhH }
4205         {
4206             \_unravel_scan_keyword:nF { fFiItTbBvV }
4207             {
4208                 \_unravel_scan_keyword:nF { fFiItTbB }
4209                 {
4210                     \_unravel_scan_keyword:nF { fFiItThHhH }
4211                     {
4212                         \_unravel_scan_keyword:nF { fFiItTvV }
4213                         {
4214                             \_unravel_scan_keyword:nTF
4215                             { fFiItTrR }
4216                             {
4217                                 \_unravel_skip_optional_space:
4218                                 \_unravel_scan_alt_rule:
4219                                 \use_none:n
4220                             }

```

```

4221         {
4222             \_unravel_scan_keyword:nF
4223             { fFiItT }
4224             {
4225                 \_unravel_error:nnnn { destination-type-missing }
4226                 { } { } { } { }
4227             }
4228         }
4229     }
4230 }
4231 }
4232 }
4233 }
4234 }
4235 \_unravel_skip_optional_space:
4236 }

```

(End of definition for \\_unravel\_scan\_pdfdest\_operands:.)

### 2.12.7 Assignments

Quoting `tex.web`: “Every prefix, and every command code that might or might not be prefixed, calls the action procedure `prefixed_command`. This routine accumulates a sequence of prefixes until coming to a non-prefix, then it carries out the command.” We define all those commands in one go, from `max_non_prefixed_command+1=71` to `max_command=102`.

```

4237 \cs_set_protected:Npn \_unravel_tmp:w
4238 {
4239     \_unravel_prev_input_gpush:
4240     \_unravel_prefixed_command:
4241 }
4242 \int_step_inline:nnnn
4243 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
4244 { 1 }
4245 { \_unravel_tex_use:n { max_command } }
4246 { \cs_new_eq:cN { \_unravel_cmd_#1: } \_unravel_tmp:w }

```

`\_unravel_prefixed_command:` Accumulated prefix codes so far are stored as the last item of the previous-input sequence.

```

4247 \cs_new_protected:Npn \_unravel_prefixed_command:
4248 {
4249     \int_while_do:nNnn
4250     \l_unravel_head_cmd_int = { \_unravel_tex_use:n { prefix } }
4251     {
4252         \_unravel_prev_input:V \l_unravel_head_tl
4253         \_unravel_get_x_non_relax:
4254         \_unravel_set_cmd:
4255         \int_compare:nNnF \l_unravel_head_cmd_int
4256         > { \_unravel_tex_use:n { max_non_prefixed_command } }
4257         {
4258             \_unravel_prev_input_gpop:N \l_unravel_tpa_tl
4259             \_unravel_error:neeee { erroneous-prefixes }
4260             { \tl_to_str:N \l_unravel_tpa_tl }
4261             { \tl_to_str:N \l_unravel_head_tl }
4262             { } { }

```

```

4263         \__unravel_back_input:
4264         \__unravel_omit_after_assignment:w
4265     }
4266 }
4267 % ^^A todo: Discard non-\global prefixes if they are irrelevant
4268 % ^^A todo: Adjust for the setting of \globaldefs
4269 \cs_if_exist_use:cF
4270 { \__unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
4271 {
4272     \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
4273     \__unravel_omit_after_assignment:w
4274 }
4275 \__unravel_after_assignment:
4276 }

```

*(End of definition for \\_\_unravel\_prefixed\_command:.)*

We now need to implement prefixed commands, for command codes in the range [71,102], with the exception of prefix=93, which would have been collected by the \\_\_unravel\_prefixed\_command: loop.

```

\__unravel_after_assignment:
  \__unravel_omit_after_assignment:w
4277 \cs_new_protected:Npn \__unravel_after_assignment:
4278 {
4279     \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
4280     \gtl_gclear:N \g__unravel_after_assignment_gtl
4281 }
4282 \cs_new_protected:Npn \__unravel_omit_after_assignment:w
4283     #1 \__unravel_after_assignment: { }

```

*(End of definition for \\_\_unravel\_after\_assignment: and \\_\_unravel\_omit\_after\_assignment:w.)*

```

\__unravel_prefixed_new:nn
4284 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
4285 {
4286     \cs_new_protected:cpn
4287     { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
4288 }

```

*(End of definition for \\_\_unravel\_prefixed\_new:nn.)*

```

\__unravel_assign_token:n
4289 \cs_new_protected:Npn \__unravel_assign_token:n #1
4290 {
4291     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4292     #1
4293     \tl_use:N \l__unravel_head_tl \scan_stop:
4294     \__unravel_print_assigned_token:
4295 }

```

*(End of definition for \\_\_unravel\_assign\_token:n.)*

```

\__unravel_assign_register:
4296 \cs_new_protected:Npn \__unravel_assign_register:
4297 {
4298     \__unravel_prev_input_gpop:N \l__unravel_head_tl

```



```

4299 \tl_use:N \l__unravel_head_tl \scan_stop:
4300 \__unravel_print_assigned_register:
4301 }

```

(End of definition for \\_\_unravel\_assign\_register:.)

\\_\_unravel\_assign\_value:nn

```

4302 \cs_new_protected:Npn \__unravel_assign_value:nn #1#2
4303 {
4304   \tl_if_empty:nF {#1}
4305   {
4306     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4307     \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
4308     #1
4309     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4310   }
4311   \__unravel_prev_input:V \l__unravel_head_tl
4312   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4313   \__unravel_scan_optional_equals:
4314   #2
4315   \__unravel_assign_register:
4316 }

```

(End of definition for \\_\_unravel\_assign\_value:nn.)

\\_\_unravel\_assign\_toks:

```

4317 \__unravel_prefixed_new:nn { toks_register } % 71
4318 {
4319   \int_compare:nNnT \l__unravel_head_char_int = 0
4320   { % \toks
4321     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4322     \__unravel_print_action:
4323     \__unravel_scan_int:
4324     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4325   }
4326   \__unravel_assign_toks:
4327 }
4328 \__unravel_prefixed_new:nn { assign_toks } % 72
4329 { \__unravel_assign_toks: }
4330 \cs_new_protected:Npn \__unravel_assign_toks:
4331 {
4332   \__unravel_prev_input_silent:V \l__unravel_head_tl
4333   \__unravel_print_action:
4334   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4335   \__unravel_scan_optional_equals:
4336   \__unravel_get_x_non_relax:
4337   \__unravel_set_cmd:
4338   \int_compare:nNnTF
4339   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
4340   {
4341     \__unravel_prev_input:V \l__unravel_head_tl
4342     \int_compare:nNnT \l__unravel_head_char_int = 0
4343     { \__unravel_scan_int: }
4344   }
4345   {

```

```

4346     \int_compare:nNnTF
4347     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
4348     { \__unravel_prev_input:V \l__unravel_head_tl }
4349     {
4350         \__unravel_back_input:
4351         \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4352     }
4353 }
4354 \__unravel_assign_register:
4355 }

```

(End of definition for \\_\_unravel\_assign\_toks:.)

```

4356 \__unravel_prefixed_new:nn { assign_int } % 73
4357 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4358 \__unravel_prefixed_new:nn { assign_dimen } % 74
4359 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4360 \__unravel_prefixed_new:nn { assign_glue } % 75
4361 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
4362 \__unravel_prefixed_new:nn { assign_mu_glue } % 76
4363 { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
4364 \__unravel_prefixed_new:nn { assign_font_dimen } % 77
4365 {
4366     \__unravel_assign_value:nn
4367     { \__unravel_scan_int: \__unravel_scan_font_ident: }
4368     { \__unravel_scan_normal_dimen: }
4369 }
4370 \__unravel_prefixed_new:nn { assign_font_int } % 78
4371 {
4372     \__unravel_assign_value:nn
4373     { \__unravel_scan_font_int: } { \__unravel_scan_int: }
4374 }
4375 \__unravel_prefixed_new:nn { set_aux } % 79
4376 { % prevdepth = 1, spacefactor = 102
4377     \int_compare:nNnTF \l__unravel_head_char_int = 1
4378     { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4379     { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4380 }
4381 \__unravel_prefixed_new:nn { set_prev_graf } % 80
4382 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4383 \__unravel_prefixed_new:nn { set_page_dimen } % 81
4384 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4385 \__unravel_prefixed_new:nn { set_page_int } % 82
4386 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4387 \__unravel_prefixed_new:nn { set_box_dimen } % 83
4388 {
4389     \__unravel_assign_value:nn
4390     { \__unravel_scan_int: } { \__unravel_scan_normal_dimen: }
4391 }

```

This is a variant of \\_\_unravel\_assign\_value:nn, with a bit more complication because the syntax of \parshape and of  $\epsilon$ -TeX primitives such as \interlinepenalties is a bit different.

```

4392 \__unravel_prefixed_new:nn { set_shape } % 84
4393 {

```

```

4394 \__unravel_prev_input:V \l__unravel_head_tl
4395 \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4396 \tl_if_head_eq_meaning:VNTF \l__unravel_defined_tl \tex_parshape:D
4397 {
4398   \__unravel_set_shape:NN 2 \__unravel_scan_normal_dimen:
4399   \__unravel_print_assigned_parshape:
4400 }
4401 {
4402   \__unravel_set_shape:NN 1 \__unravel_scan_int:
4403   \__unravel_print_assigned_set_shape:
4404 }
4405 }
4406 \cs_new_protected:Npn \__unravel_set_shape:NN #1#2
4407 {
4408   \__unravel_scan_optional_equals:
4409   \__unravel_prev_input_gpush:
4410   \__unravel_scan_int:
4411   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4412   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
4413   \prg_replicate:nn
4414     { \int_max:nn { 0 } { #1 * \l__unravel_tmpa_tl } }
4415     { \__unravel_prev_input_silent:n { ~ } #2 }
4416   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4417   \tl_use:N \l__unravel_tmpa_tl \scan_stop:
4418 }
4419 \__unravel_prefixed_new:nn { def_code } % 85
4420 {
4421   \__unravel_assign_value:nn
4422     { \__unravel_scan_int: } { \__unravel_scan_int: }
4423 }
4424 \__unravel_prefixed_new:nn { def_family } % 86
4425 {
4426   \__unravel_assign_value:nn
4427     { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
4428 }
4429 \__unravel_prefixed_new:nn { set_font } % 87
4430 {
4431   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4432   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4433   \tl_use:N \l__unravel_head_tl \scan_stop:
4434   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
4435   \__unravel_print_action:
4436 }
4437 \__unravel_prefixed_new:nn { def_font } % 88
4438 {
4439   \__unravel_prev_input_silent:V \l__unravel_head_tl
4440   \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
4441   \__unravel_scan_r_token:
4442   \__unravel_print_action:e
4443     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4444   \__unravel_scan_optional_equals:
4445   \__unravel_scan_file_name:
4446   \bool_gset_true:N \g__unravel_name_in_progress_bool
4447   \__unravel_scan_keyword:nTF { aAtT }

```

```

4448     { \__unravel_scan_normal_dimen: }
4449     {
4450         \__unravel_scan_keyword:nT { sScCaAlLeEdD }
4451         { \__unravel_scan_int: }
4452     }
4453     \bool_gset_false:N \g__unravel_name_in_progress_bool
4454     \__unravel_assign_token:n { }
4455 }

```

register=89, advance=90, multiply=91, divide=92 are implemented elsewhere.  
prefix=93 is never needed (see explanation above).

let, futurelet

```

4456 \__unravel_prefixed_new:nm { let } % 94
4457 {
4458     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4459     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4460     { % |let|
4461         \__unravel_scan_r_token:
4462         \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4463         \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
4464         \__unravel_get_next:
4465         \bool_while_do:nm
4466         { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4467         { \__unravel_get_next: }
4468         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4469         { \__unravel_get_next: }
4470         \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4471         { \__unravel_get_next: }
4472     }
4473     { % |futurelet|
4474         \__unravel_scan_r_token:
4475         \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4476         \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
4477         \__unravel_get_next:
4478         \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4479         \__unravel_get_next:
4480         \__unravel_back_input:
4481         \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4482         \__unravel_back_input:
4483     }
4484     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4485     \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4486     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4487     \use:e
4488     {
4489         \exp_not:V \l__unravel_head_tl
4490         \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4491     }
4492     \__unravel_print_assigned_token:
4493 }
4494 \__unravel_prefixed_new:nm { shorthand_def } % 95
4495 {
4496     \__unravel_prev_input_silent:V \l__unravel_head_tl
4497     \tl_set:Ne \l__unravel_prev_action_tl

```

```

4498     { \tl_to_str:N \l__unravel_head_tl }
4499   \__unravel_scan_r_token:
4500   \__unravel_print_action:e
4501     { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4502   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4503   \__unravel_just_print_assigned_token:
4504   \__unravel_scan_optional_equals:
4505   \__unravel_scan_int:
4506   \__unravel_assign_token:n { }
4507 }

```

`\__unravel_read_to_cs_safe:nTF`  
`\__unravel_read_to_cs_safe:fTF`

After `\read` or `\readline`, find an int, the mandatory keyword `to`, and an assignable token. The `\read` and `\readline` primitives throw a fatal error in `\nonstopmode` and in `\batchmode` when trying to read from a stream that is outside `[0,15]` or that is not open (according to `\ifeof`). We detect this situation using `\__unravel_read_to_cs_safe:nTF` after grabbing all arguments of the primitives. If reading is unsafe, let the user know that `TeX` would have thrown a fatal error.

```

4508 \__unravel_prefixed_new:nm { read_to_cs } % 96
4509 {
4510   \__unravel_prev_input_silent:V \l__unravel_head_tl
4511   \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
4512   \__unravel_scan_int:
4513   \__unravel_scan_to:
4514   \__unravel_scan_r_token:
4515   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4516   \__unravel_read_to_cs_safe:fTF
4517     { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
4518     { \__unravel_assign_token:n { } }
4519   {
4520     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4521     \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4522   }
4523 }
4524 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
4525 {
4526   \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
4527   { \prg_return_true: }
4528   {
4529     \int_compare:nNnTF {#1} < { 0 }
4530     { \prg_return_false: }
4531     {
4532       \int_compare:nNnTF {#1} > { 15 }
4533       { \prg_return_false: }
4534       {
4535         \tex_ifeof:D #1 \exp_stop_f:
4536         \prg_return_false:
4537         \else:
4538         \prg_return_true:
4539         \fi:
4540       }
4541     }
4542   }
4543 }
4544 \cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }

```

(End of definition for `\_unravel_read_to_cs_safe:nTF`.)

```

4545 \_unravel_prefixed_new:nn { def } % 97
4546 {
4547   \_unravel_prev_input_get:N \l__unravel_tmpa_tl
4548   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4549   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4550   \_unravel_prev_input_gpush:N \l__unravel_head_tl
4551   \int_compare:nNnTF \l__unravel_head_char_int < 2
4552     { % def/gdef
4553       \_unravel_scan_r_token:
4554       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4555       \_unravel_scan_toks:NN \c_true_bool \c_false_bool
4556     }
4557     { % edef/xdef
4558       \_unravel_scan_r_token:
4559       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4560       \_unravel_scan_toks:NN \c_true_bool \c_true_bool
4561     }
4562   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4563   \_unravel_prev_input:V \l__unravel_head_tl
4564   \_unravel_assign_token:n
4565   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4566 }

```

`\setbox` is a bit special: directly put it in the previous-input sequence with the prefixes; the box code will take care of things, and expects a single item containing what it needs to do.

```

4567 \_unravel_prefixed_new:nn { set_box } % 98
4568 {
4569   \_unravel_prev_input:V \l__unravel_head_tl
4570   \_unravel_scan_int:
4571   \_unravel_scan_optional_equals:
4572   \bool_if:NTF \g__unravel_set_box_allowed_bool
4573     { \_unravel_do_box:N \c_false_bool }
4574     {
4575       \_unravel_error:nnnnn { improper-setbox } { } { } { } { }
4576       \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4577       \_unravel_omit_after_assignment:w
4578     }
4579 }

```

#### `\hyphenation` and `\patterns`

```

4580 \_unravel_prefixed_new:nn { hyph_data } % 99
4581 {
4582   \_unravel_prev_input:V \l__unravel_head_tl
4583   \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4584   \_unravel_assign_token:n { }
4585 }

```

```

4586 \_unravel_prefixed_new:nn { set_interaction } % 100
4587 {
4588   \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4589   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4590   \tl_use:N \l__unravel_head_tl \scan_stop:
4591   \_unravel_print_assignment:e { \tl_to_str:N \l__unravel_head_tl }

```

```

4592 }
4593 \__unravel_prefixed_new:n { letterspace_font } % 101
4594 {
4595   \__unravel_prev_input_silent:V \l__unravel_head_tl
4596   \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
4597   \__unravel_scan_r_token:
4598   \__unravel_print_action:e
4599   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4600   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4601   \__unravel_just_print_assigned_token:
4602   \__unravel_scan_optional_equals:
4603   \__unravel_scan_font_ident:
4604   \__unravel_scan_int:
4605   \__unravel_assign_token:n { }
4606 }
4607 \__unravel_prefixed_new:n { pdf_copy_font } % 102
4608 {
4609   \__unravel_prev_input_silent:V \l__unravel_head_tl
4610   \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
4611   \__unravel_scan_r_token:
4612   \__unravel_print_action:e
4613   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4614   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4615   \__unravel_just_print_assigned_token:
4616   \__unravel_scan_optional_equals:
4617   \__unravel_scan_font_ident:
4618   \__unravel_assign_token:n { }
4619 }

```

Changes to numeric registers (`\count`, `\dimen`, `\skip`, `\muskip`, and commands with a built-in number).

```

4620 \__unravel_prefixed_new:n { register } % 89
4621 { \__unravel_do_register:N 0 }
4622 \__unravel_prefixed_new:n { advance } % 90
4623 { \__unravel_do_operation:N 1 }
4624 \__unravel_prefixed_new:n { multiply } % 91
4625 { \__unravel_do_operation:N 2 }
4626 \__unravel_prefixed_new:n { divide } % 92
4627 { \__unravel_do_operation:N 3 }

```

`\__unravel_do_operation:N`

`\__unravel_do_operation_fail:w`

```

4628 \cs_new_protected:Npn \__unravel_do_operation:N #1
4629 {
4630   \__unravel_prev_input_silent:V \l__unravel_head_tl
4631   \__unravel_print_action:
4632   \__unravel_get_x_next:
4633   \__unravel_set_cmd:
4634   \int_compare:nNnTF
4635     \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4636     {
4637       \int_compare:nNnTF
4638         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4639         { \__unravel_do_register:N #1 }

```

```

4640         { \_unravel_do_operation_fail:w }
4641     }
4642     {
4643     \int_compare:nNnTF
4644     \l__unravel_head_cmd_int < { \_unravel_tex_use:n { assign_int } }
4645     { \_unravel_do_operation_fail:w }
4646     {
4647     \_unravel_prev_input:V \l__unravel_head_tl
4648     \exp_args:NNf \_unravel_do_register_set:Nn #1
4649     {
4650     \int_eval:n
4651     {
4652     \l__unravel_head_cmd_int
4653     - \_unravel_tex_use:n { assign_toks }
4654     }
4655     }
4656     }
4657     }
4658 }
4659 \cs_new_protected:Npn \_unravel_do_operation_fail:w
4660 {
4661     \_unravel_error:nnnnn { after-advance } { } { } { } { }
4662     \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4663     \_unravel_omit_after_assignment:w
4664 }

```

(End of definition for \\_unravel\_do\_operation:N and \\_unravel\_do\_operation\_fail:w.)

```

\_unravel_do_register:N
  \_unravel_do_register_aux:Nn
4665 \cs_new_protected:Npn \_unravel_do_register:N #1
4666 {
4667     \exp_args:NNV \_unravel_do_register_aux:Nn #1
4668     \l__unravel_head_char_int
4669 }
4670 \cs_new_protected:Npn \_unravel_do_register_aux:Nn #1#2
4671 {
4672     \int_compare:nNnTF { \tl_tail:n {#2} } = 0
4673     {
4674     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4675     \_unravel_print_assignment:
4676     \_unravel_scan_int:
4677     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4678     \_unravel_prev_input_silent:V \l__unravel_head_tl
4679     }
4680     {
4681     \_unravel_prev_input_silent:V \l__unravel_head_tl
4682     \_unravel_print_assignment:
4683     }
4684     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4685     \exp_args:NNf \_unravel_do_register_set:Nn #1
4686     { \int_eval:n { #2 / 1 000 000 } }
4687 }

```

(End of definition for \\_unravel\_do\_register:N and \\_unravel\_do\_register\_aux:Nn.)



`\__unravel_do_register_set:Nn`

```
4688 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
4689 {
4690   \int_compare:nNnTF {#1} = 0
4691     { % truly register command
4692       \__unravel_scan_optional_equals:
4693     }
4694     { % \advance, \multiply, \divide
4695       \__unravel_scan_keyword:nF { bByY }
4696       { \__unravel_prev_input_silent:n { by } }
4697     }
4698   \int_compare:nNnTF {#1} < 2
4699     {
4700       \int_case:nnF {#2}
4701       {
4702         { 1 } { \__unravel_scan_int:          } % count
4703         { 2 } { \__unravel_scan_normal_dimen: } % dim
4704         { 3 } { \__unravel_scan_normal_glue:  } % glue
4705         { 4 } { \__unravel_scan_mu_glue:     } % muglue
4706       }
4707       { \__unravel_error:neeee { internal } { do-reg=#2 } { } { } { } }
4708     }
4709     { \__unravel_scan_int: }
4710   \__unravel_assign_register:
4711 }
```

*(End of definition for \\_\_unravel\_do\_register\_set:Nn.)*

The following is used for instance when making accents.

```
4712 \cs_new_protected:Npn \__unravel_do_assignments:
4713 {
4714   \__unravel_get_x_non_relax:
4715   \__unravel_set_cmd:
4716   \int_compare:nNnT
4717     \l__unravel_head_cmd_int
4718     > { \__unravel_tex_use:n { max_non_prefixed_command } }
4719     {
4720       \bool_gset_false:N \g__unravel_set_box_allowed_bool
4721       \__unravel_prev_input_gpush:
4722       \__unravel_prefixed_command:
4723       \bool_gset_true:N \g__unravel_set_box_allowed_bool
4724       \__unravel_do_assignments:
4725     }
4726 }
```

## 2.13 Expandable primitives

This section implements expandable primitives, which have the following command codes:

- `undefined_cs=103` for undefined control sequences (not quite a primitive).
- `expand_after=104` for `\expandafter` and `\unless`.
- `no_expand=105` for `\noexpand` and `\pdfprimitive`.
- `input=106` for `\input`, `\endinput` and `\scantokens`.

- `if_test=107` for the conditionals, `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\if[vhm]mode`, `\if[tydm]dir`, `\ifinner`, `\ifvoid`, `\if[hvtydm]box`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifprimitive`, `\ifabsnum`, `\ifabsdim`, `\ifjfont`, `\iftfont`.
- `fi_or_else=108` for `\fi`, `\else` and `\or`.
- `cs_name=109` for `\csname` and `\lastnamedcs`.
- `convert=110` for `\number`, `\romannumeral`, `\string`, `\meaning`, `\fontname`, `\eTeXrevision`, `\pdftexrevision`, `\pdftexbanner`, `\pdffontname`, `\pdffontobjnum`, `\pdffontsize`, `\pdfpageref`, `\pdfxformname`, `\pdfescapestring`, `\pdfescapename`, `\leftmarginkern`, `\rightmarginkern`, `\pdfstrcmp`, `\pdfcolorstackinit`, `\pdfescapehex`, `\pdfunescapehex`, `\pdfcreationdate`, `\pdffilemoddate`, `\pdffilesize`, `\pdfmdfivesum`, `\pdffiledump`, `\pdfmatch`, `\pdflastmatch`, `\pdfuniformdeviate`, `\pdfnormaldeviate`, `\pdfinsertht`, `\pdfximagebbox`, `\jobname`, `\expanded`, and in Lua $\TeX$  `\directlua`, `\luaescapestring`, and in X $\TeX$ /(u)p $\TeX$  `\Ucharcat`.
- `the=111` for `\the`, `\unexpanded`, and `\detokenize`.
- `top_bot_mark=112` `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`, `\topmarks`, `\firstmarks`, `\botmarks`, `\splitfirstmarks`, and `\splitbotmarks`.
- `call=113` for macro calls, implemented by `\__unravel_macro_call:`.
- `end_template=117` for  $\TeX$ 's end template.

Let  $\TeX$  trigger an error.

```

4727 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
4728 { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }

\__unravel_expandafter:
  \__unravel_unless:
\__unravel_unless_bad:
4729 \__unravel_new_tex_expandable:nn { expand_after } % 104
4730 {
4731   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4732   { \__unravel_expandafter: } { \__unravel_unless: }
4733 }
4734 \cs_new_protected:Npn \__unravel_expandafter:
4735 {
4736   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4737   \__unravel_get_next:
4738   \gtl_concat:NNN \l__unravel_head_gtl
4739   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4740   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4741   \__unravel_print_expansion:e { \gtl_to_str:N \l__unravel_head_gtl }
4742   \__unravel_get_next:
4743   \__unravel_token_if_expandable:NTF \l__unravel_head_token
4744   { \__unravel_expand_do:N \prg_do_nothing: }
4745   { \__unravel_back_input: }
4746   \__unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
4747   \__unravel_set_action_text:e
4748   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4749   \gtl_pop_left:N \l__unravel_head_gtl
4750   \__unravel_back_input:

```

```

4751     \__unravel_print_expansion:
4752   }
4753 \cs_new_protected:Npn \__unravel_unless:
4754 {
4755   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4756   \__unravel_get_token:
4757   \int_compare:nNnTF
4758     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4759   {
4760     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4761     { \__unravel_unless_bad: }
4762     {
4763       \tl_put_left:Ne \l__unravel_head_tl
4764       { \gtl_head_do:NN \l__unravel_tmpb_gtl \exp_not:N }
4765       \__unravel_expand_nonmacro:
4766     }
4767   }
4768   { \__unravel_unless_bad: }
4769 }
4770 \cs_new_protected:Npn \__unravel_unless_bad:
4771 {
4772   \__unravel_error:nnnnn { bad-unless } { } { } { } { }
4773   \__unravel_back_input:
4774 }

```

(End of definition for \\_\_unravel\_expandafter:, \\_\_unravel\_unless:, and \\_\_unravel\_unless\_bad:.)

\\_\_unravel\_noexpand:N Currently not fully implemented.  
 \\_\_unravel\_noexpand\_after: The argument of \\_\_unravel\_noexpand:N is \prg\_do\_nothing: when \noexpand  
 \\_\_unravel\_pdfprimitive: is hit by \expandafter; otherwise it is one of various loop commands (\\_\_unravel\_get\_x\_next:, \\_\_unravel\_get\_x\_or\_protected:, \\_\_unravel\_get\_token\_xdef:, \\_\_unravel\_get\_token\_x:) that would call \\_\_unravel\_get\_next: and possibly expand the token more. For these cases we simply stop after \\_\_unravel\_get\_next: and if the token is expandable we pretend its meaning is \relax.

The case of \expandafter (so \prg\_do\_nothing:) is tougher. Do nothing if the next token is an explicit non-active character (begin-group and end-group characters are detected by \l\_\_unravel\_head\_tl, the rest by testing if the token is definable). Otherwise the token must be marked with \notexpanded: (even if the token is currently a non-expandable primitive, as its meaning can be changed by the code skipped over by \expandafter). That \notexpanded: marker should be removed if the token is taken as the argument of a macro, but we fail to do that. We set the \notexpanded:... command to be a special \relax marker to make it quickly recognizable in \\_\_unravel\_get\_next:. This is incidentally the same meaning used by T<sub>E</sub>X for expandable commands.

```

4775 \__unravel_new_tex_expandable:nm { no_expand } % 105
4776 {
4777   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4778   { \__unravel_noexpand:N }
4779   { \__unravel_pdfprimitive: }
4780 }
4781 \cs_new_protected:Npn \__unravel_noexpand:N #1
4782 {
4783   \__unravel_get_token:
4784   \cs_if_eq:NNTF #1 \prg_do_nothing:

```

```

4785     {
4786     \tl_if_empty:NTF \l__unravel_head_tl
4787     { \__unravel_back_input: }
4788     {
4789     \exp_after:wN \__unravel_token_if_definable:NTF \l__unravel_head_tl
4790     { \__unravel_noexpand_after: }
4791     { \__unravel_back_input: }
4792     }
4793     }
4794     {
4795     \__unravel_back_input:
4796     \__unravel_get_next:
4797     \__unravel_token_if_expandable:NT \l__unravel_head_token
4798     { \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax: }
4799     }
4800   }
4801 \cs_new_protected:Npn \__unravel_noexpand_after:
4802 {
4803   \group_begin:
4804   \__unravel_set_escapechar:n { 92 }
4805   \exp_args:NNc
4806   \group_end:
4807   \__unravel_noexpand_after:N
4808   { notexpanded: \exp_after:wN \token_to_str:N \l__unravel_head_tl }
4809 }
4810 \cs_new_protected:Npn \__unravel_noexpand_after:N #1
4811 {
4812   \cs_gset_eq:NN #1 \__unravel_special_relax:
4813   \__unravel_back_input:n {#1}
4814 }
4815 \cs_new_protected:Npn \__unravel_pdfprimitive:
4816 { \__unravel_not_implemented:n { pdfprimitive } }

```

(End of definition for \\_\_unravel\_noexpand:N, \\_\_unravel\_noexpand\_after:, and \\_\_unravel\_pdfprimitive:.)

```

\__unravel_endinput:
\__unravel_scantokens: 4817 \__unravel_new_tex_expandable:nn { input } % 106
\__unravel_input: 4818 {
4819   \int_case:nnF \l__unravel_head_char_int
4820   {
4821     { 1 } { \__unravel_endinput: } % \endinput
4822     { 2 } { \__unravel_scantokens: } % \scantokens
4823   }
4824   { % 0=\input
4825     \bool_if:NTF \g__unravel_name_in_progress_bool
4826     { \__unravel_insert_relax: } { \__unravel_input: }
4827   }
4828 }
4829 \cs_new_protected:Npn \__unravel_endinput:
4830 {
4831   \group_begin:
4832   \msg_warning:nn { unravel } { endinput-ignored }
4833   \group_end:
4834   \__unravel_print_expansion:

```

```

4835 }
4836 \cs_new_protected:Npn \__unravel_scantokens:
4837 {
4838   \__unravel_prev_input_gpush:
4839   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4840   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4841   \exp_last_unbraced:NNNo \tl_set_rescan:Nnn
4842   \l__unravel_head_tl \prg_do_nothing: \l__unravel_tmpa_tl
4843   \__unravel_back_input:V \__unravel_everyeof:w
4844   \__unravel_back_input:V \l__unravel_head_tl
4845   \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_tmpa_tl }
4846 }
4847 \cs_new_protected:Npn \__unravel_input:
4848 {
4849   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4850   \__unravel_scan_file_name:
4851   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4852   \tl_set:Ne \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4853   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4854   \__unravel_back_input:V \l__unravel_tmpa_tl
4855   \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_head_tl }
4856 }

```

(End of definition for \\_\_unravel\_endinput:, \\_\_unravel\_scantokens:, and \\_\_unravel\_input:.)

\\_\_unravel\_csname\_loop:

```

4857 \__unravel_new_tex_expandable:nm { cs_name } % 109
4858 {
4859   \int_compare:nNnTF \l__unravel_head_char_int = 0
4860   {
4861     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4862     \__unravel_print_expansion:
4863     \__unravel_csname_loop:
4864     \__unravel_prev_input_silent:V \l__unravel_head_tl
4865     \__unravel_get_lastnamedcs:
4866     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4867     \__unravel_back_input_tl_o:
4868   }
4869   {
4870     \__unravel_back_input:V \g__unravel_lastnamedcs_tl
4871     \__unravel_print_expansion:e
4872     { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \g__unravel_lastnamedcs_tl }
4873   }
4874 }
4875 \cs_new_protected:Npn \__unravel_csname_loop:
4876 {
4877   \__unravel_get_x_next:
4878   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4879   {
4880     \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4881     {
4882       \__unravel_back_input:
4883       \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4884       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }

```

```

4885     }
4886   }
4887   {
4888     \__unravel_prev_input_silent:e
4889     { \__unravel_token_to_char:N \l__unravel_head_token }
4890     \__unravel_csname_loop:
4891   }
4892 }
4893 \cs_new_protected:Npn \__unravel_get_lastnamedcs:
4894 {
4895   \group_begin:
4896   \__unravel_prev_input_get:N \l__unravel_head_tl
4897   \tl_gset:No \g__unravel_lastnamedcs_tl
4898   { \cs:w \exp_after:wN \__unravel_get_lastnamedcs_check:N \l__unravel_head_tl }
4899   \group_end:
4900 }
4901 \cs_new:Npn \__unravel_get_lastnamedcs_check:N #1
4902 { \if_meaning:w \reverse_if:N #1 \use_i:nn \fi: }

(End of definition for \__unravel_csname_loop:.)

4903 \__unravel_new_tex_expandable:nn { convert } % 110
4904 {
4905   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4906   \__unravel_print_expansion:
4907   \int_case:nn \l__unravel_head_char_int
4908   {
4909     0    \__unravel_scan_int:
4910     1    \__unravel_scan_int:
4911     2    \__unravel_convert_string:
4912     3    \__unravel_convert_meaning:w
4913     4    \__unravel_scan_font_ident:
4914     8    \__unravel_scan_font_ident:
4915     9    \__unravel_scan_font_ident:
4916     { 10 } \__unravel_scan_font_ident:
4917     { 11 } \__unravel_scan_int:
4918     { 12 } \__unravel_scan_int:
4919     { 13 } \__unravel_scan_pdf_ext_toks:
4920     { 14 } \__unravel_scan_pdf_ext_toks:
4921     { 15 } \__unravel_scan_int:
4922     { 16 } \__unravel_scan_int:
4923     { 17 } \__unravel_scan_pdfstrcmp:
4924     { 18 } \__unravel_scan_pdfcolorstackinit:
4925     { 19 } \__unravel_scan_pdf_ext_toks:
4926     { 20 } \__unravel_scan_pdf_ext_toks:
4927     { 22 } \__unravel_scan_pdf_ext_toks:
4928     { 23 } \__unravel_scan_pdf_ext_toks:
4929     { 24 }
4930     {
4931       \__unravel_scan_keyword:n { fFillLeE }
4932       \__unravel_scan_pdf_ext_toks:
4933     }
4934     { 25 } \__unravel_scan_pdffiledump:
4935     { 26 } \__unravel_scan_pdfmatch:
4936     { 27 } \__unravel_scan_int:

```

```

4937     { 28 } \__unravel_scan_int:
4938     { 30 } \__unravel_scan_int:
4939     { 31 } \__unravel_scan_pdfimagebbox:
4940     { 33 } \__unravel_scan_directlua:
4941     { 34 } \__unravel_scan_pdf_ext_toks:
4942     { 35 } \__unravel_scan_pdf_ext_toks:
4943     { 40 }
4944     {
4945         \__unravel_scan_int:
4946         \__unravel_prev_input_silent:n { ~ }
4947         \__unravel_scan_int:
4948     }
4949 }
4950 \__unravel_prev_input_gpop:N \l__unravel_head_tl
4951 \__unravel_back_input_tl_o:
4952 }
4953 \cs_new_protected:Npn \__unravel_convert_string:
4954 {
4955     \__unravel_get_next:
4956     \tl_if_empty:NTF \l__unravel_head_tl
4957     { \__unravel_prev_input:e { \gtl_to_str:N \l__unravel_head_gtl } }
4958     { \__unravel_prev_input:V \l__unravel_head_tl }
4959 }
4960 \cs_new_protected:Npn \__unravel_convert_meaning:w
4961 \__unravel_prev_input_gpop:N \l__unravel_head_tl \__unravel_back_input_tl_o:
4962 {
4963     \__unravel_get_next:
4964     \tl_if_empty:NTF \l__unravel_head_tl
4965     {
4966         \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4967         \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
4968         \exp_args:NNV \gtl_put_left:Nn \l__unravel_tmpb_gtl \l__unravel_prev_input_tl
4969         \__unravel_prev_input_gpush_gtl:N \l__unravel_tmpb_gtl
4970         \__unravel_print_action:e { \gtl_to_str:N \l__unravel_tmpb_gtl }
4971         \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
4972         \tl_set:Ne \l__unravel_tmpa_tl { \gtl_head_do:NN \l__unravel_head_gtl \tex_meaning:D
4973         \__unravel_back_input:V \l__unravel_tmpa_tl
4974         \__unravel_print_expansion:e
4975         { \gtl_to_str:N \l__unravel_tmpb_gtl = \tl_to_str:N \l__unravel_tmpa_tl }
4976     }
4977     {
4978         \__unravel_prev_input:V \l__unravel_head_tl
4979         \__unravel_prev_input_gpop:N \l__unravel_head_tl
4980         \__unravel_back_input_tl_o:
4981     }
4982 }
4983 \cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
4984 {
4985     \__unravel_scan_toks_to_str:
4986     \__unravel_scan_toks_to_str:
4987 }
4988 \cs_new_protected:Npn \__unravel_scan_pdfimagebbox:
4989 { \__unravel_scan_int: \__unravel_scan_int: }
4990 \cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:

```

```

4991 {
4992   \__unravel_scan_keyword:nTF { pPaAgGeE }
4993     { \bool_set_true:N \l__unravel_tmpa_bool }
4994     { \bool_set_false:N \l__unravel_tmpb_bool }
4995   \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4996     { \__unravel_scan_keyword:n { pPaAgGeE } }
4997   \__unravel_scan_toks_to_str:
4998 }
4999 \cs_new_protected:Npn \__unravel_scan_pdffiledump:
5000 {
5001   \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
5002   \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
5003   \__unravel_scan_pdf_ext_toks:
5004 }
5005 \cs_new_protected:Npn \__unravel_scan_pdfmatch:
5006 {
5007   \__unravel_scan_keyword:n { iIcCaAsSeE }
5008   \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
5009   { \__unravel_scan_int: }
5010   \__unravel_scan_pdf_ext_toks:
5011   \__unravel_scan_pdf_ext_toks:
5012 }
5013 \sys_if_engine luatex:T
5014 {
5015   \cs_new_protected:Npn \__unravel_scan_directlua:
5016     {
5017       \__unravel_get_x_non_relax:
5018       \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
5019         { \__unravel_back_input: }
5020         {
5021           \__unravel_scan_int:
5022           \__unravel_get_x_non_relax:
5023         }
5024       \__unravel_scan_pdf_ext_toks:
5025     }
5026 }

```

\\_\_unravel\_get\_the:N #1 is \\_\_unravel\_get\_token\_xdef: in \edef or \xdef, \\_\_unravel\_get\_token\_x: in \message and the like, and can be other commands.

```

5027 \__unravel_new_tex_expandable:nn { the } % 111
5028 { \__unravel_get_the:N }
5029 \cs_new_protected:Npn \__unravel_get_the:N #1
5030 {
5031   \__unravel_prev_input_gpush:N \l__unravel_head_tl
5032   \__unravel_print_expansion:
5033   \int_if_odd:nTF \l__unravel_head_char_int
5034     { % \unexpanded, \detokenize
5035       \__unravel_scan_toks:NN \c_false_bool \c_false_bool
5036       \__unravel_prev_input_gpop:N \l__unravel_head_tl
5037       \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
5038     }
5039     { % \the
5040       \__unravel_get_x_next:
5041       \__unravel_rescan_something_internal:n { 5 }

```



```

5042     \__unravel_prev_input_gpop:N \l__unravel_head_tl
5043     \__unravel_set_action_text:e
5044     {
5045         \tl_head:N \l__unravel_head_tl
5046         => \tl_tail:N \l__unravel_head_tl
5047     }
5048     \tl_set:Ne \l__unravel_head_tl
5049     { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
5050 }
5051 \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
5052 {
5053     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
5054     \__unravel_prev_input_silent:e { \l__unravel_head_tl }
5055     \__unravel_print_action:
5056 }
5057 {
5058     \cs_if_eq:NNTF #1 \__unravel_get_token_x:
5059     {
5060         \exp_args:NNe \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
5061         \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
5062     }
5063     {
5064         \tl_set:Ne \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
5065         \__unravel_back_input:V \l__unravel_tmpa_tl
5066     }
5067     \__unravel_print_expansion:
5068 }
5069 #1
5070 }

```

(End of definition for \\_\_unravel\_get\_the:N.)

```

5071 \__unravel_new_tex_expandable:nn { top_bot_mark } % 112
5072 { \__unravel_back_input_tl_o: }
5073 \__unravel_new_tex_expandable:nn { end_template } % 117
5074 { \__unravel_back_input_tl_o: }

```

### 2.13.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w
5075 \cs_new_protected:Npn \__unravel_pass_text:
5076 {
5077     \__unravel_input_if_empty:TF
5078     { \__unravel_pass_text_empty: }
5079     {
5080         \__unravel_input_get:N \l__unravel_tmpb_gtl
5081         \if_true:
5082             \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
5083             \exp_after:wN \__unravel_pass_text_done:w
5084             \fi:
5085             \__unravel_input_gpop:N \l__unravel_tmpb_gtl
5086             \exp_after:wN \__unravel_pass_text:
5087         \else:
5088             \use:c { fi: }

```

```

5089         \int_set:Nn \l__unravel_if_nesting_int { 1 }
5090         \__unravel_input_gpop:N \l__unravel_tmpb_gtl
5091         \exp_after:wN \__unravel_pass_text_nested:
5092     \fi:
5093 }
5094 }
5095 \cs_new_protected:Npn \__unravel_pass_text_done:w
5096 {
5097     \__unravel_get_next:
5098     \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
5099     \else:
5100 }

```

(End of definition for \\_\_unravel\_pass\_text: and \\_\_unravel\_pass\_text\_done:w.)

\\_\_unravel\_pass\_text\_nested: Again, if there is no more input we are in trouble. The construction otherwise essentially results in

```

\if_true: \if_true: \else: <head>
\int_decr:N \l__unravel_if_nesting_int \use_none:nnnnn \fi:
\use_none:nnn \fi:
\int_incr:N \l__unravel_if_nesting_int \fi:

```

If the <head> is a primitive \if..., then the \if\_true: \else: ends with the second \fi:, and the nesting integer is incremented before appropriately closing the \if\_true:. If it is a normal token or \or or \else, \use\_none:nnn cleans up, leaving the appropriate number of \fi:. Finally, if it is \fi:, the nesting integer is decremented before removing most \fi:.

```

5101 \cs_new_protected:Npn \__unravel_pass_text_nested:
5102 {
5103     \__unravel_input_if_empty:TF
5104     { \__unravel_pass_text_empty: }
5105     {
5106         \__unravel_input_get:N \l__unravel_tmpb_gtl
5107         \if_true:
5108             \if_true:
5109                 \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
5110                 \int_decr:N \l__unravel_if_nesting_int
5111                 \use_none:nnnnn
5112             \fi:
5113             \use_none:nnn
5114         \fi:
5115         \int_incr:N \l__unravel_if_nesting_int
5116         \fi:
5117         \__unravel_input_gpop:N \l__unravel_unused_gtl
5118         \int_compare:nNnTF \l__unravel_if_nesting_int = 0
5119             { \__unravel_pass_text: }
5120             { \__unravel_pass_text_nested: }
5121     }
5122 }

```

(End of definition for \\_\_unravel\_pass\_text\_nested:.)

`\_unravel_pass_text_empty:`

```
5123 \cs_new_protected:Npn \_unravel_pass_text_empty:
5124 {
5125   \_unravel_error:nnnnn { runaway-if } { } { } { } { }
5126   \_unravel_exit_hard:w
5127 }
```

*(End of definition for \\_unravel\_pass\_text\_empty:.)*

`\_unravel_cond_push:`

`\_unravel_cond_pop:`

```
5128 \cs_new_protected:Npn \_unravel_cond_push:
5129 {
5130   \tl_gput_left:Ne \g__unravel_if_limit_tl
5131   { { \int_use:N \g__unravel_if_limit_int } }
5132   \int_gincr:N \g__unravel_if_depth_int
5133   \int_gzero:N \g__unravel_if_limit_int
5134 }
5135 \cs_new_protected:Npn \_unravel_cond_pop:
5136 {
5137   \fi:
5138   \int_gset:Nn \g__unravel_if_limit_int
5139   { \tl_head:N \g__unravel_if_limit_tl }
5140   \tl_gset:Ne \g__unravel_if_limit_tl
5141   { \tl_tail:N \g__unravel_if_limit_tl }
5142   \int_gdecr:N \g__unravel_if_depth_int
5143 }
```

*(End of definition for \\_unravel\_cond\_push: and \\_unravel\_cond\_pop:.)*

`\_unravel_change_if_limit:nn`

```
5144 \cs_new_protected:Npn \_unravel_change_if_limit:nn #1#2
5145 {
5146   \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
5147   { \int_gset:Nn \g__unravel_if_limit_int {#1} }
5148   {
5149     \tl_clear:N \l__unravel_tmpa_tl
5150     \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
5151     {
5152       \tl_put_right:Ne \l__unravel_tmpa_tl
5153       { { \tl_head:N \g__unravel_if_limit_tl } }
5154       \tl_gset:Ne \g__unravel_if_limit_tl
5155       { \tl_tail:N \g__unravel_if_limit_tl }
5156     }
5157     \tl_gset:Ne \g__unravel_if_limit_tl
5158     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
5159   }
5160 }
```

*(End of definition for \\_unravel\_change\_if\_limit:nn.)*

```
5161 \_unravel_new_tex_expandable:nn { if_test } % 107
5162 {
5163   \_unravel_cond_push:
5164   \exp_args:NV \_unravel_cond_aux:n \g__unravel_if_depth_int
5165 }
```

\\\_unravel\_cond\_aux:nn

```
5166 \cs_new_protected:Npn \\_unravel_cond_aux:n #1
5167 {
5168   \int_case:nnF \l__unravel_head_char_int
5169   {
5170     { 0 } { \\_unravel_test_two_chars:nn { 0 } {#1} } % if
5171     { 1 } { \\_unravel_test_two_chars:nn { 1 } {#1} } % ifcat
5172     { 12 } { \\_unravel_test_ifx:n {#1} }
5173     { 16 } { \\_unravel_test_case:n {#1} }
5174     { 20 } { \if_true: \\_unravel_test_incsname:n {#1} }
5175     { 21 } { \if_true: \\_unravel_test_pdfprimitive:n {#1} }
5176   }
5177   {
5178     \\_unravel_prev_input_gpush:N \l__unravel_head_tl
5179     \\_unravel_print_expansion:
5180     \int_case:nn \l__unravel_head_char_int
5181     {
5182       { 2 } % ifnum
5183       { \\_unravel_test_two_vals:N \\_unravel_scan_int: }
5184       { 3 } % ifdim
5185       { \\_unravel_test_two_vals:N \\_unravel_scan_normal_dimen: }
5186       { 4 } { \\_unravel_scan_int: } % ifodd
5187       % { 5 } { } % if[hvm]mode, ifinner, if[tydm]dir
5188       { 9 } { \\_unravel_scan_int: } % ifvoid, ifhbox, ifvbox etc
5189       { 13 } { \\_unravel_scan_int: } % ifeof
5190       % { 14 } { } % iftrue
5191       % { 15 } { } % iffalse
5192       { 17 } { \\_unravel_test_ifdefined: } % ifdefined
5193       { 18 } { \\_unravel_test_ifcsname: } % ifcsname
5194       { 19 } % iffontchar
5195       { \\_unravel_scan_font_ident: \\_unravel_scan_int: }
5196       { 22 } % ifabsnum
5197       { \\_unravel_test_two_vals:N \\_unravel_scan_int: }
5198       { 23 } % ifabsdim
5199       { \\_unravel_test_two_vals:N \\_unravel_scan_normal_dimen: }
5200       { 24 } { \\_unravel_scan_font_ident: } % ifjfont, iftfont
5201     }
5202     \\_unravel_prev_input_gpop:N \l__unravel_head_tl
5203     \\_unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
5204     \l__unravel_head_tl \scan_stop:
5205     \\_unravel_cond_true:NNNn
5206     \else:
5207     \\_unravel_cond_false:Nn
5208     \fi:
5209     {#1}
5210   }
5211 }
```

(End of definition for \\\_unravel\_cond\_aux:nn.)

\\\_unravel\_cond\_true:NNNn

```
5212 \cs_new_protected:Npn \\_unravel_cond_true:NNNn #1#2#3#4
5213 {
5214   \\_unravel_change_if_limit:nn { 3 } {#4} % wait for else/fi
```

```

5215     \_unravel_print_expansion:e { \g_unravel_action_text_str = true }
5216   }

```

(End of definition for \\_unravel\_cond\_true:NNNn.)

```

\_unravel_cond_false:Nn
\_unravel_cond_false_loop:n
  \_unravel_cond_false_common:
5217 \cs_new_protected:Npn \_unravel_cond_false:Nn #1#2
5218   {
5219     \_unravel_cond_false_loop:n {#2}
5220     \_unravel_cond_false_common:
5221     \_unravel_print_expansion:e
5222     {
5223       \g_unravel_action_text_str = false ~
5224       => ~ skip ~ to ~ \tl_to_str:N \l_unravel_head_tl
5225     }
5226   }
5227 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
5228   {
5229     \_unravel_pass_text:
5230     \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
5231     {
5232       \token_if_eq_meaning:NNT \l_unravel_head_token \or:
5233       {
5234         \_unravel_error:nnnnn { extra-or } { } { } { } { }
5235         \_unravel_cond_false_loop:n {#1}
5236       }
5237     }
5238     {
5239       \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
5240       { \_unravel_cond_pop: }
5241       \_unravel_cond_false_loop:n {#1}
5242     }
5243   }
5244 \cs_new_protected:Npn \_unravel_cond_false_common:
5245   {
5246     \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
5247     { \_unravel_cond_pop: }
5248     { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
5249   }

```

(End of definition for \\_unravel\_cond\_false:Nn, \\_unravel\_cond\_false\_loop:n, and \\_unravel\_cond\_false\_common:.)

```

\_unravel_test_two_vals:N
5250 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
5251   {
5252     #1
5253     \_unravel_get_x_non_blank:
5254     \_unravel_tl_if_in:ooTF { < = > } \l_unravel_head_tl { }
5255     {
5256       \_unravel_error:nnnnn { missing-equals } { } { } { } { }
5257       \_unravel_back_input:
5258       \tl_set:Nn \l_unravel_head_tl { = }
5259     }
5260     \_unravel_prev_input:V \l_unravel_head_tl

```

```

5261     #1
5262   }

```

(End of definition for `__unravel_test_two_vals:N`.)

```

__unravel_test_two_chars:nn

```

```

__unravel_test_two_chars_get:n 5263 \cs_new_protected:Npn __unravel_test_two_chars:nn #1
__unravel_test_two_chars_gtl:N 5264 {
5265   \exp_args:NNo \gtl_set:Nn \l__unravel_head_gtl { \l__unravel_head_tl }
5266   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
5267   \__unravel_print_expansion:
5268   \__unravel_test_two_chars_get:n {#1}
5269   \__unravel_test_two_chars_get:n {#1}
5270   \__unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
5271   \__unravel_set_action_text:e { \gtl_to_str:N \l__unravel_head_gtl }
5272   \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
5273   \exp_args:No \tl_if_head_eq_meaning:nNT \l__unravel_head_tl \reverse_if:N
5274   {
5275     \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
5276     \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
5277   }
5278   \gtl_pop_left:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
5279   \__unravel_test_two_chars_gtl:N \l__unravel_tmpb_gtl
5280   \__unravel_test_two_chars_gtl:N \l__unravel_head_gtl
5281   \l__unravel_head_tl \scan_stop:
5282   \__unravel_cond_true:NNNn
5283   \else:
5284   \__unravel_cond_false:Nn
5285   \fi:
5286 }
5287 \cs_new_protected:Npn __unravel_test_two_chars_get:n #1
5288 {
5289   \__unravel_get_x_next:
5290   \int_compare:nNnT {#1} = 0
5291   {
5292     \gtl_if_head_is_N_type:NF \l__unravel_head_gtl
5293     { \gtl_set:Ne \l__unravel_head_gtl { \gtl_to_str:N \l__unravel_head_gtl } }
5294   }
5295   \__unravel_prev_input_gtl:N \l__unravel_head_gtl
5296   \__unravel_print_action:e { \gtl_to_str:N \l__unravel_head_gtl }
5297 }
5298 \cs_new_protected:Npn __unravel_test_two_chars_gtl:N #1
5299 {
5300   \tl_put_right:Ne \l__unravel_head_tl
5301   {
5302     \gtl_if_head_is_group_begin:NTF #1 { \c_group_begin_token }
5303     {
5304       \gtl_if_head_is_group_end:NTF #1 { \c_group_end_token }
5305       {
5306         \exp_not:N \exp_not:N
5307         \exp_not:f { \gtl_head_do:NN #1 \exp_stop_f: }
5308       }
5309     }
5310   }
5311 }

```

(End of definition for `\_unravel_test_two_chars:nn`, `\_unravel_test_two_chars_get:n`, and `\_unravel_test_two_chars_gtl:N`.)

```

\__unravel_test_ifx:n The token equal to \ifx is pushed as a previous input to show an action nicely, then
\__unravel_test_ifx_str:NN retrieved as \l__unravel_tmpa_tl after getting the next two tokens as tmpb and head.
\__unravel_test_ifx_aux:NNN Then we call \l__unravel_tmpa_tl followed by these two tokens. A previous implemen-
\__unravel_test_ifx_aux:w tation made sure to get these tokens from unpacking the gtl, presumably (I should have
documented, now I might be missing something) to deal nicely with the master counter
in case these tokens are braces. On the other hand we must take care of tokens affected
by \noexpand and whose current definition is expandable, in which case the trustworthy
\meaning is that of the \l__unravel_head_token or \l__unravel_tmpb_token rather
than that of the token in \l__unravel_head_gtl or \l__unravel_tmpb_gtl.

5312 \cs_new_protected:Npn \__unravel_test_ifx:n #1
5313 {
5314   \__unravel_prev_input_gpush:N \l__unravel_head_tl
5315   \__unravel_print_expansion:
5316   \__unravel_get_next:
5317   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
5318   \cs_set_eq:NN \l__unravel_tmpb_token \l__unravel_head_token
5319   \__unravel_get_next:
5320   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
5321   \__unravel_set_action_text:e
5322   {
5323     Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
5324     \__unravel_test_ifx_str:NN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
5325     \__unravel_test_ifx_str:NN \l__unravel_head_token \l__unravel_head_gtl
5326   }
5327   \__unravel_test_ifx_aux:NNN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
5328   \__unravel_test_ifx_aux:w
5329   \__unravel_cond_true:NNN
5330   \else:
5331     \__unravel_cond_false:Nn
5332   \fi:
5333   {#1}
5334 }
5335 \cs_new:Npn \__unravel_test_ifx_str:NN #1#2
5336 {
5337   \token_if_eq_meaning:NNT #1 \__unravel_special_relax:
5338   { \iow_char:N \notexpanded: }
5339   \gtl_to_str:N #2
5340 }
5341 \cs_new_protected:Npn \__unravel_test_ifx_aux:NNN #1#2#3
5342 {
5343   \token_if_eq_meaning:NNTF #1 \__unravel_special_relax:
5344   {
5345     \gtl_head_do:NN #2 \__unravel_token_if_expandable:NTF
5346     { #3 #1 } { \gtl_head_do:NN #2 #3 }
5347   }
5348   { \gtl_head_do:NN #2 #3 }
5349 }
5350 \cs_new:Npn \__unravel_test_ifx_aux:w
5351 {
5352   \__unravel_test_ifx_aux:NNN \l__unravel_head_token \l__unravel_head_gtl

```

```

5353     \l__unravel_tmpa_tl
5354 }

```

(End of definition for \\_\_unravel\_test\_ifx:n and others.)

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn
5355 \cs_new_protected:Npn \__unravel_test_case:n #1
5356 {
5357   \if_case:w 0 ~
5358   \__unravel_prev_input_gpush:N \l__unravel_head_tl
5359   \__unravel_print_expansion:
5360   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
5361   \__unravel_scan_int:
5362   \__unravel_prev_input_get:N \l__unravel_head_tl
5363   \tl_set:Ne \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
5364   % ^A does text_case_aux use prev_input_seq?
5365   \int_compare:nNnF { \l__unravel_head_tl } = 0
5366   {
5367     \int_compare:nNnTF { \l__unravel_head_tl } > 0
5368     { \fi: \if_case:w 1 ~ \or: }
5369     { \fi: \if_case:w -1 ~ \else: }
5370   }
5371   \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
5372   \__unravel_prev_input_gpop:N \l__unravel_head_tl
5373   \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_head_tl }
5374 }
5375 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
5376 {
5377   \int_compare:nNnTF {#1} = 0
5378   { \__unravel_change_if_limit:nn { 4 } {#2} }
5379   {
5380     \__unravel_pass_text:
5381     \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
5382     {
5383       \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
5384       {
5385         \exp_args:Nf \__unravel_test_case_aux:nn
5386         { \int_eval:n { #1 - 1 } } {#2}
5387       }
5388       { \__unravel_cond_false_common: }
5389     }
5390     {
5391       \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
5392       { \__unravel_cond_pop: }
5393       \__unravel_test_case_aux:nn {#1} {#2}
5394     }
5395   }
5396 }

```

(End of definition for \\_\_unravel\_test\_case:n and \\_\_unravel\_test\_case\_aux:nn.)

```

\__unravel_test_incsname:n
5397 \cs_new_protected:Npn \__unravel_test_incsname:n #1
5398 { \__unravel_not_implemented:n { ifincsname } }

```



(End of definition for \\_unravel\_test\_incsname:n.)

\\_unravel\_test\_pdfprimitive:n

```
5399 \cs_new_protected:Npn \_unravel_test_pdfprimitive:n #1
5400 { \_unravel_not_implemented:n { ifpdfprimitive } }
```

(End of definition for \\_unravel\_test\_pdfprimitive:n.)

\\_unravel\_test\_ifdefined:

```
5401 \cs_new_protected:Npn \_unravel_test_ifdefined:
5402 {
5403   \_unravel_input_if_empty:TF
5404   { \_unravel_pass_text_empty: }
5405   {
5406     \_unravel_input_gpop:N \l__unravel_tmpb_gtl
5407     \_unravel_set_action_text:e
5408     {
5409       Conditional:~ \tl_to_str:N \l__unravel_head_tl
5410       \gtl_to_str:N \l__unravel_tmpb_gtl
5411     }
5412     \_unravel_prev_input:e
5413     {
5414       \gtl_if_tl:NTF \l__unravel_tmpb_gtl
5415       { \gtl_head:N \l__unravel_tmpb_gtl }
5416       { \gtl_to_str:N \l__unravel_tmpb_gtl }
5417     }
5418   }
5419 }
```

(End of definition for \\_unravel\_test\_ifdefined:.)

\\_unravel\_test\_ifcsname:

```
5420 \cs_new_protected:Npn \_unravel_test_ifcsname:
5421 {
5422   \_unravel_csname_loop:
5423   \_unravel_prev_input:V \l__unravel_head_tl
5424   \_unravel_get_lastnamedcs:
5425 }
```

(End of definition for \\_unravel\_test\_ifcsname:.)

```
5426 \_unravel_new_tex_expandable:nn { fi_or_else } % 108
5427 {
5428   \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
5429   {
5430     \int_compare:nNnTF \g__unravel_if_limit_int = 0
5431     {
5432       \int_compare:nNnTF \g__unravel_if_depth_int = 0
5433       { \_unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
5434       { \_unravel_insert_relax: }
5435     }
5436     { \_unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
5437   }
5438   {
5439     \_unravel_set_action_text:
```

```

5440     \int_compare:nNnF \l__unravel_head_char_int = 2
5441     {
5442         \__unravel_fi_or_else_loop:
5443         \__unravel_set_action_text:e
5444         {
5445             \g__unravel_action_text_str \c_space_tl
5446             => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
5447         }
5448     }
5449     \__unravel_print_expansion:
5450     \__unravel_cond_pop:
5451 }
5452 }
5453 \cs_new_protected:Npn \__unravel_fi_or_else_loop:
5454 {
5455     \int_compare:nNnF \l__unravel_head_char_int = 2
5456     {
5457         \__unravel_pass_text:
5458         \__unravel_set_cmd:
5459         \__unravel_fi_or_else_loop:
5460     }
5461 }

```

## 2.14 User interaction

### 2.14.1 Print

Let us start with the procedure which prints to the terminal: this will help me test the code while I'm writing it.

`\__unravel_print_normalize_null:` Change the null character to an explicit `^^@` in LuaTeX to avoid a bug whereby a null character ends a string prematurely.

`\l__unravel_print_tl`

```

5462 \tl_new:N \l__unravel_print_tl
5463 \sys_if_engine luatex:TF
5464 {
5465     \cs_new_protected:Npe \__unravel_print_normalize_null:
5466     {
5467         \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
5468         { \char_generate:mn { 0 } { 12 } }
5469         { \tl_to_str:n { ^^@ } }
5470     }
5471 }
5472 { \cs_new_protected:Npn \__unravel_print_normalize_null: { } }

```

*(End of definition for `\__unravel_print_normalize_null:` and `\l__unravel_print_tl`.)*

`\__unravel_print:n`

`\__unravel_print:e`

`\__unravel_log:n`

```

5473 \cs_new_protected:Npn \__unravel_print:n #1
5474 {
5475     \tl_set:Nn \l__unravel_print_tl {#1}
5476     \__unravel_print_normalize_null:
5477     \iow_term:e { \l__unravel_print_tl }
5478     \tl_if_empty:NF \g__unravel_output_file_tl
5479     { \iow_now:Ne \g__unravel_iow { \l__unravel_print_tl } }

```

```

5480 }
5481 \cs_generate_variant:Nn \__unravel_print:n { e }
5482 \cs_new_protected:Npn \__unravel_log:n #1
5483 {
5484   \tl_set:Nn \l__unravel_print_tl {#1}
5485   \__unravel_print_normalize_null:
5486   \tl_if_empty:NTF \g__unravel_output_file_tl
5487     { \iow_log:e { \l__unravel_print_tl } }
5488     { \iow_now:Ne \g__unravel_iow { \l__unravel_print_tl } }
5489 }

```

(End of definition for \\_\_unravel\_print:n and \\_\_unravel\_log:n.)

\\_\_unravel\_print\_step:n Steps are printed or not according to the option value, unaffected by a prompt such as s10.

```

5490 \cs_new_protected:Npn \__unravel_print_step:n #1
5491 {
5492   \int_compare:nNnF \g__unravel_online_int < 0
5493   {
5494     \int_compare:nNnTF \g__unravel_online_int = 0
5495       { \__unravel_log:n {#1} }
5496       { \__unravel_print:n {#1} }
5497   }
5498 }

```

(End of definition for \\_\_unravel\_print\_step:n.)

\\_\_unravel\_print\_message:nn The message to be printed should come already detokenized, as #2. It will be wrapped to 80 characters per line, with #1 before each line. The message is properly suppressed (or sent only to the log) according to \g\_\_unravel\_current\_online\_int so as to be sensitive to the prompt.

```

5499 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
5500 {
5501   \int_compare:nNnF \g__unravel_current_online_int < 0
5502   {
5503     \int_compare:nNnTF \g__unravel_current_online_int = 0
5504       { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_log:n }
5505       { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_print:n }
5506   }
5507 }

```

(End of definition for \\_\_unravel\_print\_message:nn.)

\\_\_unravel\_set\_action\_text:e

```

5508 \cs_new_protected:Npn \__unravel_set_action_text:e #1
5509 {
5510   \group_begin:
5511     \__unravel_set_escapechar:n { 92 }
5512     \str_gset:Ne \g__unravel_action_text_str {#1}
5513   \group_end:
5514 }

```

(End of definition for \\_\_unravel\_set\_action\_text:e.)

`\_unravel_set_action_text:`

```
5515 \cs_new_protected:Npn \_unravel_set_action_text:
5516 {
5517   \_unravel_set_action_text:e
5518   {
5519     \tl_to_str:N \l__unravel_head_tl
5520     \tl_if_single_token:VT \l__unravel_head_tl
5521     { = ~ \token_to_meaning:N \l__unravel_head_token }
5522   }
5523 }
```

*(End of definition for \\_unravel\_set\_action\_text:.)*

`\_unravel_print_state:`

```
5524 \cs_new_protected:Npn \_unravel_print_state:
5525 {
5526   \group_begin:
5527   \_unravel_set_escapechar:n { 92 }
5528   \tl_use:N \g__unravel_before_print_state_tl
5529   \int_compare:nNnF \g__unravel_current_online_int < 0
5530   {
5531     \_unravel_print_state_output:
5532     \_unravel_print_state_prev:
5533     \_unravel_print_state_input:
5534   }
5535   \group_end:
5536 }
```

*(End of definition for \\_unravel\_print\_state:.)*

`\_unravel_print_state_output:` Unless empty, print #1 with each line starting with <|~. The `\_unravel_str_truncate_left:nn` function trims #1 if needed, to fit in a maximum of `\g__unravel_max_output_int` characters.

`\_unravel_print_state_output:n`

```
5537 \cs_new_protected:Npn \_unravel_print_state_output:
5538 {
5539   \exp_args:Ne \_unravel_print_state_output:n
5540   { \gtl_to_str:N \g__unravel_output_gtl }
5541 }
5542 \cs_new_protected:Npn \_unravel_print_state_output:n #1
5543 {
5544   \tl_if_empty:nF {#1}
5545   {
5546     \_unravel_print_message:nn { <| ~ } % |
5547     { \_unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
5548   }
5549 }
```

*(End of definition for \\_unravel\_print\_state\_output: and \\_unravel\_print\_state\_output:n.)*

`\_unravel_print_state_prev:` Never trim ##1.

```
5550 \cs_new_protected:Npn \_unravel_print_state_prev:
5551 {
5552   \seq_set_map_e:Nn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
5553   { \_unravel_to_str:Nn ##1 }
```

```

5554 \seq_remove_all:Nn \l__unravel_tmpa_seq { }
5555 \seq_if_empty:NTF \l__unravel_tmpa_seq
5556 { \__unravel_print_message:nn { || ~ } { } }
5557 {
5558   \seq_map_inline:Nn \l__unravel_tmpa_seq
5559   {
5560     \__unravel_print_message:nn { || ~ } {##1}
5561   }
5562 }
5563 }

```

(End of definition for `\__unravel_print_state_prev:.`)

`\__unravel_print_state_input:` Print #1 with each line starting with `|>~.` The `\__unravel_str_truncate_right:nn`  
`\__unravel_print_state_input:n` function trims #1 if needed, to fit in a maximum of `\g__unravel_max_input_int` characters.

```

5564 \cs_new_protected:Npn \__unravel_print_state_input:
5565 {
5566   \exp_args:Ne \__unravel_print_state_input:n
5567   { \__unravel_input_to_str: }
5568 }
5569 \cs_new_protected:Npn \__unravel_print_state_input:n #1
5570 {
5571   \__unravel_print_message:nn { |> ~ } % |
5572   { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
5573 }

```

(End of definition for `\__unravel_print_state_input:` and `\__unravel_print_state_input:n`.)

`\__unravel_print_meaning:`

```

5574 \cs_new_protected:Npn \__unravel_print_meaning:
5575 {
5576   \__unravel_input_if_empty:TF
5577   { \__unravel_print_message:nn { } { Empty-input! } }
5578   {
5579     \__unravel_input_get:N \l__unravel_tmpb_gtl
5580     \__unravel_print_message:nn { }
5581     {
5582       \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
5583       = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
5584     }
5585   }
5586 }

```

(End of definition for `\__unravel_print_meaning:.`)

`\__unravel_print_action:` Some of these commands are currently synonyms but we may decide to make some options  
`\__unravel_print_action:e` act differently on them.

```

\__unravel_print_assignment: 5587 \cs_new_protected:Npn \__unravel_print_action:
\__unravel_print_assignment:e 5588 { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
\__unravel_print_expansion: 5589 \cs_new_protected:Npn \__unravel_print_action:e #1
\__unravel_print_expansion:e 5590 {
\__unravel_print_action_aux:N 5591   \__unravel_set_action_text:e {#1}
5592   \__unravel_print_action:

```

```

5593 }
5594 \cs_new_protected:Npn \__unravel_print_assignment:
5595 { \__unravel_print_action_aux:N \g__unravel_trace_assigns_bool }
5596 \cs_new_protected:Npn \__unravel_print_assignment:e #1
5597 {
5598   \__unravel_set_action_text:e {#1}
5599   \__unravel_print_assignment:
5600 }
5601 \cs_new_protected:Npn \__unravel_print_expansion:
5602 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
5603 \cs_new_protected:Npn \__unravel_print_expansion:e #1
5604 {
5605   \__unravel_set_action_text:e {#1}
5606   \__unravel_print_expansion:
5607 }
5608 \cs_new_protected:Npn \__unravel_print_action_aux:N #1
5609 {
5610   \int_gdecr:N \g__unravel_nonstop_int
5611   \int_gincr:N \g__unravel_step_int
5612   \bool_if:NT #1
5613   {
5614     \exp_args:Ne \__unravel_print_step:n
5615     {
5616       [====
5617       \bool_if:NT \g__unravel_number_steps_bool
5618         { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
5619       =====]~
5620       \int_compare:nNnTF
5621         { \str_count:N \g__unravel_action_text_str
5622         > { \g__unravel_max_action_int }
5623         {
5624           \str_range:Nnn \g__unravel_action_text_str
5625             { 1 } { \g__unravel_max_action_int - 3 } ...
5626         }
5627         { \g__unravel_action_text_str }
5628       }
5629     \__unravel_print_state:
5630     \__unravel_prompt:
5631   }
5632 }

```

(End of definition for \\_\_unravel\_print\_action: and others.)

```

\__unravel_just_print_assigned_token:
  \__unravel_print_assigned_token:
  \__unravel_print_assigned_register:
  \__unravel_print_assigned_parshape:
  \__unravel_print_assigned_set_shape:
  \__unravel_print_assigned_set_shape_aux:n
5633 \cs_new_protected:Npn \__unravel_just_print_assigned_token:
5634 {
5635   \__unravel_print_assignment:e
5636   {
5637     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5638     = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
5639   }
5640 }
5641 \cs_new_protected:Npn \__unravel_print_assigned_token:
5642 {

```

```

5643     \__unravel_after_assignment:
5644     \__unravel_just_print_assigned_token:
5645     \__unravel_omit_after_assignment:w
5646   }
5647 \cs_new:Npn \__unravel_print_assigned_aux_name:
5648   {
5649     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5650     \tl_if_single:NT \l__unravel_defined_tl
5651     { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
5652   }
5653 \cs_new_protected:Npn \__unravel_print_assigned_register:
5654   {
5655     \__unravel_after_assignment:
5656     \exp_args:Ne \__unravel_print_assignment:e % needed to stringify a \toks
5657     {
5658       \exp_not:N \__unravel_print_assigned_aux_name:
5659       = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
5660     }
5661     \__unravel_omit_after_assignment:w
5662   }
5663 \cs_new_protected:Npn \__unravel_print_assigned_parshape:
5664   {
5665     \__unravel_after_assignment:
5666     \tl_set:Nn \l__unravel_tmpa_tl { \tex_parshapedimen:D }
5667     \__unravel_print_assignment:e
5668     {
5669       \__unravel_print_assigned_aux_name: = \__unravel_the:w \l__unravel_defined_tl
5670       \int_step_function:nN { 2 * \l__unravel_defined_tl }
5671       \__unravel_print_assigned_set_shape_aux:n
5672     }
5673     \__unravel_omit_after_assignment:w
5674   }
5675 \cs_new_protected:Npn \__unravel_print_assigned_set_shape:
5676   {
5677     \__unravel_after_assignment:
5678     \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_defined_tl
5679     \__unravel_print_assignment:e
5680     {
5681       \__unravel_print_assigned_aux_name:
5682       = \__unravel_the:w \l__unravel_defined_tl 0 \exp_stop_f:
5683       \int_step_function:nN { \l__unravel_defined_tl 0 }
5684       \__unravel_print_assigned_set_shape_aux:n
5685     }
5686     \__unravel_omit_after_assignment:w
5687   }
5688 \cs_new:Npn \__unravel_print_assigned_set_shape_aux:n #1
5689   { ~ \__unravel_the:w \l__unravel_tmpa_tl #1 \exp_stop_f: }

```

(End of definition for \\_\_unravel\_just\_print\_assigned\_token: and others.)

\\_\_unravel\_print\_welcome: Welcome message.

```

5690 \cs_new_protected:Npn \__unravel_print_welcome:
5691   {
5692     \__unravel_print_message:nn { }

```

```

5693     {
5694         \bool_if:NTF \g__unravel_welcome_message_bool
5695         {
5696             \\\
5697             =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5698             \iow_indent:n
5699             {
5700                 "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\
5701                 "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\\
5702                 "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\\
5703                 Press~<enter>~to~continue;~'h'~<enter>~for~help. \\\
5704             }
5705         }
5706         { [=====Start=====] }
5707     }
5708     \__unravel_print_state:
5709     \__unravel_prompt:
5710 }

```

(End of definition for \\_\_unravel\_print\_welcome:.)

\\_\_unravel\_print\_outcome: Final message.

```

5711 \cs_new_protected:Npn \__unravel_print_outcome:
5712   { \__unravel_print_message:nn { } { [=====End=====] } }

```

(End of definition for \\_\_unravel\_print\_outcome:.)

## 2.14.2 Prompt

\\_\_unravel\_ior\_str\_get:NN  
\\_\_unravel\_ior\_str\_get:Nc

```

5713 \cs_new_protected:Npn \__unravel_ior_str_get:NN #1#2
5714   { \tex_readline:D #1 to #2 }
5715 \cs_generate_variant:Nn \__unravel_ior_str_get:NN { Nc }

```

(End of definition for \\_\_unravel\_ior\_str\_get:NN.)

\\_\_unravel\_prompt:

```

5716 \cs_new_protected:Npn \__unravel_prompt:
5717   {
5718     \int_compare:nNnF \g__unravel_nonstop_int > 0
5719     {
5720       \group_begin:
5721         \__unravel_set_escapechar:n { -1 }
5722         \int_set:Nn \tex_endlinechar:D { -1 }
5723         \tl_use:N \g__unravel_before_prompt_tl
5724         \__unravel_prompt_aux:
5725       \group_end:
5726     }
5727   }
5728 \cs_new_protected:Npn \__unravel_prompt_aux:
5729   {
5730     \clist_if_empty:NTF \g__unravel_prompt_input_clist
5731     {
5732       \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
5733       {

```



```

5734         \bool_if:NTF \g__unravel_explicit_prompt_bool
5735         { \__unravel_ior_str_get:Nc \c__unravel_prompt_ior }
5736         { \__unravel_ior_str_get:Nc \c__unravel_noprompt_ior }
5737         { Your~input }
5738     \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5739 }
5740 }
5741 {
5742     \clist_gpop:NN \g__unravel_prompt_input_clist \l__unravel_tmpa_tl
5743     \group_begin:
5744         \__unravel_set_escapechar:n { 92 }
5745         \__unravel_print:e
5746         {
5747             \bool_if:NT \g__unravel_explicit_prompt_bool { Your~input= }
5748             \tl_to_str:N \l__unravel_tmpa_tl
5749         }
5750     \group_end:
5751     \exp_args:NV \__unravel_prompt_treat:n \l__unravel_tmpa_tl
5752 }
5753 }
5754 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5755 {
5756     \tl_if_empty:nF {#1}
5757     {
5758         \str_case:enF { \tl_head:n {#1} }
5759         {
5760             { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
5761             { q }
5762             {
5763                 \int_gset:Nn \g__unravel_current_online_int { -1 }
5764                 \int_gzero:N \g__unravel_nonstop_int
5765             }
5766             { x }
5767             {
5768                 \group_end:
5769                 \__unravel_exit_hard:w
5770             }
5771             { X }
5772             {
5773                 \tex_batchmode:D
5774                 \tex_read:D -1 to \l__unravel_tmpa_tl
5775             }
5776             { s } { \__unravel_prompt_scan_int:nn {#1}
5777                 \__unravel_prompt_silent_steps:n }
5778             { o } { \__unravel_prompt_scan_int:nn {#1}
5779                 { \int_gset:Nn \g__unravel_current_online_int } }
5780             { C }
5781             {
5782                 \use:e
5783                 {
5784                     \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5785                     { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5786                 }
5787                 \tl_gput_left:Nn \g__unravel_tmpc_tl

```

```

5788         { \tl_gclear:N \g__unravel_tmpc_tl }
5789         \group_insert_after:N \g__unravel_tmpc_tl
5790         \group_insert_after:N \__unravel_prompt:
5791     }
5792     { | } % |
5793     { \__unravel_prompt_scan_int:nn {#1}
5794       \__unravel_prompt_vert:n }
5795     { u } { \__unravel_prompt_until:n {#1} }
5796     { a } { \__unravel_prompt_all: }
5797   }
5798   { \__unravel_prompt_help: }
5799 }
5800 }
5801 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
5802 {
5803   \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
5804   \l__unravel_prompt_tmpa_int =
5805     \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5806     \use_ii:nn #1 \scan_stop:
5807 }
5808 \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5809 {
5810   #2 \l__unravel_prompt_tmpa_int
5811   \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
5812 }
5813 \cs_new_protected:Npn \__unravel_prompt_help:
5814 {
5815   \__unravel_print:n { "m":~meaning-of~first~token }
5816   \__unravel_print:n { "a":~print~state~again,~without~truncating }
5817   \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
5818   \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
5819   \__unravel_print:n { "u<text>":~silent~steps~until~the~input~starts~with~<text> }
5820   \__unravel_print:n
5821     { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
5822   \__unravel_print:n { "q":~semi~quiet~(same~as~"o-1") }
5823   \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
5824   \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
5825   \__unravel_prompt_aux:
5826 }
5827 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
5828 {
5829   \int_compare:nNnF {#1} < 0
5830   {
5831     \int_gset:Nn \g__unravel_current_online_int { -1 }
5832     \tl_gset:Nn \g__unravel_before_prompt_tl
5833     {
5834       \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
5835       \tl_gclear:N \g__unravel_before_prompt_tl
5836     }
5837     \int_gset:Nn \g__unravel_nonstop_int {#1}
5838   }
5839 }
5840 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5841 {

```

```

5842     \int_compare:nNnTF {#1} < { 0 }
5843     { \__unravel_prompt_vert:Nn > {#1} }
5844     { \__unravel_prompt_vert:Nn < {#1} }
5845 }
5846 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5847 {
5848     \int_gset:Nn \g__unravel_current_online_int { -1 }
5849     \tl_gset:Nf \g__unravel_before_print_state_tl
5850     {
5851         \exp_args:Nnf \exp_stop_f: \int_compare:nNnTF
5852         { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5853         #1 { \__unravel_prev_input_count: }
5854         {
5855             \int_gset:Nn \g__unravel_nonstop_int
5856             { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5857         }
5858         {
5859             \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
5860             \tl_gclear:N \g__unravel_before_print_state_tl
5861         }
5862     }
5863 }
5864 \cs_new_protected:Npn \__unravel_prompt_all:
5865 {
5866     \tl_gset:Ne \g__unravel_tmpc_tl
5867     {
5868         \exp_not:n
5869         {
5870             \tl_gclear:N \g__unravel_tmpc_tl
5871             \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5872             \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5873             \__unravel_print_state:
5874             \int_gdecr:N \g__unravel_nonstop_int
5875             \__unravel_prompt:
5876         }
5877         \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5878         \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5879     }
5880     \group_insert_after:N \g__unravel_tmpc_tl
5881 }
5882 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5883 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End of definition for \\_\_unravel\_prompt:.)

```

\__unravel_prompt_until:n
  \g__unravel_until_tl
5884 \tl_new:N \g__unravel_until_tl
5885 \cs_new_protected:Npn \__unravel_prompt_until:n #1
5886 {
5887     \tl_gset:Ne \g__unravel_until_tl { \tl_tail:n {#1} }
5888     \int_gset:Nn \g__unravel_current_online_int { -1 }
5889     \tl_gset:Nn \g__unravel_before_print_state_tl
5890     {
5891         \__unravel_input_get_left:N \l__unravel_tmpa_tl

```

```

5892     \use:e
5893     {
5894         \exp_not:N \tl_if_in:nnTF
5895         { \exp_not:N \__unravel:nn \tl_to_str:N \l__unravel_tmpa_tl }
5896         { \exp_not:N \__unravel:nn \tl_to_str:N \g__unravel_until_tl }
5897     }
5898     {
5899         \int_gzero:N \g__unravel_nonstop_int
5900         \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
5901         \tl_gclear:N \g__unravel_before_print_state_tl
5902     }
5903     {
5904         \int_gset:Nn \g__unravel_nonstop_int
5905         { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5906     }
5907 }
5908 }

```

(End of definition for \\_\_unravel\_prompt\_until:n and \g\_\_unravel\_until\_tl.)

### 2.14.3 Errors

\\_\_unravel\_not\_implemented:n

```

5909 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5910 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End of definition for \\_\_unravel\_not\_implemented:n.)

\\_\_unravel\_error:nnnnn  
\\_\_unravel\_error:neeee

Errors within a group to make sure that none of the `l3msg` variables (or others) that may be currently in use in the code being debugged are modified.

```

5911 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5912 {
5913     \group_begin:
5914     \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5915     \group_end:
5916 }
5917 \cs_new_protected:Npn \__unravel_error:neeee #1#2#3#4#5
5918 {
5919     \group_begin:
5920     \msg_error:nneeee { unravel } {#1} {#2} {#3} {#4} {#5}
5921     \group_end:
5922 }

```

(End of definition for \\_\_unravel\_error:nnnnn.)

\\_\_unravel\_tex\_msg\_new:nnn

This stores a  $\TeX$  error message.

```

5923 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5924 {
5925     \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
5926     \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
5927 }

```

(End of definition for \\_\_unravel\_tex\_msg\_new:nnn.)

`\_unravel_tex_error:nn` Throw the `tex-error` message, with arguments: `#2` which triggered the error, `TEX`'s error message, and `TEX`'s help text.

```

5928 \cs_new_protected:Npn \_unravel_tex_error:nn #1#2
5929 {
5930   \_unravel_error:neeee { tex-error }
5931   { \tl_to_str:n {#2} }
5932   { \use:c { \_unravel_tex_msg_error_#1: } }
5933   { \use:c { \_unravel_tex_msg_help_#1: } }
5934   { }
5935 }
5936 \cs_generate_variant:Nn \_unravel_tex_error:nn { nV }

```

*(End of definition for `\_unravel_tex_error:nn`.)*

`\_unravel_tex_fatal_error:nn` Throw the `tex-fatal` error message, with arguments: `#2` which triggered the fatal error, `TEX`'s error message, and `TEX`'s help text.

```

5937 \cs_new_protected:Npn \_unravel_tex_fatal_error:nn #1#2
5938 {
5939   \_unravel_error:neeee { tex-fatal }
5940   { \tl_to_str:n {#2} }
5941   { \use:c { \_unravel_tex_msg_error_#1: } }
5942   { \use:c { \_unravel_tex_msg_help_#1: } }
5943   { }
5944 }
5945 \cs_generate_variant:Nn \_unravel_tex_fatal_error:nn { nV }

```

*(End of definition for `\_unravel_tex_fatal_error:nn`.)*

## 2.15 Keys

Each key needs to be defined twice: for its default setting and for its setting applying to a single `\unravel`. This is due to the fact that we cannot use grouping to keep settings local to a single `\unravel` since the `<code>` argument of `\unravel` may open or close groups.

```

5946 \keys_define:nn { unravel/defaults }
5947 {
5948   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5949   internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5950   max-action .int_gset:N = \g__unravel_default_max_action_int ,
5951   max-output .int_gset:N = \g__unravel_default_max_output_int ,
5952   max-input .int_gset:N = \g__unravel_default_max_input_int ,
5953   number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
5954   online .int_gset:N = \g__unravel_default_online_int ,
5955   output-file .code:n = {
5956     \int_gzero:N \g__unravel_default_online_int
5957     \tl_gset:Nn \g__unravel_default_output_file_tl {#1}
5958   } ,
5959   prompt-input .code:n
5960     = \_unravel_prompt_input:Nn \g__unravel_default_prompt_input_clist {#1} ,
5961   trace-assigns .bool_gset:N = \g__unravel_default_trace_assigns_bool ,
5962   trace-expansion .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
5963   trace-other .bool_gset:N = \g__unravel_default_trace_other_bool ,
5964   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,

```

```

5965 }
5966 \keys_define:nn { unravel }
5967 {
5968   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5969   internal-debug .bool_gset:N = \g__unravel_internal_debug_bool ,
5970   max-action .int_gset:N = \g__unravel_max_action_int ,
5971   max-output .int_gset:N = \g__unravel_max_output_int ,
5972   max-input .int_gset:N = \g__unravel_max_input_int ,
5973   number-steps .bool_gset:N = \g__unravel_number_steps_bool ,
5974   online .int_gset:N = \g__unravel_online_int ,
5975   output-file .code:n = {
5976     \int_gzero:N \g__unravel_online_int
5977     \tl_gset:Nn \g__unravel_output_file_tl {#1}
5978   } ,
5979   prompt-input .code:n
5980     = \__unravel_prompt_input:Nn \g__unravel_prompt_input_clist {#1} ,
5981   trace-assigns .bool_gset:N = \g__unravel_trace_assigns_bool ,
5982   trace-expansion .bool_gset:N = \g__unravel_trace_expansion_bool ,
5983   trace-other .bool_gset:N = \g__unravel_trace_other_bool ,
5984   welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,
5985 }

```

The machine and trace options are somewhat special so it is clearer to define them separately. The code is identical for unravel/defaults and unravel keys. To be sure of which options are set, use .meta:nn and give the path explicitly.

```

5986 \tl_map_inline:nn { { /defaults } { } }
5987 {
5988   \keys_define:nn { unravel #1 }
5989   {
5990     machine .meta:nn =
5991       { unravel #1 }
5992       {
5993         explicit-prompt = false ,
5994         internal-debug = false ,
5995         max-action = \c_max_int ,
5996         max-output = \c_max_int ,
5997         max-input = \c_max_int ,
5998         number-steps = false ,
5999         welcome-message = false ,
6000       } ,
6001     mute .meta:nn =
6002       { unravel #1 }
6003       {
6004         trace-assigns = false ,
6005         trace-expansion = false ,
6006         trace-other = false ,
6007         welcome-message = false ,
6008         online = -1 ,
6009       }
6010   }
6011 }

```

## 2.16 Main command

**\unravel** Simply call an underlying code-level command.

```
6012 \NewDocumentCommand \unravel { 0 { } +m } { \unravel:nn {#1} {#2} }
```

*(End of definition for \unravel. This function is documented on page 2.)*

**\unravelsetup** Simply call an underlying code-level command.

```
6013 \NewDocumentCommand \unravelsetup { m } { \unravel_setup:n {#1} }
```

*(End of definition for \unravelsetup. This function is documented on page 2.)*

**\unravel\_setup:n** Set keys, updating both default values and current values.

```
6014 \cs_new_protected:Npn \unravel_setup:n #1
6015 {
6016   \keys_set:nn { unravel/defaults } {#1}
6017   \keys_set:nn { unravel } {#1}
6018 }
```

*(End of definition for \unravel\_setup:n. This function is documented on page 3.)*

**\unravel:nn** The command starts with `\__unravel_unravel_marker:` to detect nesting of `\unravel` in `\unravel` and avoid re-initializing important variables. Initialize and setup keys.

`\__unravel:nn`

`\__unravel_unravel_marker:`

Initialize and setup other variables including the input. Welcome the user. Then comes the main loop: until the input is exhausted, print the current status and do one step. The main loop is exited by skipping to the first `\__unravel_exit_point:`, while some abort procedures jump to the second (and last) one instead. If the main loop finished correctly, print its outcome and finally test that everything is all right.

```
6019 \cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
6020 \cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
6021 \cs_new_protected:Npn \__unravel:nn #1#2
6022 {
6023   \__unravel_init_key_vars:
6024   \keys_set:nn { unravel } {#1}
6025   \__unravel_init_vars:
6026   \__unravel_input_gset:n {#2}
6027   \__unravel_print_welcome:
6028   \__unravel_main_loop:
6029   \__unravel_exit_point:
6030   \__unravel_print_outcome:
6031   \__unravel_final_test:
6032   \__unravel_exit_point:
6033 }
6034 \cs_new_protected:Npn \unravel_get:nnN #1#2#3
6035 {
6036   \unravel:nn {#1} {#2}
6037   \tl_set:Nc #3 { \gtl_left_tl:N \g__unravel_output_gtl }
6038 }
```

*(End of definition for \unravel:nn, \\_\_unravel:nn, and \\_\_unravel\_unravel\_marker:. This function is documented on page 2.)*

`\__unravel_init_key_vars:` Give variables that are affected by keys their default values (also controlled by keys).

```
6039 \cs_new_protected:Npn \__unravel_init_key_vars:
6040 {
6041   \sys_if_engine luatex:T { \tl_gset:No \g__unravel_lastnamedcs_tl { \tex_lastnamedcs:D }
6042   \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bo
6043   \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
6044   \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
6045   \int_gset_eq:NN \g__unravel_online_int \g__unravel_default_online_int
6046   \tl_gset_eq:NN \g__unravel_output_file_tl \g__unravel_default_output_file_tl
6047   \clist_gset_eq:NN \g__unravel_prompt_input_clist \g__unravel_default_prompt_input_clist
6048   \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
6049   \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
6050   \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
6051   \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
6052   \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
6053   \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
6054   \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
6055   \int_gzero:N \g__unravel_nonstop_int
6056   \tl_gclear:N \g__unravel_before_print_state_tl
6057   \tl_gclear:N \g__unravel_before_prompt_tl
6058 }
```

*(End of definition for \\_\_unravel\_init\_key\_vars:.)*

`\__unravel_init_vars:` Give initial values to variables used during the processing. These have no reason to be modified by the user: neither directly nor through keys.

```
6059 \cs_new_protected:Npn \__unravel_init_vars:
6060 {
6061   \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
6062   \tl_if_eq:NMF \g__unravel_output_file_tl \g__unravel_current_output_file_tl
6063   {
6064     \iow_close:N \g__unravel_iow
6065     \iow_open:Nn \g__unravel_iow \g__unravel_output_file_tl
6066     \tl_gset_eq:NN \g__unravel_current_output_file_tl \g__unravel_output_file_tl
6067   }
6068   \seq_gclear:N \g__unravel_prev_input_seq
6069   \gtl_gclear:N \g__unravel_output_gtl
6070   \int_gzero:N \g__unravel_step_int
6071   \tl_gclear:N \g__unravel_if_limit_tl
6072   \int_gzero:N \g__unravel_if_limit_int
6073   \int_gzero:N \g__unravel_if_depth_int
6074   \gtl_gclear:N \g__unravel_after_assignment_gtl
6075   \bool_gset_true:N \g__unravel_set_box_allowed_bool
6076   \bool_gset_false:N \g__unravel_name_in_progress_bool
6077   \gtl_clear:N \l__unravel_after_group_gtl
6078 }
```

*(End of definition for \\_\_unravel\_init\_vars:.)*

`\__unravel_main_loop:` Loop forever, getting the next token (with expansion) and performing the corresponding command. We use `\__unravel_get_x_next_or_done:`, which is basically `\__unravel_get_x_next:` but with a different behaviour when there are no more tokens: running out of tokens here is a successful exit of `\unravel`. Note that we cannot put the logic into `\__unravel_main_loop:` because `\__unravel_expand_do:N` suppresses the loop when a



token is marked with `\notexpanded:`, and we don't want that to suppress the main loop, only the expansion loop.

```

6079 \cs_new_protected:Npn \__unravel_get_x_next_or_done:
6080 {
6081   \__unravel_input_if_empty:TF { \__unravel_exit:w } { }
6082   \__unravel_get_next:
6083   \__unravel_token_if_expandable:NT \l__unravel_head_token
6084   { \__unravel_expand_do:N \__unravel_get_x_next_or_done: }
6085 }
6086 \cs_new_protected:Npn \__unravel_main_loop:
6087 {
6088   \__unravel_get_x_next_or_done:
6089   \__unravel_set_cmd:
6090   \__unravel_do_step:
6091   \__unravel_main_loop:
6092 }

```

*(End of definition for \\_\_unravel\_main\_loop: and \\_\_unravel\_get\_x\_next\_or\_done:.)*

`\__unravel_do_step:` Perform the action if the corresponding command exists. If that command does not exist, complain, and leave the token in the output.

```

6093 \cs_new_protected:Npn \__unravel_do_step:
6094 {
6095   \__unravel_set_action_text:
6096   \bool_if:NT \g__unravel_internal_debug_bool
6097   { \iow_term:e { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int } } }
6098   \cs_if_exist_use:cF
6099   { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
6100   { \__unravel_error:neeee { internal } { unknown-command } { } { } { } }
6101 }

```

*(End of definition for \\_\_unravel\_do\_step:.)*

`\__unravel_final_test:` Make sure that the `\unravel` finished correctly. The error message is a bit primitive.

```

\__unravel_final_bad:
6102 \cs_new_protected:Npn \__unravel_final_test:
6103 {
6104   \__unravel_input_if_empty:TF
6105   {
6106     \seq_if_empty:NTF \g__unravel_prev_input_seq
6107     {
6108       \tl_if_empty:NTF \g__unravel_if_limit_tl
6109       { \int_compare:nNnF \g__unravel_if_limit_int = 0 { \__unravel_final_bad: } }
6110       { \__unravel_final_conditionals: }
6111     }
6112     { \__unravel_final_bad: }
6113   }
6114   { \__unravel_final_bad: }
6115   \__unravel_final_after_assignment:
6116 }
6117 \cs_new_protected:Npn \__unravel_final_bad:
6118 {
6119   \__unravel_error:nnnnn { internal }
6120   { the-last-unravel-finished-badly } { } { } { }
6121 }

```

```

6122 \cs_new_protected:Npn \__unravel_final_conditionals:
6123 {
6124   \group_begin:
6125   \msg_warning:nne { unravel } { dangling-conditionals }
6126   { \tl_count:N \g__unravel_if_limit_tl }
6127   \group_end:
6128   \tl_greverse:N \g__unravel_if_limit_tl
6129   \tl_gput_right:NV \g__unravel_if_limit_tl \g__unravel_if_limit_int
6130   \tl_gset:Ne \g__unravel_if_limit_tl { \tl_tail:N \g__unravel_if_limit_tl } % remove the
6131   \prg_replicate:nn { \tl_count:N \g__unravel_if_limit_tl } { \fi: }
6132   \tl_map_function:NN \g__unravel_if_limit_tl \__unravel_final_cond_aux:n
6133 }
6134 \cs_new:Npn \__unravel_final_cond_aux:n #1
6135 {
6136   \int_case:nnF {#1}
6137   {
6138     { 2 } { \if_false: \else: }
6139     { 3 } { \if_true: }
6140     { 4 } { \if_case:w 0 ~ }
6141   }
6142   { \__unravel_final_bad: }
6143 }

```

(End of definition for \\_\_unravel\_final\_test: and \\_\_unravel\_final\_bad:.)

\\_\_unravel\_final\_after\_assignment: Salvage any remaining \afterassignment token.

```

6144 \cs_new_protected:Npn \__unravel_final_after_assignment:
6145 {
6146   \gtl_if_empty:NF \g__unravel_after_assignment_gtl
6147   { \gtl_head_do:NN \g__unravel_after_assignment_gtl \tex_afterassignment:D }
6148 }

```

(End of definition for \\_\_unravel\_final\_after\_assignment:.)

## 2.17 Messages

```

6149 \msg_new:nnnn { unravel } { prev-input }
6150 { Internal~error:~unexpected~type~of~‘prev_input’~entry. }
6151 {
6152   Found~type~#2~instead~of~#1~to~assign~to~variable~#3.~Contents:\\
6153   \iow_indent:n {#4}
6154 }
6155 \msg_new:nnn { unravel } { unknown-primitive }
6156 { Internal~error:~the~primitive~‘#1’~is~not~known. }
6157 \msg_new:nnn { unravel } { extra-fi-or-else }
6158 { Extra~fi,~or,~or~else. }
6159 \msg_new:nnn { unravel } { missing-dollar }
6160 { Missing~dollar~inserted. }
6161 \msg_new:nnn { unravel } { unknown-expandable }
6162 { Internal~error:~the~expandable~command~‘#1’~is~not~known. }
6163 \msg_new:nnn { unravel } { missing-font-id }
6164 { Missing~font~identifier.~\iow_char:N\{nullfont~inserted. }
6165 \msg_new:nnn { unravel } { missing-rparen }
6166 { Missing~right~parenthesis~inserted~for~expression. }
6167 \msg_new:nnn { unravel } { missing-cs }

```

```

6168 { Missing~control~sequence.~\iow_char:N\inaccessible~inserted. }
6169 \msg_new:nnn { unravel } { missing-box }
6170 { Missing~box~inserted. }
6171 \msg_new:nnn { unravel } { missing-to }
6172 { Missing~keyword~'to'~inserted. }
6173 \msg_new:nnn { unravel } { improper-leaders }
6174 { Leaders~not~followed~by~proper~glue. }
6175 \msg_new:nnn { unravel } { extra-close }
6176 { Extra~right~brace~or~\iow_char:N\endgroup. }
6177 \msg_new:nnn { unravel } { off-save }
6178 { Something~is~wrong~with~groups. }
6179 \msg_new:nnn { unravel } { hrule-bad-mode }
6180 { \iow_char\hrule-used-in-wrong-mode. }
6181 \msg_new:nnn { unravel } { invalid-mode }
6182 { Invalid~mode~for~this~command. }
6183 \msg_new:nnn { unravel } { color-stack-action-missing }
6184 { Missing~color~stack~action. }
6185 \msg_new:nnn { unravel } { action-type-missing }
6186 { Missing~action~type. }
6187 \msg_new:nnn { unravel } { identifier-type-missing }
6188 { Missing~identifier~type. }
6189 \msg_new:nnn { unravel } { destination-type-missing }
6190 { Missing~destination~type. }
6191 \msg_new:nnn { unravel } { erroneous-prefixes }
6192 { Prefixes~applied~to~non~assignment~command. }
6193 \msg_new:nnn { unravel } { improper-setbox }
6194 { \iow_char:N\setbox~while~fetching~base~of~an~accent. }
6195 \msg_new:nnn { unravel } { after-advance }
6196 {
6197   Missing~register~after~\iow_char:N\advance,~
6198   \iow_char:N\multiply,~or~\iow_char:N\divide.
6199 }
6200 \msg_new:nnn { unravel } { bad-unless }
6201 { \iow_char:N\unless~not~followed~by~conditional. }
6202 \msg_new:nnn { unravel } { runaway-if }
6203 { Runaway~\iow_char:N\if...~Exiting~\iow_char:N\unravel }
6204 \msg_new:nnn { unravel } { runaway-macro-parameter }
6205 {
6206   Runaway~macro~parameter~\# #2~after \\\
6207   \iow_indent:n {#1}
6208 }
6209 \msg_new:nnn { unravel } { runaway-text }
6210 { Runaway~braced~argument~for~TeX~primitive.~Exiting~\iow_char:N\unravel }
6211 \msg_new:nnn { unravel } { extra-or }
6212 { Extra~\iow_char:N\or. }
6213 \msg_new:nnn { unravel } { missing-equals }
6214 { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
6215 \msg_new:nnn { unravel } { internal }
6216 { Internal~error:~'#1'.~\ Please~report. }
6217 \msg_new:nnn { unravel } { not-implemented }
6218 { The~following~feature~is~not~implemented:~'#1'. }
6219 \msg_new:nnn { unravel } { endinput-ignored }
6220 { The~primitive~\iow_char:N\endinput~was~ignored. }
6221 \msg_new:nnn { unravel } { missing-something }

```

```

6222 { Something-is-missing,~sorry! }
6223 \msg_new:nnn { unravel } { nested-unravel }
6224 { The~\iow_char:N\unravel-command-may-not-be-nested. }
6225 \msg_new:nnnn { unravel } { tex-error }
6226 { TeX~sees~"#1"~and~throws~an~error:\\\\ \iow_indent:n {#2} }
6227 {
6228   \tl_if_empty:nTF {#3}
6229     { TeX~provides~no~further~help~for~this~error. }
6230     { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
6231 }
6232 \msg_new:nnnn { unravel } { tex-fatal }
6233 { TeX~sees~"#1"~and~throws~a~fatal~error:\\\\ \iow_indent:n {#2} }
6234 {
6235   \tl_if_empty:nTF {#3}
6236     { TeX~provides~no~further~help~for~this~error. }
6237     { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
6238 }
6239 \msg_new:nnnn { unravel } { runaway-unravel }
6240 { Runaway~\iow_char:N\unravel,~so~\iow_char:N\relax~inserted. }
6241 {
6242   Some~TeX~command~expects~input~beyond~the~end~of~
6243   the~argument~of~\iow_char:N\unravel.
6244 }
6245 \msg_new:nnn { unravel } { dangling-conditionals }
6246 { Attempting~to~issue~#1~dangling~conditionals. }

```

Some error messages from T<sub>E</sub>X itself.

```

6247 \__unravel_tex_msg_new:nnn { forbidden-case }
6248 {
6249   You~can't~use~'\exp_after:wN \token_to_str:N \l__unravel_head_tl'~in-
6250   \mode_if_vertical:TF { vertical }
6251   {
6252     \mode_if_horizontal:TF { horizontal }
6253     { \mode_if_math:TF { math } { no } }
6254   } ~ mode.
6255 }
6256 {
6257   Sorry,~but~I'm~not~programmed~to~handle~this~case;~
6258   I'll~just~pretend~that~you~didn't~ask~for~it.~
6259   If~you're~in~the~wrong~mode,~you~might~be~able~to~
6260   return~to~the~right~one~by~typing~'\iow_char:N\}'~or~
6261   '\iow_char:N\$'~or~'\iow_char:N\par'.
6262 }
6263 \__unravel_tex_msg_new:nnn { incompatible-mag }
6264 {
6265   Incompatible~magnification~
6266   ( \int_to_arabic:n { \__unravel_mag: } );~
6267   the~previous~value~will~be~retained
6268 }
6269 {
6270   I~can~handle~only~one~magnification~ratio~per~job.~So~I've~
6271   reverted~to~the~magnification~you~used~earlier~on~this~run.
6272 }
6273 \__unravel_tex_msg_new:nnn { illegal-mag }
6274 {

```

```

6275 Illegal~magnification~has~been~changed~to~1000~
6276 ( \int_to_arabic:n { \_unravel_mag: } )
6277 }
6278 { The~magnification~ratio~must~be~between~1~and~32768. }
6279 \_unravel_tex_msg_new:nnn { missing-number }
6280 { Missing~number,~treated~as~zero }
6281 {
6282 A~number~should~have~been~here;~I~inserted~'0'.~
6283 If~you~can't~figure~out~why~I~needed~to~see~a~number,~
6284 look~up~'weird~error'~in~the~index~to~The~TeXbook.
6285 }
6286 \_unravel_tex_msg_new:nnn { the-cannot }
6287 { You~can't~use~'\tl_to_str:N\l\_unravel_head_tl'~after~\iow_char:N\the }
6288 { I'm~forgetting~what~you~said~and~using~zero~instead. }
6289 \_unravel_tex_msg_new:nnn { incompatible-units }
6290 { Incompatible~glue~units }
6291 { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }
6292 \_unravel_tex_msg_new:nnn { missing-mu }
6293 { Illegal~unit~of~measure~(mu~inserted) }
6294 {
6295 The~unit~of~measurement~in~math~glue~must~be~mu.~
6296 To~recover~gracefully~from~this~error,~it's~best~to~
6297 delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
6298 two~letters.~(See~Chapter~27~of~The~TeXbook.)
6299 }
6300 \_unravel_tex_msg_new:nnn { missing-pt }
6301 { Illegal~unit~of~measure~(pt~inserted) }
6302 {
6303 Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
6304 cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
6305 I'll~assume~that~you~meant~to~say~pt,~for~printer's~points.~
6306 To~recover~gracefully~from~this~error,~it's~best~to~
6307 delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
6308 two~letters.~(See~Chapter~27~of~The~TeXbook.)
6309 }
6310 \_unravel_tex_msg_new:nnn { missing-lbrace }
6311 { Missing~\iow_char:N\{~inserted }
6312 {
6313 A~left~brace~was~mandatory~here,~so~I've~put~one~in.~
6314 You~might~want~to~delete~and/or~insert~some~corrections~
6315 so~that~I~will~find~a~matching~right~brace~soon.~
6316 (If~you're~confused~by~all~this,~try~typing~'\iow_char:N\}'~now.)
6317 }
6318 \_unravel_tex_msg_new:nnn { extra-endcsname }
6319 { Extra~\token_to_str:c{endcsname} }
6320 { I'm~ignoring~this,~since~I~wasn't~doing~a~\token_to_str:c{csname}. }
6321 \_unravel_tex_msg_new:nnn { missing-endcsname }
6322 { Missing~\token_to_str:c{endcsname}~inserted }
6323 {
6324 The~control~sequence~marked~<to~be~read~again>~should~
6325 not~appear~between~\token_to_str:c{csname}~and~
6326 \token_to_str:c{endcsname}.
6327 }
6328 \_unravel_tex_msg_new:nnn { missing-delim }

```

```

6329 { Missing-delimiter~(.~inserted) }
6330 {
6331   I~was~expecting~to~see~something~like~'('~or~'\token_to_str:N\{'~or~
6332   '\token_to_str:N\}'~here.~If~you~typed,~e.g.,~
6333   '\{'~instead~of~'\token_to_str:N\{'~you~
6334   should~probably~delete~the~'\{'~by~typing~'1'~now,~so~that~
6335   braces~don't~get~unbalanced.~Otherwise~just~proceed.~
6336   Acceptable~delimiters~are~characters~whose~\token_to_str:c{delcode}~is~
6337   nonnegative,~or~you~can~use~'\token_to_str:c{delimiter}~<delimiter~code>'.
6338 }

```

Fatal  $\TeX$  error messages.

```

6339 \__unravel_tex_msg_new:nnn { cannot-read }
6340 { ***~(cannot~\iow_char:N\read-from-terminal-in-nonstop-modes) }
6341 { }
6342 \__unravel_tex_msg_new:nnn { file-error }
6343 { ***~(job-aborted,~file-error-in-nonstop-mode) }
6344 { }
6345 \__unravel_tex_msg_new:nnn { interwoven-preambles }
6346 { (interwoven~alignment~preambles~are~not~allowed) }
6347 { }

```

Restore catcodes to their original values.

```

6348 \__unravel_setup_restore:
6349 </package>

```