

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2018/12/28

Contents

1	<code>unravel</code> documentation	2
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
2	<code>unravel</code> implementation	6
2.1	Primitives, variants, and helpers	10
2.1.1	Renamed primitives	10
2.1.2	Variants	10
2.1.3	Miscellaneous helpers	11
2.1.4	String helpers	13
2.1.5	Helpers for control flow	14
2.1.6	Helpers concerning tokens	15
2.1.7	Helpers for previous input	17
2.2	Variables	19
2.2.1	User interaction	19
2.2.2	Working with tokens	21
2.2.3	Numbers and conditionals	23
2.2.4	Boxes and groups	23
2.2.5	Constants	24
2.2.6	TeX parameters	24
2.3	Numeric codes	24
2.4	Get next token	38
2.5	Manipulating the input	44
2.5.1	Elementary operations	44
2.5.2	Insert token for error recovery	49
2.5.3	Macro calls	49
2.6	Expand next token	51
2.7	Basic scanning subroutines	53
2.8	Working with boxes	71

*This file has version number 0.2f, last revised 2018/12/28.

2.9	Paragraphs	75
2.10	Groups	77
2.11	Modes	79
2.12	One step	81
2.13	Commands	81
2.13.1	Characters: from 0 to 15	81
2.13.2	Boxes: from 16 to 31	85
2.13.3	From 32 to 47	90
2.13.4	Maths: from 48 to 56	94
2.13.5	From 57 to 70	95
2.13.6	Extensions	97
2.13.7	Assignments	104
2.14	Expandable primitives	114
2.14.1	Conditionals	121
2.15	User interaction	129
2.15.1	Print	129
2.15.2	Prompt	134
2.15.3	Errors	137
2.16	Keys	138
2.17	Main command	139
2.18	Messages	142

1 unravel documentation

The aim of this L^AT_EX package is to help debug complicated macros. This is done by letting the user step through the execution of some T_EX 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_EX in a terminal.

1.1 Commands

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

This command shows in the terminal the steps performed by T_EX 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 \unravelsetup {*<options>*}

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

\unravel:nn \unravel:nn {*<options>*} {*<code>*}

See \unravel.

<hr/> <hr/> <code>\unravel_get:nnN</code>	<code>\unravel_get:nnN {<options>} {<code>} <tl var></code>
	Performs <code>\unravel:nn</code> with the <code><options></code> and <code><code></code> then saves the output into the <code><tl var></code> . The option <code>mute</code> is useful in this case.
<hr/> <hr/> <code>\unravel_setup:n</code>	<code>\unravel_setup:n {<options>}</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.
<u>online</u>	Integer option 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.

<code>trace-assigns</code>	Boolean options (default <code>true</code>) controlling what steps produce any output at all. The keys <code>trace-assigns</code> , <code>trace-expansion</code> , <code>trace-other</code> control tracing of different types of steps.
<code>trace-expansion</code>	
<code>trace-other</code>	
<code>welcome-message</code>	Boolean option (default <code>true</code>) 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`, `\crrcr`, `&`), many math mode primitives, and `\pdfprimitive`, `\discretionary`, as well as all 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`, `\currentiflevel` and `\currentifttype` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgrouptype` too.
- Tokens passed to `\afterassignment` are not yet kept after `unravel` is done even if there has been no assignment.
- Tokens passed to `\aftergroup` are lost when `unravel` is done.
- In `XYTEX`, characters beyond the basic multilingual plane may break `unravel` (not tested).
- 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 régime 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

- Allow users to change some settings globally/for one `\unravel`.
- Allow to replay steps that have already been run.
- Fix the display for `\if` and `\ifcat` (remove extraneous `\exp_not:N`).
- 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.
- Look at all places where T_EX's procedure `prepare_mag` is called.
- 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 T_EX 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 ε -T_EX's `\numexpr` or `\protected` are not available, bail out with an error.

```
9 \expandafter\ifx\c\name numexpr\endcsname\relax
10 \errmessage{unravel requires \numexpr from eTeX}
11 \endinput\expandafter\endgroup\fi
12 \expandafter\ifx\c\name 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 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 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`.
`__unravel_setup_save_aux:n`

```

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 definition for __unravel_setup_save: and __unravel_setup_save_aux:n.)

`__unravel_setup_catcodes:nnn` This sets all characters from #1 to #2 (inclusive) to have catcode #3.

```

45 \protected \long \gdef \__unravel_setup_catcodes:nnn #1 #2 #3
46 {
47     \ifnum #1 > #2 ~ \else

```

```

48     \catcode #1 = #3 ~
49     \expandafter \_unravel_setup_catcodes:nnn \expandafter
50     { \the \numexpr #1 + 1 } {#2} {#3}
51   \fi
52 }

```

(End definition for _unravel_setup_catcodes:nnn.)

_unravel_setup_latex: This saves the catcodes and related parameters, then sets them to the value they normally have in a L^AT_EX 2_ε package (in particular, @ is a letter).

```

53 \protected \gdef \_unravel_setup_latex:
54 {
55   \_unravel_setup_save:
56   \_unravel_setup_catcodes:nnn {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:nnn {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:nnn {39} {63} {12}
71   \_unravel_setup_catcodes:nnn {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:nnn {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:nnn {128} {255} {12}
85   \endlinechar = 13 ~
86   \escapechar = 92 ~
87   \newlinechar = 10 ~
88 }

```

(End 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:nnn {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:nnn {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:nnn {39} {57} {12}
107  \catcode 58 = 11 ~
108  \__unravel_setup_catcodes:nnn {59} {64} {12}
109  \__unravel_setup_catcodes:nnn {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:nnn {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:nnn {128} {255} {12}
123  \escapechar = 92 ~
124  \endlinechar = 32 ~
125  \newlinechar = 10 ~
126  }

```

(End definition for __unravel_setup_unravel:.)

End the group where all catcodes were changed, but expand __unravel_setup_latex: to sanitize catcodes again outside the group. The catcodes are saved.

```

127 \expandafter \endgroup \__unravel_setup_latex:

```

Load a few dependencies: expl3, xparse, gtl. Load l3str if expl3 is too old and does not define \str_range:nnn. Otherwise loading l3str would give an error.

```

128 \RequirePackage{expl3,xparse}[2018/02/21]
129 \RequirePackage{gtl}[2018/12/28]
130 \csname cs_if_exist:cF\endcsname{str_range:nnn}{\RequirePackage{l3str}}

```

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 T_EX read the arguments of \ProvidesExplPackage before restoring catcodes. Then immediately go to the catcodes we want.

```

131 \csname use:n\endcsname
132 {%
133   \csname __unravel_setup_restore:\endcsname
134   \ProvidesExplPackage
135     {unravel} {2018/12/28} {0.2f} {Watching TeX digest tokens}%
136   \csname __unravel_setup_unravel:\endcsname
137 }%

```

2.1 Primitives, variants, and helpers

2.1.1 Renamed primitives

`__unravel_currentgrouptype:` 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).

```

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

```

(End 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.

```

148 \exp_after:wN \cs_new_eq:NN
149   \exp_after:wN \__unravel_special_relax:
150   \exp_not:N \__unravel_special_relax:

```

(End definition for `__unravel_special_relax:.`)

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

```

151 \int_const:Nn \c__unravel_prompt_ior { 16 }
152 \int_const:Nn \c__unravel_noprompt_ior { -1 }

```

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

2.1.2 Variants

Variants that we need.

```

153 \cs_generate_variant:Nn \seq_push:Nn { Nf }
154 \cs_generate_variant:Nn \str_head:n { f }
155 \cs_generate_variant:Nn \tl_to_str:n { o }
156 \cs_generate_variant:Nn \tl_if_eq:nnTF { o }
157 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
158 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
159 \cs_generate_variant:Nn \gtl_gput_right:Nn { NV }

```

```

160 \cs_generate_variant:Nn \ior_str_get:NN { Nc }
161 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
162 \cs_generate_variant:Nn \gtl_to_str:N { c }
163 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
164 \cs_generate_variant:Nn \gtl_get_left:NN { c }
165 \cs_generate_variant:Nn \gtl_gset:Nn { c }
166 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
167 \cs_generate_variant:Nn \gtl_gclear:N { c }
168 \cs_generate_variant:Nn \gtl_gclear_new: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).

```

169 \cs_new_protected:Npn \__unravel_tl_if_in:ooTF #1#2#3#4
170 {
171   \group_begin:
172   \exp_args:Noo \tl_if_in:nnTF {#1} {#2}
173   { \group_end: #3 } { \group_end: #4 }
174 }

```

(End definition for __unravel_tl_if_in:ooTF.)

`\l__unravel_exp_tl` Low-level because `\exp_args:Nx` redefines an internal `\l3expan` variable which may be appearing in code that we debug.

```

\__unravel_exp_args:Nx
\__unravel_exp_args:NNx
175 \tl_new:N \l__unravel_exp_tl
176 \cs_new_protected:Npn \__unravel_exp_args:Nx #1#2
177 {
178   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 {#2} }
179   \l__unravel_exp_tl
180 }
181 \cs_new_protected:Npn \__unravel_exp_args:NNx #1#2#3
182 {
183   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 \exp_not:N #2 {#3} }
184   \l__unravel_exp_tl
185 }

```

(End definition for \l__unravel_exp_tl, __unravel_exp_args:Nx, and __unravel_exp_args:NNx.)

2.1.3 Miscellaneous helpers

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

```

186 \cs_new_protected:Npn \__unravel_tmp:w { }

```

(End definition for __unravel_tmp:w.)

```

\__unravel_file_get:nN
\__unravel_file_get_aux:wN
187 \cs_set_protected:Npn \__unravel_tmp:w #1
188 {
189   \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
190   {
191     \group_begin:
192     \__unravel_everyeof:w { #1 ##2 }
193     \exp_after:wN \__unravel_file_get_aux:wN
194     \exp_after:wN \prg_do_nothing:
195     \tex_input:D ##1 \scan_stop:

```

```

196     }
197     \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
198     {
199         \group_end:
200         \tl_set:Nx ##2
201         { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
202     }
203 }
204 \exp_args:No \__unravel_tmp:w { \token_to_str:N : : }

```

(End definition for __unravel_file_get:nN and __unravel_file_get_aux:wN.)

__unravel_tl_first_int:N 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.

```

205 \cs_new:Npn \__unravel_tl_first_int:N #1
206 {
207     \int_eval:n
208     {
209         \exp_after:wN \__unravel_tl_first_int_aux:Nn
210         \exp_after:wN \__unravel_tl_first_int_aux:Nn
211         #1 ? 0 ? \q_stop
212     }
213 }
214 \cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
215 {
216     \tl_if_single:nT {#2}
217     {
218         \token_if_eq_catcode:NNT + #2
219         {
220             \if_int_compare:w 1 < 1 #2 \exp_stop_f:
221             #2
222             \exp_after:wN \use_i_ii:nnn
223             \exp_after:wN \__unravel_tl_first_int_aux:Nn
224             \exp_after:wN \use_none_delimit_by_q_stop:w
225             \fi:
226         }
227     }
228     #1
229 }

```

(End definition for __unravel_tl_first_int:N and __unravel_tl_first_int_aux:Nn.)

__unravel_prepare_mag: Used whenever T_EX needs the value of \mag.

```

230 \cs_new_protected:Npn \__unravel_prepare_mag:
231 {
232     \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
233     {
234         \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }

```

```

235         {
236             \__unravel_tex_error:nN { incompatible-mag } { }
237             \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
238         }
239     }
240     \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
241     {
242         \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
243         \int_gset:Nn \__unravel_mag: { 1000 }
244     }
245     \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
246 }

```

(End definition for __unravel_prepare_mag:.)

2.1.4 String helpers

__unravel_strip_escape: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.

```

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

```

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

```

\__unravel_to_str:n Use the type-appropriate conversion to string.
\__unravel_to_str_auxi:w
\__unravel_to_str_auxii:w
\__unravel_gtl_to_str:n
256 \cs_new:Npn \__unravel_to_str:n #1
257 {
258     \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
259     { \__unravel_to_str_auxi:w #1 ? \q_stop }
260     { \tl_to_str:n }
261     {#1}
262 }
263 \cs_set:Npn \__unravel_tmp:w #1
264 {
265     \cs_new:Npn \__unravel_to_str_auxi:w ##1##2 \q_stop
266     {
267         \exp_after:wN \__unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark

```

```

268         #1 t1 \q_mark \q_stop
269     }
270     \cs_new:Npn \__unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
271     { \cs_if_exist_use:cF { __unravel_ ##2 _to_str:n } { \tl_to_str:n } }
272 }
273 \exp_args:No \__unravel_tmp:w { \tl_to_str:n { s _ _ } }
274 \cs_new:Npn \__unravel_gtl_to_str:n { \gtl_to_str:n }

```

(End definition for __unravel_to_str:n and others.)

__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

```

275 \cs_new:Npn \__unravel_str_truncate_left:nn #1#2
276 {
277     \exp_args:Nf \__unravel_str_truncate_left_aux:nnn
278     { \str_count:n {#1} } {#1} {#2}
279 }
280 \cs_new:Npn \__unravel_str_truncate_left_aux:nnn #1#2#3
281 {
282     \int_compare:nNnTF {#1} > {#3}
283     {
284         ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
285         \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
286     }
287     { \tl_to_str:n {#2} }
288 }

```

(End 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

```

289 \cs_new:Npn \__unravel_str_truncate_right:nn #1#2
290 {
291     \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
292     { \str_count:n {#1} } {#1} {#2}
293 }
294 \cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
295 {
296     \int_compare:nNnTF {#1} > {#3}
297     {
298         \str_range:nnn {#2} { 1 } { #3 - 25 } ~
299         ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
300     }
301     { \tl_to_str:n {#2} }
302 }

```

(End definition for __unravel_str_truncate_right:nn and __unravel_str_truncate_right_aux:nnn.)

2.1.5 Helpers for control flow

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

__unravel_exit_point:

```

303 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
304 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }

```

(End definition for `_unravel_exit:w` and `_unravel_exit_point:.`)

```

\_unravel_break:w Useful to jump out of complicated conditionals.
\_unravel_break_point: 305 \cs_new_eq:NN \_unravel_break_point: \prg_do_nothing:
306 \cs_new:Npn \_unravel_break:w #1 \_unravel_break_point: { }

```

(End 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).

```

307 \prg_new_conditional:Npnn \_unravel_cmd_if_internal: { TF }
308 {
309   \int_compare:nNnTF
310     \l\_unravel_head_cmd_int < { \_unravel_tex_use:n { min_internal } }
311     { \prg_return_false: }
312     {
313       \int_compare:nNnTF
314         \l\_unravel_head_cmd_int
315         > { \_unravel_tex_use:n { max_internal } }
316         { \prg_return_false: }
317         { \prg_return_true: }
318     }
319 }

```

(End definition for `_unravel_cmd_if_internal:TF`.)

2.1.6 Helpers concerning tokens

`_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.

```

\_unravel_meaning_to_char_auxi:w 320 \cs_new:Npn \_unravel_meaning_to_char:n #1
\_unravel_meaning_to_char_auxii:w 321 { \_unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
322 \cs_new:Npn \_unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
323 { \_unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
324 \cs_new:Npn \_unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
325 { \tl_if_empty:nTF {#2} { ~ } {#2} }
326 \cs_generate_variant:Nn \_unravel_meaning_to_char:n { o }
327 \cs_new:Npn \_unravel_token_to_char:N #1
328 { \_unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

(End 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:N` because the `expl3`
`_unravel_token_if_expandable:N` TF one does not think that `undefined` is expandable.

```

329 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
330 { p , T , F , TF }
331 {
332   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
333   \prg_return_false:
334   \else:
335     \prg_return_true:
336   \fi:
337 }

```

(End 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
338 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
339 { p , T , F , TF }
340 {
341   \_unravel_token_if_expandable:NTF #1
342   {
343     \token_if_protected_macro:NTF #1
344     { \prg_return_true: }
345     {
346       \token_if_protected_long_macro:NTF #1
347       { \prg_return_true: }
348       { \prg_return_false: }
349     }
350   }
351   { \prg_return_true: }
352 }

```

(End 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.

```

353 \group_begin:
354   \char_set_catcode_active:n { 'Z }
355   \prg_new_protected_conditional:Npnn \_unravel_token_if_active_char:N #1
356   { TF }
357   {
358     \group_begin:
359       \_unravel_exp_args:Nx \char_set_lccode:nn
360       { ' \exp_args:No \str_head:n { \token_to_str:N #1 } }
361       { ' Z }
362       \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
363       { \group_end: \prg_return_true: }
364       { \group_end: \prg_return_false: }
365     }
366   \group_end:

```

(End 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.

```

367 \group_begin:
368   \char_set_catcode_active:n { 'Z }
369   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
370   { TF }
371   {

```

```

372 \group_begin:
373   \__unravel_set_escapechar:n { 92 }
374   \tl_set:Nx \l__unravel_tmpa_tl
375     { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
376   \tl_if_empty:NTF \l__unravel_tmpa_tl
377     {
378       \__unravel_token_if_active_char:NTF #1
379       { \group_end: \prg_return_true: }
380       { \group_end: \prg_return_false: }
381     }
382     { \group_end: \prg_return_true: }
383   }
384 \group_end:

```

(End 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.

```

385 \prg_new_protected_conditional:Npnn \__unravel_gtl_if_head_is_definable:N #1
386 { TF , F }
387 {
388   \gtl_if_single_token:NTF #1
389   {
390     \gtl_if_head_is_N_type:NTF #1
391     {
392       \gtl_head_do:NN #1 \__unravel_token_if_definable:NTF
393       { \prg_return_true: }
394       { \prg_return_false: }
395     }
396     { \prg_return_false: }
397   }
398   { \prg_return_false: }
399 }

```

(End definition for __unravel_gtl_if_head_is_definable:NTF.)

2.1.7 Helpers for previous input

```

\__unravel_prev_input_count:
  \__unravel_prev_input_count_aux:n
400 \cs_new:Npn \__unravel_prev_input_count:
401 {
402   \int_eval:n
403   {
404     0
405     \seq_map_function:NN \g__unravel_prev_input_seq
406       \__unravel_prev_input_count_aux:n
407   }
408 }
409 \cs_new:Npn \__unravel_prev_input_count_aux:n #1
410 { \tl_if_empty:nF {#1} { + 1 } }

```

(End definition for __unravel_prev_input_count: and __unravel_prev_input_count_aux:n.)

```

\__unravel_prev_input_get:N
\__unravel_prev_input_gpush:
    \__unravel_prev_input_gpush:N
\__unravel_prev_input_gpop:N
    \__unravel_prev_input_gpush_gtl:
    \__unravel_prev_input_gpush_gtl:N
    \__unravel_prev_input_gpop_gtl:N
411 \cs_new_protected:Npn \__unravel_prev_input_get:N
412 { \seq_get_right:NN \g__unravel_prev_input_seq }
413 \cs_new_protected:Npn \__unravel_prev_input_gpush:
414 { \seq_gput_right:Nn \g__unravel_prev_input_seq { } }
415 \cs_new_protected:Npn \__unravel_prev_input_gpush:N
416 { \seq_gput_right:NV \g__unravel_prev_input_seq }
417 \cs_new_protected:Npn \__unravel_prev_input_gpop:N
418 { \seq_gpop_right:NN \g__unravel_prev_input_seq }
419 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
420 { \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl }
421 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
422 { \seq_gput_right:NV \g__unravel_prev_input_seq }
423 \cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
424 { \seq_gpop_right:NN \g__unravel_prev_input_seq }

```

(End definition for __unravel_prev_input_get:N and others.)

```

\__unravel_prev_input_silent:n
\__unravel_prev_input_silent:V
\__unravel_prev_input_silent:x
\__unravel_prev_input:n
\__unravel_prev_input:V
\__unravel_prev_input:x
425 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
426 {
427     \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
428     \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
429     \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
430 }
431 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V }
432 \cs_new_protected:Npn \__unravel_prev_input_silent:x
433 { \__unravel_exp_args:Nx \__unravel_prev_input_silent:n }
434 \cs_new_protected:Npn \__unravel_prev_input:n #1
435 {
436     \__unravel_prev_input_silent:n {#1}
437     \__unravel_print_action:x { \tl_to_str:n {#1} }
438 }
439 \cs_generate_variant:Nn \__unravel_prev_input:n { V }
440 \cs_new_protected:Npn \__unravel_prev_input:x
441 { \__unravel_exp_args:Nx \__unravel_prev_input:n }

```

(End definition for __unravel_prev_input_silent:n and __unravel_prev_input:n.)

```

\__unravel_prev_input_gtl:N
442 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
443 {
444     \__unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
445     \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
446     \__unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
447 }

```

(End definition for __unravel_prev_input_gtl:N.)

__unravel_prev_input_join_get:nN 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.

```

448 \cs_new_protected:Npn \__unravel_prev_input_join_get:nN #1
449 {

```

```

450 \int_case:nnF {#1}
451 {
452   { 2 } { \_unravel_join_get_aux:NNN \skip_eval:n \tex_glueexpr:D }
453   { 3 } { \_unravel_join_get_aux:NNN \muskip_eval:n \tex_muexpr:D }
454 }
455 {
456   \_unravel_error:nnnnn { internal } { join-factor } { } { } { }
457   \_unravel_join_get_aux:NNN \use:n \prg_do_nothing:
458 }
459 }
460 \cs_new_protected:Npn \_unravel_join_get_aux:NNN #1#2#3
461 {
462   \_unravel_prev_input_gpop:N \l__unravel_head_tl
463   \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
464   \tl_set:Nx #3 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl } }
465 }

```

(End definition for _unravel_prev_input_join_get:nN and _unravel_join_get_aux:NNN.)

2.2 Variables

2.2.1 User interaction

\g_unravel_before_print_state_tl
\g_unravel_before_prompt_tl

Code to run before printing the state or before the prompt.

```

466 \tl_new:N \g_unravel_before_print_state_tl
467 \tl_new:N \g_unravel_before_prompt_tl

```

(End definition for \g_unravel_before_print_state_tl and \g_unravel_before_prompt_tl.)

\l__unravel_prompt_tmpa_int

```

468 \int_new:N \l__unravel_prompt_tmpa_int

```

(End definition for \l__unravel_prompt_tmpa_int.)

\g_unravel_nonstop_int

The number of prompts to skip.

```

469 \int_new:N \g_unravel_nonstop_int

```

(End definition for \g_unravel_nonstop_int.)

\g_unravel_default_explicit_prompt_bool
\g_unravel_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.

\g_unravel_default_internal_debug_bool
\g_unravel_internal_debug_bool

```

470 \bool_new:N \g_unravel_default_explicit_prompt_bool

```

```

471 \bool_new:N \g_unravel_default_internal_debug_bool

```

```

472 \bool_new:N \g_unravel_default_number_steps_bool

```

```

473 \int_new:N \g_unravel_default_online_int

```

```

474 \bool_new:N \g_unravel_default_trace_assigns_bool

```

```

475 \bool_new:N \g_unravel_default_trace_expansion_bool

```

```

476 \bool_new:N \g_unravel_default_trace_other_bool

```

```

477 \bool_new:N \g_unravel_default_welcome_message_bool

```

```

478 \bool_gset_true:N \g_unravel_default_number_steps_bool

```

```

479 \int_gset:Nn \g_unravel_default_online_int { 1 }

```

```

480 \bool_gset_true:N \g_unravel_default_trace_assigns_bool

```

```

481 \bool_gset_true:N \g_unravel_default_trace_expansion_bool

```

```

482 \bool_gset_true:N \g_unravel_default_trace_other_bool

```

\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
\g_unravel_welcome_message_bool

```

483 \bool_gset_true:N \g__unravel_default_welcome_message_bool
484 \bool_new:N \g__unravel_explicit_prompt_bool
485 \bool_new:N \g__unravel_internal_debug_bool
486 \bool_new:N \g__unravel_number_steps_bool
487 \int_new:N \g__unravel_online_int
488 \bool_new:N \g__unravel_trace_assigns_bool
489 \bool_new:N \g__unravel_trace_expansion_bool
490 \bool_new:N \g__unravel_trace_other_bool
491 \bool_new:N \g__unravel_welcome_message_bool

```

(End definition for \g__unravel_default_explicit_prompt_bool and others.)

\g__unravel_step_int Current expansion step.

```

492 \int_new:N \g__unravel_step_int

```

(End 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:x, which sets the escape character as appropriate before converting the argument to a string.

```

493 \str_new:N \g__unravel_action_text_str

```

(End 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
\g__unravel_default_max_input_int
\g__unravel_max_action_int
\g__unravel_max_output_int
\g__unravel_max_input_int
494 \int_new:N \g__unravel_default_max_action_int
495 \int_new:N \g__unravel_default_max_output_int
496 \int_new:N \g__unravel_default_max_input_int
497 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
498 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
499 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
500 \int_new:N \g__unravel_max_action_int
501 \int_new:N \g__unravel_max_output_int
502 \int_new:N \g__unravel_max_input_int

```

(End 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.

```

503 \bool_new:N \g__unravel_speedup_macros_bool
504 \bool_gset_true:N \g__unravel_speedup_macros_bool

```

(End definition for \g__unravel_speedup_macros_bool.)

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

```

505 \int_new:N \l__unravel_print_int

```

(End 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 T_EX'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.

```
506 \int_new:N \g__unravel_input_int
```

(End definition for `\g__unravel_input_int`.)

`\g__unravel_input_tmpa_int`
`\l__unravel_input_tmpa_tl`

```
507 \int_new:N \g__unravel_input_tmpa_int
```

```
508 \tl_new:N \l__unravel_input_tmpa_tl
```

(End 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.

```
509 \seq_new:N \g__unravel_prev_input_seq
```

```
510 \tl_new:N \l__unravel_prev_input_tl
```

```
511 \gtl_new:N \l__unravel_prev_input_gtl
```

(End 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 T_EX's digestion.

```
512 \gtl_new:N \g__unravel_output_gtl
```

(End 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 T_EX.

```
513 \gtl_new:N \l__unravel_head_gtl
```

```
514 \tl_new:N \l__unravel_head_tl
```

```
515 \cs_new_eq:NN \l__unravel_head_token ?
```

```
516 \int_new:N \l__unravel_head_cmd_int
```

```
517 \int_new:N \l__unravel_head_char_int
```

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

`\l__unravel_head_meaning_tl`

```
518 \tl_new:N \l__unravel_head_meaning_tl
```

(End definition for `\l__unravel_head_meaning_tl`.)

`\l__unravel_argi_tl`
`\l__unravel_argii_tl`

Token list variables used to store first/second arguments.

```
519 \tl_new:N \l__unravel_argi_tl
```

```
520 \tl_new:N \l__unravel_argii_tl
```

(End 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` 521 `\tl_new:N \l__unravel_tmpa_tl`

`\l__unravel_tmpa_seq` 522 `\gtl_new:N \l__unravel_unused_gtl`

`\l__unravel_unused_gtl` 523 `\gtl_new:N \l__unravel_tmpb_gtl`

`\l__unravel_tmpb_token` 524 `\tl_new:N \g__unravel_tmpc_tl`

525 `\seq_new:N \l__unravel_tmpa_seq`

526 `\cs_new_eq:NN \l__unravel_tmpb_token ?`

(End 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: rather, this sequence contains a string representation of the code, which is not suitable for the definition. This 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.

`\l__unravel_defining_tl`

527 `\tl_new:N \l__unravel_defined_tl`

528 `\tl_new:N \l__unravel_defining_tl`

(End definition for \l__unravel_defined_tl and \l__unravel_defining_tl.)

`__unravel_inaccessible:w`

529 `\cs_new_eq:NN __unravel_inaccessible:w ?`

(End definition for __unravel_inaccessible:w.)

`\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`

530 `\gtl_new:N \g__unravel_after_assignment_gtl`

531 `\bool_new:N \g__unravel_set_box_allowed_bool`

532 `\bool_new:N \g__unravel_name_in_progress_bool`

(End 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.

533 `\gtl_new:N \l__unravel_after_group_gtl`

(End definition for \l__unravel_after_group_gtl.)

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

534 `\group_begin:`

535 `\cs_set:Npx __unravel_tmp:w #1 { \c_hash_str #1 }`

536 `\tl_const:Nx \c__unravel_parameters_tl`

537 `{ ^ \tl_map_function:nN { 123456789 } __unravel_tmp:w }`

538 `\group_end:`

(End 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_scan_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.

539 `\int_new:N \g__unravel_val_level_int`

(End 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` 540 `\tl_new:N \g__unravel_if_limit_tl`

`\g__unravel_if_depth_int` 541 `\int_new:N \g__unravel_if_limit_int`

542 `\int_new:N \g__unravel_if_depth_int`

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

`\l__unravel_if_nesting_int`

543 `\int_new:N \l__unravel_if_nesting_int`

(End 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.

544 `\seq_new:N \l__unravel_leaders_box_seq`

(End 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.

545 `\int_new:N \g__unravel_ends_int`

546 `\int_gset:Nn \g__unravel_ends_int { 3 }`

(End definition for `\g__unravel_ends_int`.)

2.2.5 Constants

```

\c__unravel_plus_tl
\c__unravel_minus_tl
\c__unravel_times_tl
\c__unravel_over_tl
\c__unravel_lq_tl
\c__unravel_rq_tl
\c__unravel_dq_tl
\c__unravel_lp_tl
\c__unravel_rp_tl
\c__unravel_eq_tl
\c__unravel_comma_tl
\c__unravel_point_tl
547 \tl_const:Nn \c__unravel_plus_tl { + }
548 \tl_const:Nn \c__unravel_minus_tl { - }
549 \tl_const:Nn \c__unravel_times_tl { * }
550 \tl_const:Nn \c__unravel_over_tl { / }
551 \tl_const:Nn \c__unravel_lq_tl { ' }
552 \tl_const:Nn \c__unravel_rq_tl { ' }
553 \tl_const:Nn \c__unravel_dq_tl { " }
554 \tl_const:Nn \c__unravel_lp_tl { ( }
555 \tl_const:Nn \c__unravel_rp_tl { ) }
556 \tl_const:Nn \c__unravel_eq_tl { = }
557 \tl_const:Nn \c__unravel_comma_tl { , }
558 \tl_const:Nn \c__unravel_point_tl { . }

```

(End definition for \c__unravel_plus_tl and others.)

```

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

```

(End 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.

```

560 \int_new:N \g__unravel_mag_set_int

```

(End definition for \g__unravel_mag_set_int.)

2.3 Numeric codes

First we define some numeric codes, following Section 15 of the TeX web code, then we associate a command code to each TeX primitive, and a character code, to decide what action to perform upon seeing them.

```

\__unravel_tex_const:nn
\__unravel_tex_use:n
561 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
562 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
563 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }

```

(End definition for __unravel_tex_const:nn and __unravel_tex_use:n.)

```

\__unravel_tex_primitive:nnn
564 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
565 {
566   \tl_const:cx { c__unravel_tex_#1_tl }
567   { { \__unravel_tex_use:n {#2} } {#3} }
568 }

```

(End definition for __unravel_tex_primitive:nnn.)

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
569 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
570 {
571   \cs_new_protected:cpn
572     { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
573 }
574 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
575 {
576   \cs_new_eq:cc
577     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
578     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
579 }

(End definition for \__unravel_new_tex_cmd:nn and \__unravel_new_eq_tex_cmd:nn.)

```

```

\__unravel_new_tex_expandable:nn
580 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
581 {
582   \cs_new_protected:cpn
583     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
584 }

(End definition for \__unravel_new_tex_expandable:nn.)

```

Contrarily to T_EX, all macros are call, no long_call and the like.

```

585 \__unravel_tex_const:nn { relax } { 0 }
586 \__unravel_tex_const:nn { begin-group_char } { 1 }
587 \__unravel_tex_const:nn { end-group_char } { 2 }
588 \__unravel_tex_const:nn { math_char } { 3 }
589 \__unravel_tex_const:nn { tab_mark } { 4 }
590 \__unravel_tex_const:nn { alignment_char } { 4 }
591 \__unravel_tex_const:nn { car_ret } { 5 }
592 \__unravel_tex_const:nn { macro_char } { 6 }
593 \__unravel_tex_const:nn { superscript_char } { 7 }
594 \__unravel_tex_const:nn { subscript_char } { 8 }
595 \__unravel_tex_const:nn { endv } { 9 }
596 \__unravel_tex_const:nn { blank_char } { 10 }
597 \__unravel_tex_const:nn { the_char } { 11 }
598 \__unravel_tex_const:nn { other_char } { 12 }
599 \__unravel_tex_const:nn { par_end } { 13 }
600 \__unravel_tex_const:nn { stop } { 14 }
601 \__unravel_tex_const:nn { delim_num } { 15 }
602 \__unravel_tex_const:nn { max_char_code } { 15 }
603 \__unravel_tex_const:nn { char_num } { 16 }
604 \__unravel_tex_const:nn { math_char_num } { 17 }
605 \__unravel_tex_const:nn { mark } { 18 }
606 \__unravel_tex_const:nn { xray } { 19 }
607 \__unravel_tex_const:nn { make_box } { 20 }
608 \__unravel_tex_const:nn { hmove } { 21 }
609 \__unravel_tex_const:nn { vmove } { 22 }
610 \__unravel_tex_const:nn { un_hbox } { 23 }
611 \__unravel_tex_const:nn { un_vbox } { 24 }
612 \__unravel_tex_const:nn { remove_item } { 25 }
613 \__unravel_tex_const:nn { hskip } { 26 }
614 \__unravel_tex_const:nn { vskip } { 27 }

```

615	<code>_unravel_tex_const:nn { mskip</code>	<code>} { 28 }</code>
616	<code>_unravel_tex_const:nn { kern</code>	<code>} { 29 }</code>
617	<code>_unravel_tex_const:nn { mkern</code>	<code>} { 30 }</code>
618	<code>_unravel_tex_const:nn { leader_ship</code>	<code>} { 31 }</code>
619	<code>_unravel_tex_const:nn { halign</code>	<code>} { 32 }</code>
620	<code>_unravel_tex_const:nn { valign</code>	<code>} { 33 }</code>
621	<code>_unravel_tex_const:nn { no_align</code>	<code>} { 34 }</code>
622	<code>_unravel_tex_const:nn { vrule</code>	<code>} { 35 }</code>
623	<code>_unravel_tex_const:nn { hrule</code>	<code>} { 36 }</code>
624	<code>_unravel_tex_const:nn { insert</code>	<code>} { 37 }</code>
625	<code>_unravel_tex_const:nn { vadjust</code>	<code>} { 38 }</code>
626	<code>_unravel_tex_const:nn { ignore_spaces</code>	<code>} { 39 }</code>
627	<code>_unravel_tex_const:nn { after_assignment</code>	<code>} { 40 }</code>
628	<code>_unravel_tex_const:nn { after_group</code>	<code>} { 41 }</code>
629	<code>_unravel_tex_const:nn { break_penalty</code>	<code>} { 42 }</code>
630	<code>_unravel_tex_const:nn { start_par</code>	<code>} { 43 }</code>
631	<code>_unravel_tex_const:nn { ital_corr</code>	<code>} { 44 }</code>
632	<code>_unravel_tex_const:nn { accent</code>	<code>} { 45 }</code>
633	<code>_unravel_tex_const:nn { math_accent</code>	<code>} { 46 }</code>
634	<code>_unravel_tex_const:nn { discretionary</code>	<code>} { 47 }</code>
635	<code>_unravel_tex_const:nn { eq_no</code>	<code>} { 48 }</code>
636	<code>_unravel_tex_const:nn { left_right</code>	<code>} { 49 }</code>
637	<code>_unravel_tex_const:nn { math_comp</code>	<code>} { 50 }</code>
638	<code>_unravel_tex_const:nn { limit_switch</code>	<code>} { 51 }</code>
639	<code>_unravel_tex_const:nn { above</code>	<code>} { 52 }</code>
640	<code>_unravel_tex_const:nn { math_style</code>	<code>} { 53 }</code>
641	<code>_unravel_tex_const:nn { math_choice</code>	<code>} { 54 }</code>
642	<code>_unravel_tex_const:nn { non_script</code>	<code>} { 55 }</code>
643	<code>_unravel_tex_const:nn { vcenter</code>	<code>} { 56 }</code>
644	<code>_unravel_tex_const:nn { case_shift</code>	<code>} { 57 }</code>
645	<code>_unravel_tex_const:nn { message</code>	<code>} { 58 }</code>
646	<code>_unravel_tex_const:nn { extension</code>	<code>} { 59 }</code>
647	<code>_unravel_tex_const:nn { in_stream</code>	<code>} { 60 }</code>
648	<code>_unravel_tex_const:nn { begin_group</code>	<code>} { 61 }</code>
649	<code>_unravel_tex_const:nn { end_group</code>	<code>} { 62 }</code>
650	<code>_unravel_tex_const:nn { omit</code>	<code>} { 63 }</code>
651	<code>_unravel_tex_const:nn { ex_space</code>	<code>} { 64 }</code>
652	<code>_unravel_tex_const:nn { no_boundary</code>	<code>} { 65 }</code>
653	<code>_unravel_tex_const:nn { radical</code>	<code>} { 66 }</code>
654	<code>_unravel_tex_const:nn { end_cs_name</code>	<code>} { 67 }</code>
655	<code>_unravel_tex_const:nn { min_internal</code>	<code>} { 68 }</code>
656	<code>_unravel_tex_const:nn { char_given</code>	<code>} { 68 }</code>
657	<code>_unravel_tex_const:nn { math_given</code>	<code>} { 69 }</code>
658	<code>_unravel_tex_const:nn { last_item</code>	<code>} { 70 }</code>
659	<code>_unravel_tex_const:nn { max_non_prefixed_command</code>	<code>} { 70 }</code>
660	<code>_unravel_tex_const:nn { toks_register</code>	<code>} { 71 }</code>
661	<code>_unravel_tex_const:nn { assign_toks</code>	<code>} { 72 }</code>
662	<code>_unravel_tex_const:nn { assign_int</code>	<code>} { 73 }</code>
663	<code>_unravel_tex_const:nn { assign_dimen</code>	<code>} { 74 }</code>
664	<code>_unravel_tex_const:nn { assign_glue</code>	<code>} { 75 }</code>
665	<code>_unravel_tex_const:nn { assign_mu_glue</code>	<code>} { 76 }</code>
666	<code>_unravel_tex_const:nn { assign_font_dimen</code>	<code>} { 77 }</code>
667	<code>_unravel_tex_const:nn { assign_font_int</code>	<code>} { 78 }</code>
668	<code>_unravel_tex_const:nn { set_aux</code>	<code>} { 79 }</code>

```

669 \_unravel_tex_const:nn { set_prev_graf          } { 80 }
670 \_unravel_tex_const:nn { set_page_dimen         } { 81 }
671 \_unravel_tex_const:nn { set_page_int           } { 82 }
672 \_unravel_tex_const:nn { set_box_dimen          } { 83 }
673 \_unravel_tex_const:nn { set_shape              } { 84 }
674 \_unravel_tex_const:nn { def_code               } { 85 }
675 \_unravel_tex_const:nn { def_family             } { 86 }
676 \_unravel_tex_const:nn { set_font               } { 87 }
677 \_unravel_tex_const:nn { def_font              } { 88 }
678 \_unravel_tex_const:nn { register               } { 89 }
679 \_unravel_tex_const:nn { max_internal           } { 89 }
680 \_unravel_tex_const:nn { advance                } { 90 }
681 \_unravel_tex_const:nn { multiply               } { 91 }
682 \_unravel_tex_const:nn { divide                } { 92 }
683 \_unravel_tex_const:nn { prefix                } { 93 }
684 \_unravel_tex_const:nn { let                   } { 94 }
685 \_unravel_tex_const:nn { shorthand_def          } { 95 }
686 \_unravel_tex_const:nn { read_to_cs             } { 96 }
687 \_unravel_tex_const:nn { def                   } { 97 }
688 \_unravel_tex_const:nn { set_box                } { 98 }
689 \_unravel_tex_const:nn { hyph_data             } { 99 }
690 \_unravel_tex_const:nn { set_interaction         } { 100 }
691 \_unravel_tex_const:nn { letterspace_font       } { 101 }
692 \_unravel_tex_const:nn { pdf_copy_font          } { 102 }
693 \_unravel_tex_const:nn { max_command            } { 102 }
694 \_unravel_tex_const:nn { undefined_cs           } { 103 }
695 \_unravel_tex_const:nn { expand_after           } { 104 }
696 \_unravel_tex_const:nn { no_expand              } { 105 }
697 \_unravel_tex_const:nn { input                 } { 106 }
698 \_unravel_tex_const:nn { if_test                } { 107 }
699 \_unravel_tex_const:nn { fi_or_else            } { 108 }
700 \_unravel_tex_const:nn { cs_name                } { 109 }
701 \_unravel_tex_const:nn { convert               } { 110 }
702 \_unravel_tex_const:nn { the                   } { 111 }
703 \_unravel_tex_const:nn { top_bot_mark          } { 112 }
704 \_unravel_tex_const:nn { call                   } { 113 }
705 \_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 LuaTeX 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.

```

706 \sys_if_engine luatex:T
707 {
708   \__unravel_tex_primitive:nnn
709     { \exp_after:wN \use_none:n \token_to_meaning:N \__unravel_special_relax: }
710     { relax } { 1 }
711 }
712 \__unravel_tex_primitive:nnn { relax } { relax } { 256 }
713 \__unravel_tex_primitive:nnn { span } { tab_mark } { 256 }
714 \__unravel_tex_primitive:nnn { cr } { car_ret } { 257 }
715 \__unravel_tex_primitive:nnn { crcr } { car_ret } { 258 }
716 \__unravel_tex_primitive:nnn { par } { par_end } { 256 }
717 \__unravel_tex_primitive:nnn { end } { stop } { 0 }
718 \__unravel_tex_primitive:nnn { dump } { stop } { 1 }
719 \__unravel_tex_primitive:nnn { delimiter } { delim_num } { 0 }
720 \__unravel_tex_primitive:nnn { char } { char_num } { 0 }
721 \__unravel_tex_primitive:nnn { mathchar } { math_char_num } { 0 }
722 \__unravel_tex_primitive:nnn { mark } { mark } { 0 }
723 \__unravel_tex_primitive:nnn { marks } { mark } { 5 }
724 \__unravel_tex_primitive:nnn { show } { xray } { 0 }
725 \__unravel_tex_primitive:nnn { showbox } { xray } { 1 }
726 \__unravel_tex_primitive:nnn { showthe } { xray } { 2 }
727 \__unravel_tex_primitive:nnn { showlists } { xray } { 3 }
728 \__unravel_tex_primitive:nnn { showgroups } { xray } { 4 }
729 \__unravel_tex_primitive:nnn { showtokens } { xray } { 5 }
730 \__unravel_tex_primitive:nnn { showifs } { xray } { 6 }
731 \__unravel_tex_primitive:nnn { box } { make_box } { 0 }
732 \__unravel_tex_primitive:nnn { copy } { make_box } { 1 }
733 \__unravel_tex_primitive:nnn { lastbox } { make_box } { 2 }
734 \__unravel_tex_primitive:nnn { vsplit } { make_box } { 3 }
735 \__unravel_tex_primitive:nnn { vtop } { make_box } { 4 }
736 \__unravel_tex_primitive:nnn { vbox } { make_box } { 5 }
737 \__unravel_tex_primitive:nnn { hbox } { make_box } { 106 }
738 \__unravel_tex_primitive:nnn { moveright } { hmove } { 0 }
739 \__unravel_tex_primitive:nnn { moveleft } { hmove } { 1 }
740 \__unravel_tex_primitive:nnn { lower } { vmove } { 0 }
741 \__unravel_tex_primitive:nnn { raise } { vmove } { 1 }
742 \__unravel_tex_primitive:nnn { unhbox } { un_hbox } { 0 }
743 \__unravel_tex_primitive:nnn { unhcopy } { un_hbox } { 1 }
744 \__unravel_tex_primitive:nnn { unvbox } { un_vbox } { 0 }
745 \__unravel_tex_primitive:nnn { unvcopy } { un_vbox } { 1 }
746 \__unravel_tex_primitive:nnn { pagediscards } { un_vbox } { 2 }
747 \__unravel_tex_primitive:nnn { splitdiscards } { un_vbox } { 3 }
748 \__unravel_tex_primitive:nnn { unpenalty } { remove_item } { 12 }
749 \__unravel_tex_primitive:nnn { unkern } { remove_item } { 11 }
750 \__unravel_tex_primitive:nnn { unskip } { remove_item } { 10 }
751 \__unravel_tex_primitive:nnn { hfil } { hskip } { 0 }
752 \__unravel_tex_primitive:nnn { hfill } { hskip } { 1 }
753 \__unravel_tex_primitive:nnn { hss } { hskip } { 2 }
754 \__unravel_tex_primitive:nnn { hfilneg } { hskip } { 3 }
755 \__unravel_tex_primitive:nnn { hskip } { hskip } { 4 }
756 \__unravel_tex_primitive:nnn { vfil } { vskip } { 0 }

```

```

757 \_unravel_tex_primitive:nnn { vfill } { vskip } { 1 }
758 \_unravel_tex_primitive:nnn { vss } { vskip } { 2 }
759 \_unravel_tex_primitive:nnn { vfilneg } { vskip } { 3 }
760 \_unravel_tex_primitive:nnn { vskip } { vskip } { 4 }
761 \_unravel_tex_primitive:nnn { mskip } { mskip } { 5 }
762 \_unravel_tex_primitive:nnn { kern } { kern } { 1 }
763 \_unravel_tex_primitive:nnn { mkern } { mkern } { 99 }
764 \_unravel_tex_primitive:nnn { shipout } { leader_ship } { 99 }
765 \_unravel_tex_primitive:nnn { leaders } { leader_ship } { 100 }
766 \_unravel_tex_primitive:nnn { cleaders } { leader_ship } { 101 }
767 \_unravel_tex_primitive:nnn { xleaders } { leader_ship } { 102 }
768 \_unravel_tex_primitive:nnn { halign } { halign } { 0 }
769 \_unravel_tex_primitive:nnn { valign } { valign } { 0 }
770 \_unravel_tex_primitive:nnn { beginL } { valign } { 4 }
771 \_unravel_tex_primitive:nnn { endL } { valign } { 5 }
772 \_unravel_tex_primitive:nnn { beginR } { valign } { 8 }
773 \_unravel_tex_primitive:nnn { endR } { valign } { 9 }
774 \_unravel_tex_primitive:nnn { noalign } { no_align } { 0 }
775 \_unravel_tex_primitive:nnn { vrule } { vrule } { 0 }
776 \_unravel_tex_primitive:nnn { hrule } { hrule } { 0 }
777 \_unravel_tex_primitive:nnn { insert } { insert } { 0 }
778 \_unravel_tex_primitive:nnn { vadjust } { vadjust } { 0 }
779 \_unravel_tex_primitive:nnn { ignorespaces } { ignore_spaces } { 0 }
780 \_unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
781 \_unravel_tex_primitive:nnn { aftergroup } { after_group } { 0 }
782 \_unravel_tex_primitive:nnn { penalty } { break_penalty } { 0 }
783 \_unravel_tex_primitive:nnn { indent } { start_par } { 1 }
784 \_unravel_tex_primitive:nnn { noindent } { start_par } { 0 }
785 \_unravel_tex_primitive:nnn { quitvmode } { start_par } { 2 }
786 \_unravel_tex_primitive:nnn { / } { ital_corr } { 0 }
787 \_unravel_tex_primitive:nnn { accent } { accent } { 0 }
788 \_unravel_tex_primitive:nnn { mathaccent } { math_accent } { 0 }
789 \_unravel_tex_primitive:nnn { - } { discretionary } { 1 }
790 \_unravel_tex_primitive:nnn { discretionary } { discretionary } { 0 }
791 \_unravel_tex_primitive:nnn { eqno } { eq_no } { 0 }
792 \_unravel_tex_primitive:nnn { leqno } { eq_no } { 1 }
793 \_unravel_tex_primitive:nnn { left } { left_right } { 30 }
794 \_unravel_tex_primitive:nnn { right } { left_right } { 31 }
795 \_unravel_tex_primitive:nnn { middle } { left_right } { 17 }
796 \_unravel_tex_primitive:nnn { mathord } { math_comp } { 16 }
797 \_unravel_tex_primitive:nnn { mathop } { math_comp } { 17 }
798 \_unravel_tex_primitive:nnn { mathbin } { math_comp } { 18 }
799 \_unravel_tex_primitive:nnn { mathrel } { math_comp } { 19 }
800 \_unravel_tex_primitive:nnn { mathopen } { math_comp } { 20 }
801 \_unravel_tex_primitive:nnn { mathclose } { math_comp } { 21 }
802 \_unravel_tex_primitive:nnn { mathpunct } { math_comp } { 22 }
803 \_unravel_tex_primitive:nnn { mathinner } { math_comp } { 23 }
804 \_unravel_tex_primitive:nnn { underline } { math_comp } { 26 }
805 \_unravel_tex_primitive:nnn { overline } { math_comp } { 27 }
806 \_unravel_tex_primitive:nnn { displaylimits } { limit_switch } { 0 }
807 \_unravel_tex_primitive:nnn { limits } { limit_switch } { 1 }
808 \_unravel_tex_primitive:nnn { nolimits } { limit_switch } { 2 }
809 \_unravel_tex_primitive:nnn { above } { above } { 0 }
810 \_unravel_tex_primitive:nnn { over } { above } { 1 }

```

```

811 \_unravel_tex_primitive:nnn { atop } { above } { 2 }
812 \_unravel_tex_primitive:nnn { abovewithdelims } { above } { 3 }
813 \_unravel_tex_primitive:nnn { overwithdelims } { above } { 4 }
814 \_unravel_tex_primitive:nnn { atopwithdelims } { above } { 5 }
815 \_unravel_tex_primitive:nnn { displaystyle } { math_style } { 0 }
816 \_unravel_tex_primitive:nnn { textstyle } { math_style } { 2 }
817 \_unravel_tex_primitive:nnn { scriptstyle } { math_style } { 4 }
818 \_unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
819 \_unravel_tex_primitive:nnn { mathchoice } { math_choice } { 0 }
820 \_unravel_tex_primitive:nnn { nonscript } { non_script } { 0 }
821 \_unravel_tex_primitive:nnn { vcenter } { vcenter } { 0 }
822 \_unravel_tex_primitive:nnn { lowercase } { case_shift } { 256 }
823 \_unravel_tex_primitive:nnn { uppercase } { case_shift } { 512 }
824 \_unravel_tex_primitive:nnn { message } { message } { 0 }
825 \_unravel_tex_primitive:nnn { errmessage } { message } { 1 }
826 \_unravel_tex_primitive:nnn { openout } { extension } { 0 }
827 \_unravel_tex_primitive:nnn { write } { extension } { 1 }
828 \_unravel_tex_primitive:nnn { closeout } { extension } { 2 }
829 \_unravel_tex_primitive:nnn { special } { extension } { 3 }
830 \_unravel_tex_primitive:nnn { immediate } { extension } { 4 }
831 \_unravel_tex_primitive:nnn { setlanguage } { extension } { 5 }
832 \_unravel_tex_primitive:nnn { pdfliteral } { extension } { 6 }
833 \_unravel_tex_primitive:nnn { pdfobj } { extension } { 7 }
834 \_unravel_tex_primitive:nnn { pdfrefobj } { extension } { 8 }
835 \_unravel_tex_primitive:nnn { pdfxform } { extension } { 9 }
836 \_unravel_tex_primitive:nnn { pdfrefxform } { extension } { 10 }
837 \_unravel_tex_primitive:nnn { pdfximage } { extension } { 11 }
838 \_unravel_tex_primitive:nnn { pdfrefximage } { extension } { 12 }
839 \_unravel_tex_primitive:nnn { pdfannot } { extension } { 13 }
840 \_unravel_tex_primitive:nnn { pdfstartlink } { extension } { 14 }
841 \_unravel_tex_primitive:nnn { pdfendlink } { extension } { 15 }
842 \_unravel_tex_primitive:nnn { pdfoutline } { extension } { 16 }
843 \_unravel_tex_primitive:nnn { pdfdest } { extension } { 17 }
844 \_unravel_tex_primitive:nnn { pdfthread } { extension } { 18 }
845 \_unravel_tex_primitive:nnn { pdfstartthread } { extension } { 19 }
846 \_unravel_tex_primitive:nnn { pdfendthread } { extension } { 20 }
847 \_unravel_tex_primitive:nnn { pdfsavepos } { extension } { 21 }
848 \_unravel_tex_primitive:nnn { pdfinfo } { extension } { 22 }
849 \_unravel_tex_primitive:nnn { pdfcatalog } { extension } { 23 }
850 \_unravel_tex_primitive:nnn { pdfnames } { extension } { 24 }
851 \_unravel_tex_primitive:nnn { pdffontattr } { extension } { 25 }
852 \_unravel_tex_primitive:nnn { pdfincludechars } { extension } { 26 }
853 \_unravel_tex_primitive:nnn { pdfmapfile } { extension } { 27 }
854 \_unravel_tex_primitive:nnn { pdfmapline } { extension } { 28 }
855 \_unravel_tex_primitive:nnn { pdftrailer } { extension } { 29 }
856 \_unravel_tex_primitive:nnn { pdfresettimer } { extension } { 30 }
857 \_unravel_tex_primitive:nnn { pdffontexpand } { extension } { 31 }
858 \_unravel_tex_primitive:nnn { pdfsetrandomseed } { extension } { 32 }
859 \_unravel_tex_primitive:nnn { pdfsnaprefpoint } { extension } { 33 }
860 \_unravel_tex_primitive:nnn { pdfsnapy } { extension } { 34 }
861 \_unravel_tex_primitive:nnn { pdfsnapycomp } { extension } { 35 }
862 \_unravel_tex_primitive:nnn { pdfglyptounicode } { extension } { 36 }
863 \_unravel_tex_primitive:nnn { pdfcolorstack } { extension } { 37 }
864 \_unravel_tex_primitive:nnn { pdfsetmatrix } { extension } { 38 }

```

```

865 \__unravel_tex_primitive:nnn { pdfsave } { extension } { 39 }
866 \__unravel_tex_primitive:nnn { pdfrestore } { extension } { 40 }
867 \__unravel_tex_primitive:nnn { pdfnoblur } { extension } { 41 }
868 \__unravel_tex_primitive:nnn { openin } { in_stream } { 1 }
869 \__unravel_tex_primitive:nnn { closein } { in_stream } { 0 }
870 \__unravel_tex_primitive:nnn { begingroup } { begin_group } { 0 }
871 \__unravel_tex_primitive:nnn { endgroup } { end_group } { 0 }
872 \__unravel_tex_primitive:nnn { omit } { omit } { 0 }
873 \__unravel_tex_primitive:nnn { ~ } { ex_space } { 0 }
874 \__unravel_tex_primitive:nnn { noboundary } { no_boundary } { 0 }
875 \__unravel_tex_primitive:nnn { radical } { radical } { 0 }
876 \__unravel_tex_primitive:nnn { endcsname } { end_cs_name } { 0 }
877 \__unravel_tex_primitive:nnn { lastpenalty } { last_item } { 0 }
878 \__unravel_tex_primitive:nnn { lastkern } { last_item } { 1 }
879 \__unravel_tex_primitive:nnn { lastskip } { last_item } { 2 }
880 \__unravel_tex_primitive:nnn { lastnodetype } { last_item } { 3 }
881 \__unravel_tex_primitive:nnn { inputlineno } { last_item } { 4 }
882 \__unravel_tex_primitive:nnn { badness } { last_item } { 5 }
883 \__unravel_tex_primitive:nnn { pdftexversion } { last_item } { 6 }
884 \__unravel_tex_primitive:nnn { pdflastobj } { last_item } { 7 }
885 \__unravel_tex_primitive:nnn { pdflastxform } { last_item } { 8 }
886 \__unravel_tex_primitive:nnn { pdflastximage } { last_item } { 9 }
887 \__unravel_tex_primitive:nnn { pdflastximagepages } { last_item } { 10 }
888 \__unravel_tex_primitive:nnn { pdflastannot } { last_item } { 11 }
889 \__unravel_tex_primitive:nnn { pdflastxpos } { last_item } { 12 }
890 \__unravel_tex_primitive:nnn { pdflastypos } { last_item } { 13 }
891 \__unravel_tex_primitive:nnn { pdfretval } { last_item } { 14 }
892 \__unravel_tex_primitive:nnn { pdflastximagecolordepth } { last_item } { 15 }
893 \__unravel_tex_primitive:nnn { pdfelapsedtime } { last_item } { 16 }
894 \__unravel_tex_primitive:nnn { pdfshellescape } { last_item } { 17 }
895 \__unravel_tex_primitive:nnn { pdfrandomseed } { last_item } { 18 }
896 \__unravel_tex_primitive:nnn { pdflastlink } { last_item } { 19 }
897 \__unravel_tex_primitive:nnn { eTeXversion } { last_item } { 20 }
898 \__unravel_tex_primitive:nnn { currentgrouplevel } { last_item } { 21 }
899 \__unravel_tex_primitive:nnn { currentgrouptype } { last_item } { 22 }
900 \__unravel_tex_primitive:nnn { currentiflevel } { last_item } { 23 }
901 \__unravel_tex_primitive:nnn { currentifttype } { last_item } { 24 }
902 \__unravel_tex_primitive:nnn { currentifbranch } { last_item } { 25 }
903 \__unravel_tex_primitive:nnn { gluestretchorder } { last_item } { 26 }
904 \__unravel_tex_primitive:nnn { glueshrinkorder } { last_item } { 27 }
905 \__unravel_tex_primitive:nnn { fontcharwd } { last_item } { 28 }
906 \__unravel_tex_primitive:nnn { fontcharht } { last_item } { 29 }
907 \__unravel_tex_primitive:nnn { fontchardp } { last_item } { 30 }
908 \__unravel_tex_primitive:nnn { fontcharic } { last_item } { 31 }
909 \__unravel_tex_primitive:nnn { parshapelength } { last_item } { 32 }
910 \__unravel_tex_primitive:nnn { parshapeindent } { last_item } { 33 }
911 \__unravel_tex_primitive:nnn { parshapedimen } { last_item } { 34 }
912 \__unravel_tex_primitive:nnn { gluestretch } { last_item } { 35 }
913 \__unravel_tex_primitive:nnn { glueshrink } { last_item } { 36 }
914 \__unravel_tex_primitive:nnn { mutoglu } { last_item } { 37 }
915 \__unravel_tex_primitive:nnn { gluetomu } { last_item } { 38 }
916 \__unravel_tex_primitive:nnn { numexpr } { last_item } { 39 }
917 \__unravel_tex_primitive:nnn { dimexpr } { last_item } { 40 }
918 \__unravel_tex_primitive:nnn { glueexpr } { last_item } { 41 }

```

```

919 \_unravel_tex_primitive:nnn { muexpr          } { last_item } { 42 }
920 \_unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
921 \_unravel_tex_primitive:nnn { output          } { assign_toks } { 1 }
922 \_unravel_tex_primitive:nnn { everypar       } { assign_toks } { 2 }
923 \_unravel_tex_primitive:nnn { everymath      } { assign_toks } { 3 }
924 \_unravel_tex_primitive:nnn { everydisplay  } { assign_toks } { 4 }
925 \_unravel_tex_primitive:nnn { everyhbox    } { assign_toks } { 5 }
926 \_unravel_tex_primitive:nnn { everyvbox    } { assign_toks } { 6 }
927 \_unravel_tex_primitive:nnn { everyjob     } { assign_toks } { 7 }
928 \_unravel_tex_primitive:nnn { everycr      } { assign_toks } { 8 }
929 \_unravel_tex_primitive:nnn { errhelp       } { assign_toks } { 9 }
930 \_unravel_tex_primitive:nnn { pdfpagesattr  } { assign_toks } { 10 }
931 \_unravel_tex_primitive:nnn { pdfpageattr   } { assign_toks } { 11 }
932 \_unravel_tex_primitive:nnn { pdfpageresources } { assign_toks } { 12 }
933 \_unravel_tex_primitive:nnn { pdfpkmode     } { assign_toks } { 13 }
934 \_unravel_tex_primitive:nnn { everyeof      } { assign_toks } { 14 }
935 \_unravel_tex_primitive:nnn { pretolerance  } { assign_int } { 0 }
936 \_unravel_tex_primitive:nnn { tolerance    } { assign_int } { 1 }
937 \_unravel_tex_primitive:nnn { linepenalty  } { assign_int } { 2 }
938 \_unravel_tex_primitive:nnn { hyphenpenalty } { assign_int } { 3 }
939 \_unravel_tex_primitive:nnn { exhyphenpenalty } { assign_int } { 4 }
940 \_unravel_tex_primitive:nnn { clubpenalty  } { assign_int } { 5 }
941 \_unravel_tex_primitive:nnn { widowpenalty } { assign_int } { 6 }
942 \_unravel_tex_primitive:nnn { displaywidowpenalty } { assign_int } { 7 }
943 \_unravel_tex_primitive:nnn { brokenpenalty } { assign_int } { 8 }
944 \_unravel_tex_primitive:nnn { binoppenalty } { assign_int } { 9 }
945 \_unravel_tex_primitive:nnn { relpenalty   } { assign_int } { 10 }
946 \_unravel_tex_primitive:nnn { predisplaypenalty } { assign_int } { 11 }
947 \_unravel_tex_primitive:nnn { postdisplaypenalty } { assign_int } { 12 }
948 \_unravel_tex_primitive:nnn { interlinepenalty } { assign_int } { 13 }
949 \_unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int } { 14 }
950 \_unravel_tex_primitive:nnn { finalhyphendemerits } { assign_int } { 15 }
951 \_unravel_tex_primitive:nnn { adjdemerits   } { assign_int } { 16 }
952 \_unravel_tex_primitive:nnn { mag           } { assign_int } { 17 }
953 \_unravel_tex_primitive:nnn { delimiterfactor } { assign_int } { 18 }
954 \_unravel_tex_primitive:nnn { looseness     } { assign_int } { 19 }
955 \_unravel_tex_primitive:nnn { time          } { assign_int } { 20 }
956 \_unravel_tex_primitive:nnn { day           } { assign_int } { 21 }
957 \_unravel_tex_primitive:nnn { month         } { assign_int } { 22 }
958 \_unravel_tex_primitive:nnn { year          } { assign_int } { 23 }
959 \_unravel_tex_primitive:nnn { showboxbreadth } { assign_int } { 24 }
960 \_unravel_tex_primitive:nnn { showboxdepth   } { assign_int } { 25 }
961 \_unravel_tex_primitive:nnn { hbadness       } { assign_int } { 26 }
962 \_unravel_tex_primitive:nnn { vbadness       } { assign_int } { 27 }
963 \_unravel_tex_primitive:nnn { pausing        } { assign_int } { 28 }
964 \_unravel_tex_primitive:nnn { tracingonline   } { assign_int } { 29 }
965 \_unravel_tex_primitive:nnn { tracingmacros   } { assign_int } { 30 }
966 \_unravel_tex_primitive:nnn { tracingstats    } { assign_int } { 31 }
967 \_unravel_tex_primitive:nnn { tracingparagraphs } { assign_int } { 32 }
968 \_unravel_tex_primitive:nnn { tracingpages    } { assign_int } { 33 }
969 \_unravel_tex_primitive:nnn { tracingoutput   } { assign_int } { 34 }
970 \_unravel_tex_primitive:nnn { tracinglostchars } { assign_int } { 35 }
971 \_unravel_tex_primitive:nnn { tracingcommands } { assign_int } { 36 }
972 \_unravel_tex_primitive:nnn { tracingrestores } { assign_int } { 37 }

```

```

973 \_unravel\_tex\_primitive:nnn { uchypb } { assign\_int } { 38 }
974 \_unravel\_tex\_primitive:nnn { outputpenalty } { assign\_int } { 39 }
975 \_unravel\_tex\_primitive:nnn { maxdeadcycles } { assign\_int } { 40 }
976 \_unravel\_tex\_primitive:nnn { hangafter } { assign\_int } { 41 }
977 \_unravel\_tex\_primitive:nnn { floatingpenalty } { assign\_int } { 42 }
978 \_unravel\_tex\_primitive:nnn { globaldefs } { assign\_int } { 43 }
979 \_unravel\_tex\_primitive:nnn { fam } { assign\_int } { 44 }
980 \_unravel\_tex\_primitive:nnn { escapechar } { assign\_int } { 45 }
981 \_unravel\_tex\_primitive:nnn { defaultthyphenchar } { assign\_int } { 46 }
982 \_unravel\_tex\_primitive:nnn { defaultskewchar } { assign\_int } { 47 }
983 \_unravel\_tex\_primitive:nnn { endlinesechar } { assign\_int } { 48 }
984 \_unravel\_tex\_primitive:nnn { newlinesechar } { assign\_int } { 49 }
985 \_unravel\_tex\_primitive:nnn { language } { assign\_int } { 50 }
986 \_unravel\_tex\_primitive:nnn { lefthyphenmin } { assign\_int } { 51 }
987 \_unravel\_tex\_primitive:nnn { righthyphenmin } { assign\_int } { 52 }
988 \_unravel\_tex\_primitive:nnn { holdinginserts } { assign\_int } { 53 }
989 \_unravel\_tex\_primitive:nnn { errorcontextlines } { assign\_int } { 54 }
990 \_unravel\_tex\_primitive:nnn { pdfoutput } { assign\_int } { 55 }
991 \_unravel\_tex\_primitive:nnn { pdfcompresslevel } { assign\_int } { 56 }
992 \_unravel\_tex\_primitive:nnn { pdfdecimaldigits } { assign\_int } { 57 }
993 \_unravel\_tex\_primitive:nnn { pdfmovechars } { assign\_int } { 58 }
994 \_unravel\_tex\_primitive:nnn { pdfimageresolution } { assign\_int } { 59 }
995 \_unravel\_tex\_primitive:nnn { pdfpkresolution } { assign\_int } { 60 }
996 \_unravel\_tex\_primitive:nnn { pdfuniqueresname } { assign\_int } { 61 }
997 \_unravel\_tex\_primitive:nnn
998 { pdfoptionalwaysusepdfpagebox } { assign\_int } { 62 }
999 \_unravel\_tex\_primitive:nnn
1000 { pdfoptionpdfinclusionerrorlevel } { assign\_int } { 63 }
1001 \_unravel\_tex\_primitive:nnn
1002 { pdfoptionpdfminorversion } { assign\_int } { 64 }
1003 \_unravel\_tex\_primitive:nnn { pdfminorversion } { assign\_int } { 64 }
1004 \_unravel\_tex\_primitive:nnn { pdfforcepagebox } { assign\_int } { 65 }
1005 \_unravel\_tex\_primitive:nnn { pdfpagebox } { assign\_int } { 66 }
1006 \_unravel\_tex\_primitive:nnn
1007 { pdfinclusionerrorlevel } { assign\_int } { 67 }
1008 \_unravel\_tex\_primitive:nnn { pdfgamma } { assign\_int } { 68 }
1009 \_unravel\_tex\_primitive:nnn { pdfimagegamma } { assign\_int } { 69 }
1010 \_unravel\_tex\_primitive:nnn { pdfimagehicolor } { assign\_int } { 70 }
1011 \_unravel\_tex\_primitive:nnn { pdfimageapplygamma } { assign\_int } { 71 }
1012 \_unravel\_tex\_primitive:nnn { pdfadjustspacing } { assign\_int } { 72 }
1013 \_unravel\_tex\_primitive:nnn { pdfprotrudechars } { assign\_int } { 73 }
1014 \_unravel\_tex\_primitive:nnn { pdftracingfonts } { assign\_int } { 74 }
1015 \_unravel\_tex\_primitive:nnn { pdfobjcompresslevel } { assign\_int } { 75 }
1016 \_unravel\_tex\_primitive:nnn
1017 { pdfadjustinterwordglue } { assign\_int } { 76 }
1018 \_unravel\_tex\_primitive:nnn { pdfprependkern } { assign\_int } { 77 }
1019 \_unravel\_tex\_primitive:nnn { pdfappendkern } { assign\_int } { 78 }
1020 \_unravel\_tex\_primitive:nnn { pdfgentounicode } { assign\_int } { 79 }
1021 \_unravel\_tex\_primitive:nnn { pdfdraftmode } { assign\_int } { 80 }
1022 \_unravel\_tex\_primitive:nnn { pdfinclusioncopyfonts } { assign\_int } { 81 }
1023 \_unravel\_tex\_primitive:nnn { tracingassigns } { assign\_int } { 82 }
1024 \_unravel\_tex\_primitive:nnn { tracinggroups } { assign\_int } { 83 }
1025 \_unravel\_tex\_primitive:nnn { tracingifs } { assign\_int } { 84 }
1026 \_unravel\_tex\_primitive:nnn { tracingscantokens } { assign\_int } { 85 }

```

```

1027 \_unravel_tex_primitive:nnn { tracingnesting } { assign_int } { 86 }
1028 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
1029 \_unravel_tex_primitive:nnn { lastlinefit } { assign_int } { 88 }
1030 \_unravel_tex_primitive:nnn { savingdiscards } { assign_int } { 89 }
1031 \_unravel_tex_primitive:nnn { savinghyphcodes } { assign_int } { 90 }
1032 \_unravel_tex_primitive:nnn { TeXxetstate } { assign_int } { 91 }
1033 \_unravel_tex_primitive:nnn { parindent } { assign_dimen } { 0 }
1034 \_unravel_tex_primitive:nnn { mathsurround } { assign_dimen } { 1 }
1035 \_unravel_tex_primitive:nnn { lineskiplimit } { assign_dimen } { 2 }
1036 \_unravel_tex_primitive:nnn { hsize } { assign_dimen } { 3 }
1037 \_unravel_tex_primitive:nnn { vsize } { assign_dimen } { 4 }
1038 \_unravel_tex_primitive:nnn { maxdepth } { assign_dimen } { 5 }
1039 \_unravel_tex_primitive:nnn { splitmaxdepth } { assign_dimen } { 6 }
1040 \_unravel_tex_primitive:nnn { boxmaxdepth } { assign_dimen } { 7 }
1041 \_unravel_tex_primitive:nnn { hfuzz } { assign_dimen } { 8 }
1042 \_unravel_tex_primitive:nnn { vfuzz } { assign_dimen } { 9 }
1043 \_unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
1044 \_unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
1045 \_unravel_tex_primitive:nnn { scriptspace } { assign_dimen } { 12 }
1046 \_unravel_tex_primitive:nnn { predplaysize } { assign_dimen } { 13 }
1047 \_unravel_tex_primitive:nnn { displaywidth } { assign_dimen } { 14 }
1048 \_unravel_tex_primitive:nnn { displayindent } { assign_dimen } { 15 }
1049 \_unravel_tex_primitive:nnn { overfullrule } { assign_dimen } { 16 }
1050 \_unravel_tex_primitive:nnn { hangindent } { assign_dimen } { 17 }
1051 \_unravel_tex_primitive:nnn { hoffset } { assign_dimen } { 18 }
1052 \_unravel_tex_primitive:nnn { voffset } { assign_dimen } { 19 }
1053 \_unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
1054 \_unravel_tex_primitive:nnn { pdfhorigin } { assign_dimen } { 21 }
1055 \_unravel_tex_primitive:nnn { pdfvorigin } { assign_dimen } { 22 }
1056 \_unravel_tex_primitive:nnn { pdfpagewidth } { assign_dimen } { 23 }
1057 \_unravel_tex_primitive:nnn { pdfpageheight } { assign_dimen } { 24 }
1058 \_unravel_tex_primitive:nnn { pdflinkmargin } { assign_dimen } { 25 }
1059 \_unravel_tex_primitive:nnn { pdfdestmargin } { assign_dimen } { 26 }
1060 \_unravel_tex_primitive:nnn { pdfthreadmargin } { assign_dimen } { 27 }
1061 \_unravel_tex_primitive:nnn { pdffirstlineheight } { assign_dimen } { 28 }
1062 \_unravel_tex_primitive:nnn { pdfastlinedepth } { assign_dimen } { 29 }
1063 \_unravel_tex_primitive:nnn { pdfeachlineheight } { assign_dimen } { 30 }
1064 \_unravel_tex_primitive:nnn { pdfeachlinedepth } { assign_dimen } { 31 }
1065 \_unravel_tex_primitive:nnn { pdfignoredimen } { assign_dimen } { 32 }
1066 \_unravel_tex_primitive:nnn { pdfpxdimen } { assign_dimen } { 33 }
1067 \_unravel_tex_primitive:nnn { lineskip } { assign_glue } { 0 }
1068 \_unravel_tex_primitive:nnn { baselineskip } { assign_glue } { 1 }
1069 \_unravel_tex_primitive:nnn { parskip } { assign_glue } { 2 }
1070 \_unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue } { 3 }
1071 \_unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue } { 4 }
1072 \_unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
1073 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1074 \_unravel_tex_primitive:nnn { leftskip } { assign_glue } { 7 }
1075 \_unravel_tex_primitive:nnn { rightskip } { assign_glue } { 8 }
1076 \_unravel_tex_primitive:nnn { topskip } { assign_glue } { 9 }
1077 \_unravel_tex_primitive:nnn { splittopskip } { assign_glue } { 10 }
1078 \_unravel_tex_primitive:nnn { tabskip } { assign_glue } { 11 }
1079 \_unravel_tex_primitive:nnn { spaceskip } { assign_glue } { 12 }
1080 \_unravel_tex_primitive:nnn { xspaceskip } { assign_glue } { 13 }

```

```

1081 \_unravel_tex_primitive:nnn { parfillskip          } { assign_glue } { 14 }
1082 \_unravel_tex_primitive:nnn { thinmuskip          } { assign_mu_glue } { 15 }
1083 \_unravel_tex_primitive:nnn { medmuskip           } { assign_mu_glue } { 16 }
1084 \_unravel_tex_primitive:nnn { thickmuskip        } { assign_mu_glue } { 17 }
1085 \_unravel_tex_primitive:nnn { fontdimen         } { assign_font_dimen } { 0 }
1086 \_unravel_tex_primitive:nnn { hyphenchar       } { assign_font_int } { 0 }
1087 \_unravel_tex_primitive:nnn { skewchar          } { assign_font_int } { 1 }
1088 \_unravel_tex_primitive:nnn { lpcode             } { assign_font_int } { 2 }
1089 \_unravel_tex_primitive:nnn { rpcode             } { assign_font_int } { 3 }
1090 \_unravel_tex_primitive:nnn { efcodes             } { assign_font_int } { 4 }
1091 \_unravel_tex_primitive:nnn { tagcode            } { assign_font_int } { 5 }
1092 \_unravel_tex_primitive:nnn { pdfnoligatures     } { assign_font_int } { 6 }
1093 \_unravel_tex_primitive:nnn { knbscode           } { assign_font_int } { 7 }
1094 \_unravel_tex_primitive:nnn { stbscode           } { assign_font_int } { 8 }
1095 \_unravel_tex_primitive:nnn { shbscode           } { assign_font_int } { 9 }
1096 \_unravel_tex_primitive:nnn { knbccode          } { assign_font_int } { 10 }
1097 \_unravel_tex_primitive:nnn { knaccode          } { assign_font_int } { 11 }
1098 \_unravel_tex_primitive:nnn { spacefactor       } { set_aux } { 102 }
1099 \_unravel_tex_primitive:nnn { prevdepth        } { set_aux } { 1 }
1100 \_unravel_tex_primitive:nnn { prevgraf          } { set_prev_graf } { 0 }
1101 \_unravel_tex_primitive:nnn { pagegoal         } { set_page_dimen } { 0 }
1102 \_unravel_tex_primitive:nnn { pagetotal        } { set_page_dimen } { 1 }
1103 \_unravel_tex_primitive:nnn { pagestretch       } { set_page_dimen } { 2 }
1104 \_unravel_tex_primitive:nnn { pagefilstretch    } { set_page_dimen } { 3 }
1105 \_unravel_tex_primitive:nnn { pagefillstretch    } { set_page_dimen } { 4 }
1106 \_unravel_tex_primitive:nnn { pagefilllstretch  } { set_page_dimen } { 5 }
1107 \_unravel_tex_primitive:nnn { pageshrink        } { set_page_dimen } { 6 }
1108 \_unravel_tex_primitive:nnn { pagedepth         } { set_page_dimen } { 7 }
1109 \_unravel_tex_primitive:nnn { deadcycles        } { set_page_int } { 0 }
1110 \_unravel_tex_primitive:nnn { insertpenalties   } { set_page_int } { 1 }
1111 \_unravel_tex_primitive:nnn { interactionmode    } { set_page_int } { 2 }
1112 \_unravel_tex_primitive:nnn { wd                } { set_box_dimen } { 1 }
1113 \_unravel_tex_primitive:nnn { dp                } { set_box_dimen } { 2 }
1114 \_unravel_tex_primitive:nnn { ht                } { set_box_dimen } { 3 }
1115 \_unravel_tex_primitive:nnn { parshape          } { set_shape } { 0 }
1116 \_unravel_tex_primitive:nnn { interlinepenalties } { set_shape } { 1 }
1117 \_unravel_tex_primitive:nnn { clubpenalties     } { set_shape } { 2 }
1118 \_unravel_tex_primitive:nnn { widowpenalties    } { set_shape } { 3 }
1119 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
1120 \_unravel_tex_primitive:nnn { catcode           } { def_code } { 0 }
1121 \_unravel_tex_primitive:nnn { lccode            } { def_code } { 256 }
1122 \_unravel_tex_primitive:nnn { uccode            } { def_code } { 512 }
1123 \_unravel_tex_primitive:nnn { sfcode            } { def_code } { 768 }
1124 \_unravel_tex_primitive:nnn { mathcode         } { def_code } { 1024 }
1125 \_unravel_tex_primitive:nnn { delcode           } { def_code } { 1591 }
1126 \_unravel_tex_primitive:nnn { textfont         } { def_family } { -48 }
1127 \_unravel_tex_primitive:nnn { scriptfont        } { def_family } { -32 }
1128 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family } { -16 }
1129 \_unravel_tex_primitive:nnn { nullfont         } { set_font } { 0 }
1130 \_unravel_tex_primitive:nnn { font             } { def_font } { 0 }
1131 \_unravel_tex_primitive:nnn { count            } { register } { 1 000 000 }
1132 \_unravel_tex_primitive:nnn { dimen              } { register } { 2 000 000 }
1133 \_unravel_tex_primitive:nnn { skip              } { register } { 3 000 000 }
1134 \_unravel_tex_primitive:nnn { muskip             } { register } { 4 000 000 }

```

```

1135 \__unravel_tex_primitive:nnn { advance } { advance } { 0 }
1136 \__unravel_tex_primitive:nnn { multiply } { multiply } { 0 }
1137 \__unravel_tex_primitive:nnn { divide } { divide } { 0 }
1138 \__unravel_tex_primitive:nnn { long } { prefix } { 1 }
1139 \__unravel_tex_primitive:nnn { outer } { prefix } { 2 }
1140 \__unravel_tex_primitive:nnn { global } { prefix } { 4 }
1141 \__unravel_tex_primitive:nnn { protected } { prefix } { 8 }
1142 \__unravel_tex_primitive:nnn { let } { let } { 0 }
1143 \__unravel_tex_primitive:nnn { futurelet } { let } { 1 }
1144 \__unravel_tex_primitive:nnn { chardef } { shorthand_def } { 0 }
1145 \__unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
1146 \__unravel_tex_primitive:nnn { countdef } { shorthand_def } { 2 }
1147 \__unravel_tex_primitive:nnn { dimendef } { shorthand_def } { 3 }
1148 \__unravel_tex_primitive:nnn { skipdef } { shorthand_def } { 4 }
1149 \__unravel_tex_primitive:nnn { muskipdef } { shorthand_def } { 5 }
1150 \__unravel_tex_primitive:nnn { toksdef } { shorthand_def } { 6 }
1151 \__unravel_tex_primitive:nnn { read } { read_to_cs } { 0 }
1152 \__unravel_tex_primitive:nnn { readline } { read_to_cs } { 1 }
1153 \__unravel_tex_primitive:nnn { def } { def } { 0 }
1154 \__unravel_tex_primitive:nnn { gdef } { def } { 1 }
1155 \__unravel_tex_primitive:nnn { edef } { def } { 2 }
1156 \__unravel_tex_primitive:nnn { xdef } { def } { 3 }
1157 \__unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1158 \__unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1159 \__unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1160 \__unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1161 \__unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1162 \__unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1163 \__unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1164 \__unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1165 \__unravel_tex_primitive:nnn { pdfcopyfont } { pdf_copy_font } { 0 }
1166 \__unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1167 \__unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1168 \__unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1169 \__unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1170 \__unravel_tex_primitive:nnn { pdfprimitive } { no_expand } { 1 }
1171 \__unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1172 \__unravel_tex_primitive:nnn { input } { input } { 0 }
1173 \__unravel_tex_primitive:nnn { endinput } { input } { 1 }
1174 \__unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1175 \__unravel_tex_primitive:nnn { if } { if_test } { 0 }
1176 \__unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }
1177 \__unravel_tex_primitive:nnn { ifnum } { if_test } { 2 }
1178 \__unravel_tex_primitive:nnn { ifdim } { if_test } { 3 }
1179 \__unravel_tex_primitive:nnn { ifodd } { if_test } { 4 }
1180 \__unravel_tex_primitive:nnn { ifvmode } { if_test } { 5 }
1181 \__unravel_tex_primitive:nnn { ifhmode } { if_test } { 6 }
1182 \__unravel_tex_primitive:nnn { ifmmode } { if_test } { 7 }
1183 \__unravel_tex_primitive:nnn { ifinner } { if_test } { 8 }
1184 \__unravel_tex_primitive:nnn { ifvoid } { if_test } { 9 }
1185 \__unravel_tex_primitive:nnn { ifhbox } { if_test } { 10 }
1186 \__unravel_tex_primitive:nnn { ifvbox } { if_test } { 11 }
1187 \__unravel_tex_primitive:nnn { ifx } { if_test } { 12 }
1188 \__unravel_tex_primitive:nnn { ifeof } { if_test } { 13 }

```

```

1189 \__unravel_tex_primitive:nnn { iftrue } { if_test } { 14 }
1190 \__unravel_tex_primitive:nnn { iffalse } { if_test } { 15 }
1191 \__unravel_tex_primitive:nnn { ifcase } { if_test } { 16 }
1192 \__unravel_tex_primitive:nnn { ifdefined } { if_test } { 17 }
1193 \__unravel_tex_primitive:nnn { ifcsname } { if_test } { 18 }
1194 \__unravel_tex_primitive:nnn { iffontchar } { if_test } { 19 }
1195 \__unravel_tex_primitive:nnn { ifincsname } { if_test } { 20 }
1196 \__unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1197 \__unravel_tex_primitive:nnn { ifpdfabsnum } { if_test } { 22 }
1198 \__unravel_tex_primitive:nnn { ifpdfabsdim } { if_test } { 23 }
1199 \__unravel_tex_primitive:nnn { fi } { fi_or_else } { 2 }
1200 \__unravel_tex_primitive:nnn { else } { fi_or_else } { 3 }
1201 \__unravel_tex_primitive:nnn { or } { fi_or_else } { 4 }
1202 \__unravel_tex_primitive:nnn { csname } { cs_name } { 0 }
1203 \__unravel_tex_primitive:nnn { number } { convert } { 0 }
1204 \__unravel_tex_primitive:nnn { romannumeral } { convert } { 1 }
1205 \__unravel_tex_primitive:nnn { string } { convert } { 2 }
1206 \__unravel_tex_primitive:nnn { meaning } { convert } { 3 }
1207 \__unravel_tex_primitive:nnn { fontname } { convert } { 4 }
1208 \__unravel_tex_primitive:nnn { eTeXrevision } { convert } { 5 }
1209 \__unravel_tex_primitive:nnn { pdftexrevision } { convert } { 6 }
1210 \__unravel_tex_primitive:nnn { pdftexbanner } { convert } { 7 }
1211 \__unravel_tex_primitive:nnn { pdffontname } { convert } { 8 }
1212 \__unravel_tex_primitive:nnn { pdffontobjnum } { convert } { 9 }
1213 \__unravel_tex_primitive:nnn { pdffontsize } { convert } { 10 }
1214 \__unravel_tex_primitive:nnn { pdfpageref } { convert } { 11 }
1215 \__unravel_tex_primitive:nnn { pdfxformname } { convert } { 12 }
1216 \__unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
1217 \__unravel_tex_primitive:nnn { pdfescapename } { convert } { 14 }
1218 \__unravel_tex_primitive:nnn { leftmarginkern } { convert } { 15 }
1219 \__unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
1220 \__unravel_tex_primitive:nnn
1221 { \sys_if_engine_xetex:F { pdf } strcmp } { convert } { 17 }
1222 \__unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
1223 \__unravel_tex_primitive:nnn { pdfescapehex } { convert } { 19 }
1224 \__unravel_tex_primitive:nnn { pdfunescapehex } { convert } { 20 }
1225 \__unravel_tex_primitive:nnn { pdfcreationdate } { convert } { 21 }
1226 \__unravel_tex_primitive:nnn { pdffilemoddate } { convert } { 22 }
1227 \__unravel_tex_primitive:nnn { pdffilesize } { convert } { 23 }
1228 \__unravel_tex_primitive:nnn { pdfmdfivesum } { convert } { 24 }
1229 \__unravel_tex_primitive:nnn { pdffiledump } { convert } { 25 }
1230 \__unravel_tex_primitive:nnn { pdfmatch } { convert } { 26 }
1231 \__unravel_tex_primitive:nnn { pdflastmatch } { convert } { 27 }
1232 \__unravel_tex_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
1233 \__unravel_tex_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
1234 \__unravel_tex_primitive:nnn { pdfinserttht } { convert } { 30 }
1235 \__unravel_tex_primitive:nnn { pdfximagebbox } { convert } { 31 }
1236 \__unravel_tex_primitive:nnn { jobname } { convert } { 32 }
1237 \sys_if_engine_luatex:T
1238 {
1239 \__unravel_tex_primitive:nnn { directlua } { convert } { 33 }
1240 \__unravel_tex_primitive:nnn { expanded } { convert } { 34 }
1241 \__unravel_tex_primitive:nnn { luaescapestring } { convert } { 35 }
1242 }

```

```

1243 \sys_if_engine_xetex:T
1244 {
1245   \__unravel_tex_primitive:nnn { Ucharcat          } { convert } { 40 }
1246 }
1247 \__unravel_tex_primitive:nnn { the                  } { the } { 0 }
1248 \__unravel_tex_primitive:nnn { unexpanded          } { the } { 1 }
1249 \__unravel_tex_primitive:nnn { detokenize          } { the } { 5 }
1250 \__unravel_tex_primitive:nnn { topmark              } { top_bot_mark } { 0 }
1251 \__unravel_tex_primitive:nnn { firstmark            } { top_bot_mark } { 1 }
1252 \__unravel_tex_primitive:nnn { botmark              } { top_bot_mark } { 2 }
1253 \__unravel_tex_primitive:nnn { splitfirstmark       } { top_bot_mark } { 3 }
1254 \__unravel_tex_primitive:nnn { splitbotmark         } { top_bot_mark } { 4 }
1255 \__unravel_tex_primitive:nnn { topmarks             } { top_bot_mark } { 5 }
1256 \__unravel_tex_primitive:nnn { firstmarks           } { top_bot_mark } { 6 }
1257 \__unravel_tex_primitive:nnn { botmarks             } { top_bot_mark } { 7 }
1258 \__unravel_tex_primitive:nnn { splitfirstmarks      } { top_bot_mark } { 8 }
1259 \__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`: If the input is empty, forcefully exit. Otherwise, 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.

```

1260 \cs_new_protected:Npn \__unravel_get_next:
1261 {
1262   \__unravel_input_if_empty:TF
1263   { \__unravel_exit:w }
1264   {
1265     \__unravel_input_gpop:N \l__unravel_head_gtl
1266     \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1267     \gtl_if_tl:NNTF \l__unravel_head_gtl
1268     {
1269       \tl_set:Nx \l__unravel_head_tl
1270       { \gtl_head:N \l__unravel_head_gtl }
1271       \token_if_eq_meaning:NNT
1272       \l__unravel_head_token \__unravel_special_relax:
1273       \__unravel_get_next_notexpanded:
1274     }
1275     { \tl_clear:N \l__unravel_head_tl }
1276   }
1277 }
1278 \cs_new_protected:Npn \__unravel_get_next_aux:w
1279 { \cs_set_eq:NN \l__unravel_head_token }

```

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

```

\_unravel_get_next_notexpanded: At this point we have likely encountered a special \relax marker that we use to mark
\_unravel_notexpanded_test:w cases where \noexpand acts on a control sequence or an active character. To make sure
\_unravel_notexpanded_expand:nN of that check the control sequence has the form \notexpanded:.... Since we don't
\_unravel_notexpanded_expand:NN 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.

1280 \cs_set_protected:Npn \_unravel_tmp:w #1
1281 {
1282   \cs_new_protected:Npn \_unravel_get_next_notexpanded:
1283   {
1284     \tl_if_eq:onTF { \l__unravel_head_tl } { \_unravel_unravel_marker: }
1285     { \_unravel_get_next_marker: }
1286     {
1287       \_unravel_exp_args:NNx \use:nn \_unravel_notexpanded_test:w
1288       { \scan_stop: \exp_after:wN \cs_to_str:N \l__unravel_head_tl Z }
1289       \q_mark \_unravel_notexpanded_expand:n
1290       #1 Z \q_mark \use_none:n
1291       \q_stop
1292     }
1293   }
1294   \cs_new_protected:Npn \_unravel_notexpanded_test:w
1295   ##1 #1 ##2 Z \q_mark ##3##4 \q_stop
1296   { ##3 {##2} }
1297 }
1298 \exp_args:Nx \_unravel_tmp:w { \scan_stop: \tl_to_str:n { notexpanded: } }
1299 \group_begin:
1300 \char_set_catcode_active:n { 0 }
1301 \cs_new_protected:Npn \_unravel_notexpanded_expand:n #1
1302 {
1303   \_unravel_exp_args:Nx \tl_if_empty:nTF { \str_tail:n {#1} }
1304   {
1305     \group_begin:
1306     \char_set_lccode:nn { 0 } { '#1 }
1307     \tex_lowercase:D
1308     {
1309       \group_end:
1310       \_unravel_notexpanded_expand:N ^^@
1311     }
1312   }
1313   {
1314     \group_begin: \exp_args:NNc \group_end:
1315     \_unravel_notexpanded_expand:N { \use_none:n #1 }
1316   }
1317 }
1318 \group_end:
1319 \cs_new_protected:Npn \_unravel_notexpanded_expand:N #1

```

```

1320 {
1321   \gtl_set:Nn \l__unravel_head_gtl {#1}
1322   \tl_set:Nn \l__unravel_head_tl {#1}
1323   \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
1324 }

```

(End definition for __unravel_get_next_notexpanded: and others.)

__unravel_get_next_marker: This is used to deal with nested unravel.

```

1325 \cs_new_protected:Npn \__unravel_get_next_marker:
1326 {
1327   \__unravel_get_next:
1328   \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
1329     { \__unravel_error:nxxxx { nested-unravel } { } { } { } { } }
1330     { \__unravel_error:nxxxx { internal } { marker~unknown } { } { } { } { } }
1331   \__unravel_input_gpop_item:NF \l__unravel_argi_tl
1332     { \__unravel_error:nxxxx { internal } { marker~1 } { } { } { } { } }
1333   \__unravel_input_gpop_item:NF \l__unravel_argii_tl
1334     { \__unravel_error:nxxxx { internal } { marker~2 } { } { } { } { } }
1335   \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
1336   \__unravel_exp_args:Nx \__unravel_back_input:n
1337     { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
1338   \__unravel_get_next:
1339 }

```

(End 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.

```

1340 \cs_new_protected:Npn \__unravel_get_token:
1341 {
1342   \__unravel_get_next:
1343   \__unravel_set_cmd:
1344 }

```

(End 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 X_YTeX or LuaTeX primitive perhaps). Otherwise, it can be a control sequence or a character.

```

1345 \cs_new_protected:Npn \__unravel_set_cmd:
1346 {
1347   \__unravel_set_cmd_aux_meaning:
1348   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1349     { }
1350     {
1351       \__unravel_token_if_expandable:NTF \l__unravel_head_token
1352         {
1353           \token_if_macro:NTF \l__unravel_head_token

```

```

1354         { \__unravel_set_cmd_aux_macro: }
1355         { \__unravel_set_cmd_aux_unknown: }
1356     }
1357     {
1358         \token_if_cs:NTF \l__unravel_head_token
1359         { \__unravel_set_cmd_aux_cs: }
1360         { \__unravel_set_cmd_aux_char: }
1361     }
1362 }
1363 }

```

(End 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.

```

1364 \cs_new_protected:Npn \__unravel_set_cmd_aux_meaning:
1365 {
1366     \tl_set:Nx \l__unravel_head_meaning_tl
1367     {
1368         \exp_after:wN \__unravel_strip_escape:w
1369         \token_to_meaning:N \l__unravel_head_token
1370         \tl_to_str:n { : }
1371     }
1372     \tl_set:Nx \l__unravel_head_meaning_tl
1373     {
1374         \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1375         \l__unravel_head_meaning_tl \q_stop
1376     }
1377 }
1378 \use:x
1379 {
1380     \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1381     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1382 }

```

(End 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
1383 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
1384 {
1385     \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1386     {
1387         \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1388         { c__unravel_tex_#1_tl }
1389         #2
1390     }
1391 }
1392 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
1393 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
1394 {
1395     \int_set:Nn \l__unravel_head_cmd_int {#1}

```

```

1396 \int_set:Nn \l__unravel_head_char_int {#2}
1397 }

```

(End 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.

```

1398 \cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
1399 {
1400 \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1401 \int_zero:N \l__unravel_head_char_int
1402 }

```

(End 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.

```

1403 \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1404 {
1405 \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1406 \c__unravel_tex_relax_tl
1407 \__unravel_error:nxxxx { unknown-primitive }
1408 { \l__unravel_head_meaning_tl } { } { } { }
1409 }

```

(End 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` followed by " and an uppercase hexadecimal number, or one of `\count`, `\dimen`, `\skip`, `\muskip` or `\toks` followed by a decimal number.

```

1410 \cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
1411 {
1412 \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1413 { \tl_to_str:n { elect~font } }
1414 {
1415 \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1416 \c__unravel_tex_nullfont_tl
1417 }
1418 { \__unravel_set_cmd_aux_numeric: }
1419 }

```

(End 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` in upTeX), 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.

```

1420 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:
1421 {

```

```

1422 \tl_set:Nx \l__unravel_tmpa_tl
1423 {
1424   \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1425   \l__unravel_head_meaning_tl + 0
1426 }
1427 \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1428 \l__unravel_tmpa_tl \q_stop
1429 }
1430 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1431 {
1432   \if_int_compare:w '#1 < 'A \exp_stop_f:
1433   \exp_not:N \q_mark
1434   \exp_after:wN \use_i:nn
1435   \fi:
1436   #1 \__unravel_set_cmd_aux_numeric:N
1437 }
1438 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1439 {
1440   \str_case:nnF {#1}
1441   {
1442     { char } { \__unravel_set_cmd_aux_given:n { char_given } }
1443     { kchar } { \__unravel_set_cmd_aux_given:n { char_given } }
1444     { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1445   }
1446   {
1447     \__unravel_set_cmd_aux_primitive:nTF {#1}
1448     { }
1449     { \__unravel_set_cmd_aux_unknown: }
1450     \int_add:Nn \l__unravel_head_char_int { 100 000 }
1451   }
1452   \int_add:Nn \l__unravel_head_char_int {#2}
1453 }
1454 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1455 {
1456   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1457   \int_zero:N \l__unravel_head_char_int
1458 }

```

(End 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 ‘.

```

1459 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:
1460 {
1461   \tl_set:Nx \l__unravel_head_meaning_tl
1462   { \token_to_meaning:N \l__unravel_head_token }
1463   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1464   { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1465   \exp_after:wN \__unravel_set_cmd_aux_char:w
1466   \l__unravel_head_meaning_tl \q_stop
1467   \__unravel_exp_args:NNx \int_set:Nn \l__unravel_head_char_int

```

```

1468     { ' \_ unravel_token_to_char:N \_ unravel_head_token }
1469   }
1470   \cs_new_protected:Npn \_ unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1471   {
1472     \int_set:Nn \_ unravel_head_cmd_int
1473     { \_ unravel_tex_use:n { #1_char } }
1474   }

```

(End 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.

```

1475   \cs_new:Npn \_ unravel_input_to_str:
1476   {
1477     \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1478     \_ unravel_input_to_str_aux:n
1479   }
1480   \cs_new:Npn \_ unravel_input_to_str_aux:n #1
1481   { \gtl_to_str:c { g__unravel_input_#1_gtl } }

```

(End 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.

```

1482   \cs_new_protected:Npn \_ unravel_input_if_empty:TF
1483   {
1484     \int_compare:nNnTF \g__unravel_input_int = 0
1485     { \use_i:nn }
1486     {
1487       \gtl_if_empty:cTF
1488       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1489       {
1490         \int_gdecr:N \g__unravel_input_int
1491         \_ unravel_input_if_empty:TF
1492       }
1493     }
1494     \_ unravel_input_split:
1495     \use_ii:nn
1496   }
1497 }
1498 }

```

(End 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 character

```

1499   \cs_new_protected:Npn \_ unravel_input_split:
1500   {
1501     \int_compare:nNnT \g__unravel_input_int = 1
1502     {

```

```

1503         \exp_args:Nc \__unravel_input_split_aux:N
1504         { g__unravel_input_1_gtl }
1505     }
1506 }
1507 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1508 {
1509     \gtl_if_tl:NT #1
1510     {
1511         \gtl_if_head_is_N_type:NT #1
1512         {
1513             \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1514             \__unravel_exp_args:NNx \use:nn
1515             \__unravel_input_split_auxii:N
1516             { \tl_head:N \l__unravel_input_tmpa_tl }
1517         }
1518     }
1519 }
1520 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1521 {
1522     \token_if_parameter:NF #1
1523     {
1524         \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1525         { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1526         \group_begin:
1527         \cs_set:Npn \__unravel_input_split_auxiii:w
1528             ##1 \__unravel_input_split_end: { + 1 }
1529         \int_gset:Nn \g__unravel_input_int
1530             { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1531         \group_end:
1532         \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1533         \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1534     }
1535 }
1536 \cs_new:Npn \__unravel_input_split_end: { }
1537 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1538     #1 \__unravel_input_split_end:
1539 {
1540     \gtl_gclear_new:c
1541     { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1542     \gtl_gset:cn
1543     { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1544     \int_gdecr:N \g__unravel_input_tmpa_int
1545 }

```

(End definition for __unravel_input_split:.)

__unravel_input_gset:n At first, all of the input is in the same gtl.

```

1546 \cs_new_protected:Npn \__unravel_input_gset:n
1547 {
1548     \int_gzero:N \g__unravel_input_int
1549     \__unravel_back_input:n
1550 }

```

(End definition for __unravel_input_gset:n.)

_unravel_input_get:N

```

1551 \cs_new_protected:Npn \_unravel_input_get:N #1
1552 {
1553   \_unravel_input_if_empty:TF
1554     { \gtl_set:Nn #1 { \q_no_value } }
1555     {
1556       \gtl_get_left:cN
1557       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1558     }
1559 }

```

(End definition for _unravel_input_get:N.)

_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.

```

1560 \cs_new_protected:Npn \_unravel_input_gpop:N #1
1561 {
1562   \_unravel_input_if_empty:TF
1563     { \gtl_set:Nn #1 { \q_no_value } }
1564     {
1565       \gtl_gpop_left:cN
1566       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1567     }
1568 }

```

(End 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.

```

1569 \cs_new_protected:Npn \_unravel_input_merge:
1570 {
1571   \int_gdecr:N \g__unravel_input_int
1572   \gtl_gconcat:ccc
1573     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1574     { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1575     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1576   \gtl_gclear:c
1577   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1578 }

```

(End definition for _unravel_input_merge:.)

_unravel_input_gpop_item:N

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

```

1579 \prg_new_protected_conditional:Npnn \_unravel_input_gpop_item:N #1 { F }
1580 {
1581   \int_compare:nNnTF \g__unravel_input_int = 0
1582     { \prg_return_false: }
1583     {
1584       \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1585       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1

```

```

1586     }
1587   }
1588   \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
1589   {
1590     \gtl_gpop_left_item:NNTF #1#2
1591     { \prg_return_true: }
1592     {
1593       \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
1594       { \prg_return_false: }
1595       {
1596         \int_compare:nNnTF \g__unravel_input_int = 1
1597         { \prg_return_false: }
1598         {
1599           \__unravel_input_merge:
1600           \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1601             {
1602               g__unravel_input_
1603               \int_use:N \g__unravel_input_int _gtl
1604             }
1605             #2
1606           }
1607         }
1608       }
1609     }

```

(End definition for __unravel_input_gpop_item:NNTF and __unravel_input_gpop_item_aux:NN.)

__unravel_input_gpop_tl:N

```

1610   \cs_new_protected:Npn \__unravel_input_gpop_tl:N #1
1611   { \tl_clear:N #1 \__unravel_input_gpop_tl_aux:N #1 }
1612   \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:N #1
1613   {
1614     \int_compare:nNnF \g__unravel_input_int = 0
1615     {
1616       \exp_args:Nc \__unravel_input_gpop_tl_aux:NN
1617       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1618     }
1619   }
1620   \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
1621   {
1622     \gtl_if_tl:NNTF #1
1623     {
1624       \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1625       \gtl_gclear:N #1
1626       \int_gdecr:N \g__unravel_input_int
1627       \__unravel_input_gpop_tl_aux:N #2
1628     }
1629     {
1630       \int_compare:nNnTF \g__unravel_input_int > 1
1631       { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
1632       { \use_i:nn }
1633       {
1634         \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1635         \gtl_gpop_left_tl:N #1

```

```

1636     }
1637     {
1638         \__unravel_input_merge:
1639         \__unravel_input_gpop_tl_aux:N #2
1640     }
1641 }
1642 }

```

(End definition for __unravel_input_gpop_tl:N.)

__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.

```

1643 \cs_new_protected:Npn \__unravel_back_input:n
1644 {
1645     \int_gincr:N \g__unravel_input_int
1646     \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1647     \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1648 }
1649 \cs_generate_variant:Nn \__unravel_back_input:n { V , o }
1650 \cs_new_protected:Npn \__unravel_back_input:x
1651 { \__unravel_exp_args:Nx \__unravel_back_input:n }

```

(End definition for __unravel_back_input:n.)

__unravel_back_input_gtl:N Insert a generalized token list back into the input.

```

1652 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1653 {
1654     \gtl_if_tl:NTF #1
1655     { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1656     {
1657         \gtl_gconcat:cNc
1658         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1659         #1
1660         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1661     }
1662 }

```

(End definition for __unravel_back_input_gtl:N.)

__unravel_back_input: Insert the last token read back into the input stream.

```

1663 \cs_new_protected:Npn \__unravel_back_input:
1664 { \__unravel_back_input_gtl:N \l__unravel_head_gtl }

```

(End 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.

```

1665 \cs_new_protected:Npn \__unravel_back_input_tl_o:
1666 {
1667     \tl_set:Nx \l__unravel_tmpa_tl
1668     { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1669     \__unravel_back_input:V \l__unravel_tmpa_tl
1670     \__unravel_print_expansion:x
1671     { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1672 }

```

(End 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.

```

1673 \cs_new_protected:Npn \_unravel_insert_relax:
1674 {
1675   \_unravel_back_input:
1676   \gtl_set_eq:NN \l\_unravel_head_gtl \c\_unravel_frozen_relax_gtl
1677   \_unravel_back_input:
1678   \_unravel_print_action:
1679 }

```

(End definition for _unravel_insert_relax:.)

_unravel_insert_group_begin_error:

```

1680 \cs_new_protected:Npn \_unravel_insert_group_begin_error:
1681 {
1682   \tl_set_eq:NN \l\_unravel_tmpa_tl \l\_unravel_head_tl
1683   \_unravel_back_input:
1684   \gtl_set_eq:NN \l\_unravel_head_gtl \c_group_begin_gtl
1685   \_unravel_back_input:
1686   \_unravel_tex_error:nV { missing-lbrace } \l\_unravel_tmpa_tl
1687   \_unravel_print_action:
1688 }

```

(End definition for _unravel_insert_group_begin_error:.)

_unravel_insert_dollar_error:

```

1689 \cs_new_protected:Npn \_unravel_insert_dollar_error:
1690 {
1691   \_unravel_back_input:
1692   \_unravel_back_input:n { $ } % $
1693   \_unravel_error:nnnnn { missing-dollar } { } { } { } { }
1694   \_unravel_print_action:
1695 }

```

(End definition for _unravel_insert_dollar_error:.)

2.5.3 Macro calls

_unravel_macro_prefix:N

_unravel_macro_parameter:N

_unravel_macro_replacement:N

```

1696 \use:x
1697 {
1698   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:NN #1 }
1699   {
1700     \exp_not:n { \exp_after:wN \_unravel_macro_split_do:wN }
1701     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1702     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1703     \exp_not:N \q_stop
1704   }
1705   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:wN }

```

```

1706         \exp_not:n {#1} \tl_to_str:n { : } \exp_not:n { #2 -> }
1707         \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1708         { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
1709     }
1710 \cs_new:Npn \__unravel_macro_prefix:N #1
1711 { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1712 \cs_new:Npn \__unravel_macro_parameter:N #1
1713 { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1714 \cs_new:Npn \__unravel_macro_replacement:N #1
1715 { \__unravel_macro_split_do:NN #1 \use_iii:nnn }

```

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

__unravel_macro_call: 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 __unravel_macro_call_safe: we can be safe by extracting the longest balanced prefix in the input and working with that. __unravel_macro_call_quick: __unravel_macro_call_quick_loop:NNN __unravel_macro_call_quick_runaway:Nw

```

1716 \cs_new_protected:Npn \__unravel_macro_call:
1717 {
1718     \bool_if:NTF \g__unravel_speedup_macros_bool
1719     {
1720         \tl_set:Nx \l__unravel_tmpa_tl
1721         { ^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1722         \__unravel_tl_if_in:ooTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1723         { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1724     }
1725     { \__unravel_macro_call_safe: }
1726     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1727     \__unravel_print_expansion:
1728 }
1729 \cs_new_protected:Npn \__unravel_macro_call_safe:
1730 {
1731     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1732     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1733 }
1734 \cs_new_protected:Npn \__unravel_macro_call_quick:
1735 {
1736     \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1737     { ? \use_none_delimit_by_q_stop:w } \q_stop
1738 }
1739 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1740 {
1741     \use_none:n #2
1742     \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1743     { \__unravel_macro_call_quick_runaway:Nw #3 }
1744     \tl_put_right:Nx \l__unravel_head_tl
1745     { { \exp_not:V \l__unravel_tmpa_tl } }
1746     \__unravel_macro_call_quick_loop:NNN
1747     #3
1748 }
1749 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1750 {
1751     \__unravel_error:nxxxx { runaway-macro-parameter }

```

```

1752     { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1753 }

```

(End 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 mimic `TEX`'s structure, distinguishing macros from other commands because we find macro arguments very differently from primitives.

```

1754 \cs_new_protected:Npn \__unravel_expand_do:N
1755 {
1756   \__unravel_set_action_text:
1757   \bool_if:NT \g__unravel_internal_debug_bool
1758   {
1759     \__unravel_set_cmd:
1760     \__unravel_exp_args:Nx \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_int
1761   }
1762   \token_if_macro:NTF \l__unravel_head_token
1763   { \__unravel_macro_call: }
1764   { \__unravel_expand_nonmacro: }
1765 }

```

(End 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.

```

1766 \cs_new_protected:Npn \__unravel_expand_nonmacro:
1767 {
1768   \__unravel_set_cmd_aux_meaning:
1769   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1770   {
1771     \cs_if_exist_use:cF
1772     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1773     { \__unravel_error:nxxxx { internal } { expandable } { } { } { } }
1774   }
1775   {
1776     \__unravel_error:nxxxx { unknown-primitive }

```

```

1777         { \l__unravel_head_meaning_tl } { } { } { }
1778         \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1779         \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1780         \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1781         \__unravel_print_expansion:
1782     }
1783 }

```

(End 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^AT_EX3 these are situations where T_EX “f-expands”). It is also the basis of the `__unravel_main_loop:`.

```

1784 \cs_new_protected:Npn \__unravel_get_x_next:
1785 {
1786     \__unravel_get_next:
1787     \__unravel_token_if_expandable:NT \l__unravel_head_token
1788     { \__unravel_expand_do:N \__unravel_get_x_next: }
1789 }

```

(End 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.

```

1790 \cs_new_protected:Npn \__unravel_get_x_or_protected:
1791 {
1792     \__unravel_get_next:
1793     \__unravel_token_if_protected:NF \l__unravel_head_token
1794     { \__unravel_expand_do:N \__unravel_get_x_or_protected: }
1795 }

```

(End 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).

```

1796 \cs_new_protected:Npn \__unravel_get_token_xdef:
1797 {
1798     \__unravel_get_next:
1799     \__unravel_token_if_protected:NF \l__unravel_head_token
1800     { \__unravel_expand_do:N \__unravel_get_token_xdef: }
1801 }
1802 \cs_new_protected:Npn \__unravel_get_token_x:
1803 {
1804     \__unravel_get_next:
1805     \__unravel_token_if_protected:NF \l__unravel_head_token
1806     { \__unravel_expand_do:N \__unravel_get_token_x: }
1807 }

```

(End 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.

```

1808 \cs_new_protected:Npn \__unravel_get_x_non_blank:
1809 {
1810   \__unravel_get_x_next:
1811   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1812     { \__unravel_get_x_non_blank: }
1813 }
```

(End definition for __unravel_get_x_non_blank:.)

`__unravel_get_x_non_relax:` This function does not set the cmd and char integers.

```

1814 \cs_new_protected:Npn \__unravel_get_x_non_relax:
1815 {
1816   \__unravel_get_x_next:
1817   \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
1818     { \__unravel_get_x_non_relax: }
1819   {
1820     \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_special_relax:
1821       { \__unravel_get_x_non_relax: }
1822     {
1823       \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1824         { \__unravel_get_x_non_relax: }
1825     }
1826   }
1827 }
```

(End definition for __unravel_get_x_non_relax:.)

`__unravel_skip_optional_space:`

```

1828 \cs_new_protected:Npn \__unravel_skip_optional_space:
1829 {
1830   \__unravel_get_x_next:
1831   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
1832     { \__unravel_back_input: }
1833 }
```

(End 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_scan_something_internal:n` leaves raw numbers in the previous-input sequence.

```

1834 \cs_new_protected:Npn \__unravel_scan_optional_equals:
1835 {
1836   \__unravel_get_x_non_blank:
1837   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1838     { \__unravel_prev_input:n { = } }
1839   {
1840     \__unravel_prev_input_silent:n { = }
1841     \__unravel_back_input:
1842   }
1843 }
```

(End 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:`.

```

1844 \cs_new_protected:Npn \__unravel_scan_left_brace:
1845 {
1846   \__unravel_get_x_non_relax:
1847   \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
1848   {
1849     \__unravel_insert_group_begin_error:
1850     \__unravel_get_next:
1851   }
1852 }

```

(End definition for `__unravel_scan_left_brace:`.)

`__unravel_scan_keyword:n` The details of how T_EX 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 { pPtT }`. 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_EX’s skipping of leading spaces here must be intertwined with the search for keyword, as is shown by the (plain T_EX) example

```

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

1853 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1854 { \__unravel_scan_keyword:nTF {#1} { } { } }
1855 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1856 { T , F , TF }
1857 {
1858   \__unravel_prev_input_gpush_gtl:
1859   \__unravel_scan_keyword_loop:NNN \c_true_bool
1860   #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1861 }
1862 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1863 {
1864   \quark_if_recursion_tail_stop_do:nn {#2}
1865   { \__unravel_scan_keyword_true: }

```

```

1866 \quark_if_recursion_tail_stop_do:nn {#3}
1867 { \__unravel_error:nxxxx { internal } { odd-keyword-length } { } { } { } }
1868 \__unravel_get_x_next:
1869 \__unravel_scan_keyword_test:NNTF #2#3
1870 {
1871   \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1872   \__unravel_scan_keyword_loop:NNN \c_false_bool
1873 }
1874 {
1875   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
1876   { \__unravel_scan_keyword_false:w }
1877   \bool_if:NF #1
1878   { \__unravel_scan_keyword_false:w }
1879   \__unravel_scan_keyword_loop:NNN #1#2#3
1880 }
1881 }
1882 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1883 { TF }
1884 {
1885   \__unravel_gtl_if_head_is_definable:NNTF \l__unravel_head_gtl
1886   { \prg_return_false: }
1887   {
1888     \str_if_eq:eeTF
1889     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1890     { \prg_return_true: }
1891     {
1892       \str_if_eq:eeTF
1893       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
1894       { \prg_return_true: }
1895       { \prg_return_false: }
1896     }
1897   }
1898 }
1899 \cs_new_protected:Npn \__unravel_scan_keyword_true:
1900 {
1901   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1902   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1903   \prg_return_true:
1904 }
1905 \cs_new_protected:Npn \__unravel_scan_keyword_false:w
1906 #1 \q_recursion_stop
1907 {
1908   \__unravel_back_input:
1909   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1910   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1911   \prg_return_false:
1912 }

```

(End definition for __unravel_scan_keyword:n and others.)

__unravel_scan_to: Used when to is mandatory: after \read or \readline and after \vsplit.

```

1913 \cs_new_protected:Npn \__unravel_scan_to:
1914 {
1915   \__unravel_scan_keyword:nF { tTo0 }

```

```

1916     {
1917       \__unravel_error:nnnnn { missing-to } { } { } { } { }
1918       \__unravel_prev_input:n { to }
1919     }
1920   }

```

(End definition for __unravel_scan_to:.)

__unravel_scan_font_ident: Find a font identifier.

```

1921 \cs_new_protected:Npn \__unravel_scan_font_ident:
1922 {
1923   \__unravel_get_x_non_blank:
1924   \__unravel_set_cmd:
1925   \int_case:nnF \l__unravel_head_cmd_int
1926   {
1927     { \__unravel_tex_use:n { def_font } }
1928     { \__unravel_prev_input:V \l__unravel_head_tl }
1929     { \__unravel_tex_use:n { letterspace_font } }
1930     { \__unravel_prev_input:V \l__unravel_head_tl }
1931     { \__unravel_tex_use:n { pdf_copy_font } }
1932     { \__unravel_prev_input:V \l__unravel_head_tl }
1933     { \__unravel_tex_use:n { set_font } }
1934     { \__unravel_prev_input:V \l__unravel_head_tl }
1935     { \__unravel_tex_use:n { def_family } }
1936     {
1937       \__unravel_prev_input:V \l__unravel_head_tl
1938       \__unravel_scan_int:
1939     }
1940   }
1941   {
1942     \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
1943     \__unravel_back_input:
1944     \__unravel_prev_input:n { \__unravel_nullfont: }
1945   }
1946 }

```

(End definition for __unravel_scan_font_ident:.)

__unravel_scan_font_int: Find operands for one of \hyphenchar's friends (command code assign_font_int=78).

```

1947 \cs_new_protected:Npn \__unravel_scan_font_int:
1948 {
1949   \int_case:nnF \l__unravel_head_char_int
1950   {
1951     { 0 } { \__unravel_scan_font_ident: }
1952     { 1 } { \__unravel_scan_font_ident: }
1953     { 6 } { \__unravel_scan_font_ident: }
1954   }
1955   { \__unravel_scan_font_ident: \__unravel_scan_int: }
1956 }

```

(End definition for __unravel_scan_font_int:.)

__unravel_scan_font_dimen: Find operands for \fontdimen.

```

1957 \cs_new_protected:Npn \__unravel_scan_font_dimen:
1958 {

```

```

1959     \__unravel_scan_int:
1960     \__unravel_scan_font_ident:
1961   }

(End definition for \__unravel_scan_font_dimen:.)

```

__unravel_scan_something_internal:n
 __unravel_scan_something_aux:nwn

Receives an (explicit) “level” argument:

- `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 ≥ 4 (that is, 4, 5, or 8) complain about a missing number and insert a zero dimension, to get exactly T_EX’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).

```

1962 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1963 {
1964   \__unravel_set_cmd:
1965   \__unravel_set_action_text:
1966   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1967   \exp_after:wN \__unravel_scan_something_aux:nwn
1968   \l__unravel_tmpa_tl \q_stop {#1}
1969 }
1970 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1971 {
1972   \int_compare:nT { #3 < 4 <= #1 }
1973   {

```

```

1974     \__unravel_back_input:
1975     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
1976     \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
1977     \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
1978     \__unravel_break:w
1979 }
1980 \int_compare:nNnT {#1} = { 8 }
1981 {
1982     \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
1983     \__unravel_scan_something_internal_auxii:nn 0 { 0 }
1984     \__unravel_break:w
1985 }
1986 \tl_if_empty:nF {#2}
1987 {
1988     \__unravel_prev_input_gpush:N \l__unravel_head_tl
1989     \__unravel_print_action:
1990     #2
1991     \__unravel_prev_input_gpop:N \l__unravel_head_tl
1992 }
1993 \int_compare:nNnTF {#3} < { 4 }
1994 { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
1995 { \tl_set:Nx \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
1996 \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
1997 \__unravel_break_point:
1998 \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
1999 }
2000 \cs_new_protected:Npn \__unravel_scan_something_internal_auxii:nn #1#2
2001 {
2002     \__unravel_prev_input_silent:n {#1}
2003     \__unravel_set_action_text:
2004     \__unravel_set_action_text:x
2005     { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
2006     \int_gset:Nn \g__unravel_val_level_int {#2}
2007 }
2008 \cs_generate_variant:Nn \__unravel_scan_something_internal_auxii:nn { V }

```

(End definition for __unravel_scan_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 and register, the level
 of the token depends on the char integer. When the token is not allowed after \the (or
 at any other position where __unravel_scan_something_internal:n is called), the
 resulting level is 8, large enough so that the main function knows it is forbidden.

```

2009 \cs_new:Npn \__unravel_thing_case:
2010 {
2011     \int_case:nnF \l__unravel_head_cmd_int
2012     {
2013         { 68 } { 0 } % char_given
2014         { 69 } { 0 } % math_given
2015         { 70 } { \__unravel_thing_last_item: } % last_item
2016         { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
2017         { 72 } { 5 } % assign_toks
2018         { 73 } { 0 } % assign_int

```

```

2019 { 74 } { 1 } % assign_dimen
2020 { 75 } { 2 } % assign_glue
2021 { 76 } { 3 } % assign_mu_glue
2022 { 77 } { 1 \_unravel_scan_font_dimen: } % assign_font_dimen
2023 { 78 } { 0 \_unravel_scan_font_int: } % assign_font_int
2024 { 79 } { \_unravel_thing_set_aux: } % set_aux
2025 { 80 } { 0 } % set_prev_graf
2026 { 81 } { 1 } % set_page_dimen
2027 { 82 } { 0 } % set_page_int
2028 { 83 } { 1 \_unravel_scan_int: } % set_box_dimen
2029 { 84 } { 0 \_unravel_scan_int: } % set_shape
2030 { 85 } { 0 \_unravel_scan_int: } % def_code
2031 { 86 } { 4 \_unravel_scan_int: } % def_family
2032 { 87 } { 4 } % set_font
2033 { 88 } { 4 } % def_font
2034 { 89 } { \_unravel_thing_register: } % register
2035 {101 } { 4 } % letterspace_font
2036 {102 } { 4 } % pdf_copy_font
2037 }
2038 { 8 }
2039 }
2040 \cs_new:Npn \_unravel_thing_set_aux:
2041 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
2042 \cs_new:Npn \_unravel_thing_last_item:
2043 {
2044   \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
2045   {
2046     \int_case:nnF \l__unravel_head_char_int
2047     {
2048       { 1 } { 1 } % lastkern
2049       { 2 } { 2 } % lastskip
2050     }
2051     { 0 } % other integer parameters
2052   }
2053   {
2054     \int_case:nnF \l__unravel_head_char_int
2055     {
2056       { 26 } { 0 \_unravel_scan_normal_glue: } % gluestretchorder
2057       { 27 } { 0 \_unravel_scan_normal_glue: } % glueshrinkorder
2058       { 28 } % fontcharwd
2059       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2060       { 29 } % fontcharht
2061       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2062       { 30 } % fontchardp
2063       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2064       { 31 } % fontcharic
2065       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2066       { 32 } { 1 \_unravel_scan_int: } % parshape length
2067       { 33 } { 1 \_unravel_scan_int: } % parshape indent
2068       { 34 } { 1 \_unravel_scan_int: } % parshape dimen
2069       { 35 } { 1 \_unravel_scan_normal_glue: } % gluestretch
2070       { 36 } { 1 \_unravel_scan_normal_glue: } % glueshrink
2071       { 37 } { 2 \_unravel_scan_mu_glue: } % mutoglu
2072       { 38 } { 3 \_unravel_scan_normal_glue: } % gluetomu

```

```

2073         { 39 } % numexpr
2074         { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
2075         { 40 } % dimexpr
2076         { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
2077         { 41 } % glueexpr
2078         { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
2079         { 42 } % muexpr
2080         { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
2081     }
2082     { }
2083 }
2084 }
2085 \cs_new:Npn \__unravel_thing_register:
2086 {
2087     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
2088     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
2089     { \__unravel_scan_int: }
2090 }

```

(End 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.

```

2091 \cs_new_protected:Npn \__unravel_scan_toks_register:
2092 {
2093     \int_compare:nNnT \l__unravel_head_char_int = 0
2094     { \__unravel_scan_int: }
2095 }

```

(End definition for `__unravel_scan_toks_register:`.)

`__unravel_thing_use_get:nnNN` 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.

```

2096 \cs_new_protected:Npn \__unravel_thing_use_get:nnNN #1#2#3#4
2097 {
2098     \int_compare:nNnTF {#2} < { 3 }
2099     {
2100         \int_compare:nNnT {#1} = { 3 }
2101         { \__unravel_tex_error:nV { incompatible-units } #3 }
2102         \tl_set:Nx #4
2103         {
2104             \int_case:nn { \int_min:nn {#1} {#2} }
2105             {
2106                 { 0 } \int_eval:n
2107                 { 1 } \dim_eval:n
2108                 { 2 } \skip_eval:n
2109             }
2110             { \int_compare:nNnT {#1} = { 3 } \tex_mutogluue:D #3 }
2111         }
2112     }
2113     {
2114         \int_case:nnF {#1}
2115         {
2116             { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
2117             { 3 } { \tl_set:Nx #4 { \mskip_eval:n {#3} } }

```

```

2118     }
2119     {
2120         \_unravel_tex_error:nV { incompatible-units } #3
2121         \tl_set:Nx #4 { \muskip_eval:n { \tex_glue_tomu:D #3 } }
2122     }
2123 }
2124 }

```

(End definition for _unravel_thing_use_get:nnNN.)

```

\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
2125 \cs_new_protected:Npn \_unravel_scan_expr:N #1
2126 { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
2127 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
2128 {
2129     \_unravel_get_x_non_blank:
2130     \_unravel_scan_factor:N #1
2131     \_unravel_scan_expr_op:NN #1#2
2132 }
2133 \cs_new_protected:Npn \_unravel_scan_expr_op:NN #1#2
2134 {
2135     \_unravel_get_x_non_blank:
2136     \tl_case:NnF \l__unravel_head_tl
2137     {
2138         \c__unravel_plus_tl
2139         {
2140             \_unravel_prev_input:V \l__unravel_head_tl
2141             \_unravel_scan_expr_aux:NN #1#2
2142         }
2143         \c__unravel_minus_tl
2144         {
2145             \_unravel_prev_input:V \l__unravel_head_tl
2146             \_unravel_scan_expr_aux:NN #1#2
2147         }
2148         \c__unravel_times_tl
2149         {
2150             \_unravel_prev_input:V \l__unravel_head_tl
2151             \_unravel_get_x_non_blank:
2152             \_unravel_scan_factor:N \_unravel_scan_int:
2153             \_unravel_scan_expr_op:NN #1#2
2154         }
2155         \c__unravel_over_tl
2156         {
2157             \_unravel_prev_input:V \l__unravel_head_tl
2158             \_unravel_get_x_non_blank:
2159             \_unravel_scan_factor:N \_unravel_scan_int:
2160             \_unravel_scan_expr_op:NN #1#2
2161         }
2162         \c__unravel_rp_tl
2163         {
2164             \bool_if:NTF #2
2165             { \_unravel_prev_input:V \l__unravel_head_tl }
2166             { \_unravel_back_input: }
2167         }
2168     }

```

```

2168     }
2169     {
2170         \bool_if:NTF #2
2171         {
2172             \__unravel_error:nnnnn { missing-rparen } { } { } { } { } { }
2173             \__unravel_back_input:
2174             \__unravel_prev_input:V \c__unravel_rp_tl
2175         }
2176         {
2177             \token_if_eq_meaning:NMF \l__unravel_head_token \scan_stop:
2178             { \__unravel_back_input: }
2179         }
2180     }
2181 }
2182 \cs_new_protected:Npn \__unravel_scan_factor:N #1
2183 {
2184     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2185     {
2186         \__unravel_prev_input:V \l__unravel_head_tl
2187         \__unravel_scan_expr_aux:NN #1 \c_true_bool
2188     }
2189     {
2190         \__unravel_back_input:
2191         #1
2192     }
2193 }

```

(End 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.

```

2194 \cs_new_protected:Npn \__unravel_scan_signs:
2195 {
2196     \__unravel_get_x_non_blank:
2197     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
2198     {
2199         \__unravel_prev_input:V \l__unravel_head_tl
2200         \__unravel_scan_signs:
2201     }
2202     {
2203         \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_minus_tl
2204         {
2205             \__unravel_prev_input:V \l__unravel_head_tl
2206             \__unravel_scan_signs:
2207         }
2208     }
2209 }

```

(End definition for __unravel_scan_signs:.)

```

\__unravel_scan_int:
\__unravel_scan_int_char: 2210 \cs_new_protected:Npn \__unravel_scan_int:
\__unravel_scan_int_lq: 2211 {
\__unravel_scan_int_explicit:n 2212     \__unravel_scan_signs:

```

```

2213     \__unravel_set_cmd:
2214     \__unravel_cmd_if_internal:TF
2215     { \__unravel_scan_something_internal:n { 0 } }
2216     { \__unravel_scan_int_char: }
2217 }
2218 \cs_new_protected:Npn \__unravel_scan_int_char:
2219 {
2220   \tl_case:NnF \l__unravel_head_tl
2221   {
2222     \c__unravel_lq_tl { \__unravel_scan_int_lq: }
2223     \c__unravel_rq_tl
2224     {
2225       \__unravel_prev_input:V \l__unravel_head_tl
2226       \__unravel_get_x_next:
2227       \__unravel_scan_int_explicit:Nn \c_false_bool { ' }
2228     }
2229     \c__unravel_dq_tl
2230     {
2231       \__unravel_prev_input:V \l__unravel_head_tl
2232       \__unravel_get_x_next:
2233       \__unravel_scan_int_explicit:Nn \c_false_bool { " }
2234     }
2235   }
2236   { \__unravel_scan_int_explicit:Nn \c_false_bool { } }
2237 }
2238 \cs_new_protected:Npn \__unravel_scan_int_lq:
2239 {
2240   \__unravel_get_next:
2241   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2242   {
2243     \tl_set:Nx \l__unravel_head_tl
2244     { \__unravel_token_to_char:N \l__unravel_head_token }
2245   }
2246   \tl_set:Nx \l__unravel_tmpa_tl
2247   { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
2248   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2249   \__unravel_print_action:x
2250   { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2251   \__unravel_skip_optional_space:
2252 }
2253 \cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
2254 {
2255   \if_int_compare:w 1
2256   < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2257   \exp_after:wN \use_i:nn
2258   \else:
2259   \exp_after:wN \use_ii:nn
2260   \fi:
2261   {
2262     \__unravel_prev_input:V \l__unravel_head_tl
2263     \__unravel_get_x_next:
2264     \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
2265   }
2266   {

```

```

2267 \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2268 { \__unravel_back_input: }
2269 \bool_if:NF #1
2270 {
2271   \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2272   \__unravel_prev_input:n { 0 }
2273 }
2274 }
2275 }

```

(End definition for `__unravel_scan_int:` and others.)

`__unravel_scan_normal_dimen:`

```

2276 \cs_new_protected:Npn \__unravel_scan_normal_dimen:
2277 { \__unravel_scan_dimen:nN { 2 } \c_false_bool }

```

(End 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`.

```

2278 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2279 {
2280   \__unravel_scan_signs:
2281   \__unravel_prev_input_gpush:
2282   \__unravel_set_cmd:
2283   \__unravel_cmd_if_internal:TF
2284   {
2285     \int_compare:nNnTF {#1} = { 3 }
2286     { \__unravel_scan_something_internal:n { 3 } }
2287     { \__unravel_scan_something_internal:n { 1 } }
2288     \int_compare:nNnT \g__unravel_val_level_int = { 0 }
2289     { \__unravel_scan_dim_unit:nN {#1} #2 }
2290   }
2291   { \__unravel_scan_dimen_char:nN {#1} #2 }
2292   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2293   \__unravel_prev_input_silent:V \l__unravel_head_tl
2294 }
2295 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
2296 {
2297   \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2298   { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2299   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
2300   {
2301     \__unravel_prev_input:n { . }
2302     \__unravel_scan_decimal_loop:
2303   }
2304   {
2305     \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl
2306     {
2307       \__unravel_back_input:

```

```

2308         \__unravel_scan_int:
2309         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2310         { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2311         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
2312         {
2313             \__unravel_input_gpop:N \l__unravel_tmpb_gtl
2314             \__unravel_prev_input:n { . }
2315             \__unravel_scan_decimal_loop:
2316         }
2317     }
2318     {
2319         \__unravel_back_input:
2320         \__unravel_scan_int:
2321     }
2322 }
2323 \__unravel_scan_dim_unit:nN {#1} #2
2324 }
2325 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
2326 {
2327     \bool_if:NT #2
2328     {
2329         \__unravel_scan_keyword:nT { fFiIlL }
2330         {
2331             \__unravel_scan_inf_unit_loop:
2332             \__unravel_break:w
2333         }
2334     }
2335     \__unravel_get_x_non_blank:
2336     \__unravel_set_cmd:
2337     \__unravel_cmd_if_internal:TF
2338     {
2339         \__unravel_prev_input_gpush:
2340         \__unravel_scan_something_internal:n {#1}
2341         \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2342         \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2343         \__unravel_break:w
2344     }
2345     { \__unravel_back_input: }
2346     \int_compare:nNnT {#1} = { 3 }
2347     {
2348         \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2349         \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
2350         \__unravel_prev_input:n { mu }
2351         \__unravel_break:w
2352     }
2353     \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2354     \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2355     \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2356     \__unravel_scan_keyword:nT { tTrRuUeE }
2357     { \__unravel_prepare_mag: }
2358     \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2359     \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2360     \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2361     \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }

```

```

2362 \_unravel\_scan\_keyword:nT { mMM } { \_unravel\_break:w }
2363 \_unravel\_scan\_keyword:nT { bBpP } { \_unravel\_break:w }
2364 \_unravel\_scan\_keyword:nT { dDdD } { \_unravel\_break:w }
2365 \_unravel\_scan\_keyword:nT { cCcC } { \_unravel\_break:w }
2366 \_unravel\_scan\_keyword:nT { nNdD } { \_unravel\_break:w }
2367 \_unravel\_scan\_keyword:nT { nNcC } { \_unravel\_break:w }
2368 \_unravel\_scan\_keyword:nT { sSpP } { \_unravel\_break:w }
2369 \_unravel\_tex\_error:nV { missing-pt } \l\_unravel\_head\_tl
2370 \_unravel\_prev\_input:n { pt }
2371 \_unravel\_break\_point:
2372 }
2373 \cs\_new\_protected:Npn \_unravel\_scan\_inf\_unit\_loop:
2374 { \_unravel\_scan\_keyword:nT { lL } { \_unravel\_scan\_inf\_unit\_loop: } }
2375 \cs\_new\_protected:Npn \_unravel\_scan\_decimal\_loop:
2376 {
2377 \_unravel\_get\_x\_next:
2378 \tl\_if\_empty:NTF \l\_unravel\_head\_tl
2379 { \use\_ii:nn }
2380 { \_unravel\_tl\_if\_in:ooTF { 0123456789 } \l\_unravel\_head\_tl }
2381 {
2382 \_unravel\_prev\_input:V \l\_unravel\_head\_tl
2383 \_unravel\_scan\_decimal\_loop:
2384 }
2385 {
2386 \token\_if\_eq\_catcode:NMF \l\_unravel\_head\_token \c\_space\_token
2387 { \_unravel\_back\_input: }
2388 \_unravel\_prev\_input\_silent:n { ~ }
2389 }
2390 }

```

(End definition for _unravel_scan_dimen:nN.)

```

\_unravel\_scan\_normal\_glue:
\_unravel\_scan\_mu\_glue:
2391 \cs\_new\_protected:Npn \_unravel\_scan\_normal\_glue:
2392 { \_unravel\_scan\_glue:n { 2 } }
2393 \cs\_new\_protected:Npn \_unravel\_scan\_mu\_glue:
2394 { \_unravel\_scan\_glue:n { 3 } }

```

(End definition for _unravel_scan_normal_glue: and _unravel_scan_mu_glue:.)

```

\_unravel\_scan\_glue:n
2395 \cs\_new\_protected:Npn \_unravel\_scan\_glue:n #1
2396 {
2397 \_unravel\_prev\_input\_gpush:
2398 \_unravel\_scan\_signs:
2399 \_unravel\_prev\_input\_gpush:
2400 \_unravel\_set\_cmd:
2401 \_unravel\_cmd\_if\_internal:TF
2402 {
2403 \_unravel\_scan\_something\_internal:n {#1}
2404 \int\_case:nnF \g\_unravel\_val\_level\_int
2405 {
2406 { 0 } { \_unravel\_scan\_dim\_unit:nN {#1} \c\_false\_bool }
2407 { 1 } { }
2408 }

```

```

2409         { \_unravel_break:w }
2410     }
2411     { \_unravel_back_input: \_unravel_scan_dimen:nN {#1} \c_false_bool }
2412     \_unravel_prev_input_join_get:nN {#1} \l_unravel_tmpa_tl
2413     \_unravel_prev_input_gpush:
2414     \_unravel_prev_input_gpush:N \l_unravel_tmpa_tl
2415     \_unravel_scan_keyword:nT { pPlLuUsS }
2416     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2417     \_unravel_scan_keyword:nT { mMiInNuUsS }
2418     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2419     \_unravel_break_point:
2420     \_unravel_prev_input_join_get:nN {#1} \l_unravel_tmpa_tl
2421     \_unravel_prev_input_silent:V \l_unravel_tmpa_tl
2422 }

```

(End definition for _unravel_scan_glue:n.)

_unravel_scan_file_name:

```

2423 \cs_new_protected:Npn \_unravel_scan_file_name:
2424 {
2425     \bool_gset_true:N \g_unravel_name_in_progress_bool
2426     \_unravel_get_x_non_blank:
2427     \_unravel_scan_file_name_loop:
2428     \bool_gset_false:N \g_unravel_name_in_progress_bool
2429     \_unravel_prev_input_silent:n { ~ }
2430 }
2431 \cs_new_protected:Npn \_unravel_scan_file_name_loop:
2432 {
2433     \_unravel_gtl_if_head_is_definable:NTF \l_unravel_head_gtl
2434     { \_unravel_back_input: }
2435     {
2436         \tl_set:Nx \l_unravel_tmpa_tl
2437         { \_unravel_token_to_char:N \l_unravel_head_token }
2438         \tl_if_eq:NNF \l_unravel_tmpa_tl \c_space_tl
2439         {
2440             \_unravel_prev_input_silent:V \l_unravel_tmpa_tl
2441             \_unravel_get_x_next:
2442             \_unravel_scan_file_name_loop:
2443         }
2444     }
2445 }

```

(End 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.

```

2446 \cs_new_protected:Npn \_unravel_scan_r_token:
2447 {
2448     \bool_do_while:nn
2449     { \tl_if_eq_p:NN \l_unravel_head_tl \c_space_tl }
2450     { \_unravel_get_next: }
2451     \_unravel_gtl_if_head_is_definable:NF \l_unravel_head_gtl
2452     {
2453         \_unravel_error:nnnnn { missing-cs } { } { } { } { } { }

```

```

2454     \__unravel_back_input:
2455     \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2456   }
2457   \__unravel_prev_input_silent:V \l__unravel_head_tl
2458   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2459 }

```

(End definition for __unravel_scan_r_token:.)

__unravel_scan_toks_to_str:

```

2460 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2461 {
2462   \__unravel_prev_input_gpush:
2463   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2464   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2465   \__unravel_prev_input_silent:x
2466   { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2467 }

```

(End definition for __unravel_scan_toks_to_str:.)

__unravel_scan_pdf_ext_toks:

```

2468 \cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
2469 {
2470   \__unravel_prev_input_gpush:
2471   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2472   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2473   \__unravel_prev_input_silent:x
2474   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2475 }

```

(End 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).

```

2476 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2477 {
2478   \bool_if:NT #1 { \__unravel_scan_param: }
2479   \__unravel_scan_left_brace:
2480   \bool_if:NTF #2
2481   { \__unravel_scan_group_x:N #1 }
2482   { \__unravel_scan_group_n:N #1 }
2483 }

```

(End 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.

```

2484 \cs_new_protected:Npn \__unravel_scan_param:
2485 {
2486   \tl_clear:N \l__unravel_tmpa_tl
2487   \__unravel_scan_param_aux:
2488   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl

```

```

2489     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2490   }
2491   \cs_new_protected:Npn \__unravel_scan_param_aux:
2492   {
2493     \__unravel_get_next:
2494     \tl_concat:NNN \l__unravel_tmpa_tl
2495       \l__unravel_tmpa_tl \l__unravel_head_tl
2496     \tl_if_empty:NTF \l__unravel_head_tl
2497       { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2498   }

```

(End 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.

```

2499   \cs_new_protected:Npn \__unravel_scan_group_n:N #1
2500   {
2501     \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
2502     \__unravel_back_input:
2503     \__unravel_input_gpop_item:NF \l__unravel_head_tl
2504     {
2505       \__unravel_error:nnnnn { runaway-text } { } { } { } { }
2506       \__unravel_exit:w
2507     }
2508     \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2509     \bool_if:NT #1
2510       { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2511     \__unravel_prev_input_silent:V \l__unravel_head_tl
2512   }

```

(End 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.

```

2513   \cs_new_protected:Npn \__unravel_scan_group_x:N #1
2514   {
2515     \__unravel_input_gpop_tl:N \l__unravel_head_tl
2516     \__unravel_back_input:V \l__unravel_head_tl
2517     \bool_if:NTF #1
2518     {
2519       \__unravel_prev_input_silent:V \c_left_brace_str
2520       \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2521       \__unravel_scan_group_xdef:n { 1 }
2522     }
2523     {
2524       \__unravel_prev_input_gpush_gtl:
2525       \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2526       \__unravel_scan_group_x:n { 1 }
2527       \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2528       \__unravel_prev_input_silent:x
2529       { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2530     }
2531   }

```

(End 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.

```

2532 \cs_new_protected:Npn \__unravel_scan_group_xdef:n #1
2533 {
2534   \__unravel_get_token_xdef:
2535   \tl_if_empty:NTF \l__unravel_head_tl
2536   {
2537     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2538     {
2539       \__unravel_prev_input_silent:V \c_left_brace_str
2540       \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2541       \__unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2542     }
2543     {
2544       \__unravel_prev_input_silent:V \c_right_brace_str
2545       \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2546       \int_compare:nNnF {#1} = 1
2547       { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2548     }
2549   }
2550   {
2551     \__unravel_prev_input_silent:V \l__unravel_head_tl
2552     \tl_put_right:Nx \l__unravel_defining_tl
2553     { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2554     \__unravel_scan_group_xdef:n {#1}
2555   }
2556 }
2557 \cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }

```

(End definition for `__unravel_scan_group_xdef:n`.)

`__unravel_scan_group_x:n`

```

2558 \cs_new_protected:Npn \__unravel_scan_group_x:n #1
2559 {
2560   \__unravel_get_token_x:
2561   \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2562   \tl_if_empty:NTF \l__unravel_head_tl
2563   {
2564     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2565     { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2566     {
2567       \int_compare:nNnF {#1} = 1
2568       { \__unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2569     }
2570   }
2571   { \__unravel_scan_group_x:n {#1} }
2572 }
2573 \cs_generate_variant:Nn \__unravel_scan_group_x:n { f }

```

(End definition for `__unravel_scan_group_x:n`.)

`__unravel_scan_alt_rule:`

```

2574 \cs_new_protected:Npn \__unravel_scan_alt_rule:
2575 {

```

```

2576 \__unravel_scan_keyword:nTF { wWiIdDtThH }
2577 {
2578   \__unravel_scan_normal_dimen:
2579   \__unravel_scan_alt_rule:
2580 }
2581 {
2582   \__unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2583   {
2584     \__unravel_scan_normal_dimen:
2585     \__unravel_scan_alt_rule:
2586   }
2587   {
2588     \__unravel_scan_keyword:nT { dDeEpPtThH }
2589     {
2590       \__unravel_scan_normal_dimen:
2591       \__unravel_scan_alt_rule:
2592     }
2593   }
2594 }
2595 }

```

(End definition for __unravel_scan_alt_rule:.)

__unravel_scan_spec: Some TeX primitives accept the keywords `to` and `spread`, followed by a dimension.

```

2596 \cs_new_protected:Npn \__unravel_scan_spec:
2597 {
2598   \__unravel_scan_keyword:nTF { tToD } { \__unravel_scan_normal_dimen: }
2599   {
2600     \__unravel_scan_keyword:nT { sSpPrReEaAdD }
2601     { \__unravel_scan_normal_dimen: }
2602   }
2603   \__unravel_scan_left_brace:
2604 }

```

(End 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.

```

2605 \cs_new_protected:Npn \__unravel_do_box:N #1
2606 {
2607   \__unravel_get_x_non_relax:

```

```

2608 \__unravel_set_cmd:
2609 \int_compare:nNnTF
2610 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2611 { \__unravel_do_begin_box:N #1 }
2612 {
2613   \bool_if:NTF #1
2614   {
2615     \int_case:nnTF \l__unravel_head_cmd_int
2616     {
2617       { \__unravel_tex_use:n { hrule } } { }
2618       { \__unravel_tex_use:n { vrule } } { }
2619     }
2620     { \__unravel_do_leaders_rule: }
2621     { \__unravel_do_box_error: }
2622   }
2623   { \__unravel_do_box_error: }
2624 }
2625 }

```

(End 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_EX does), there is no need to remove the after assignment token here.

```

2626 \cs_new_protected:Npn \__unravel_do_box_error:
2627 {
2628   \__unravel_back_input:
2629   \__unravel_error:nnnnn { missing-box } { } { } { } { }
2630   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2631   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2632 }

```

(End 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.

```

2633 \cs_new_protected:Npn \__unravel_do_begin_box:N #1
2634 {
2635   \__unravel_prev_input:V \l__unravel_head_tl
2636   \int_case:nnTF \l__unravel_head_char_int
2637   {
2638     { 0 } { \__unravel_scan_int: } % box
2639     { 1 } { \__unravel_scan_int: } % copy
2640     { 2 } { } % lastbox
2641     { 3 } % vsplit
2642     {
2643       \__unravel_scan_int:
2644       \__unravel_scan_to:
2645       \__unravel_scan_normal_dimen:
2646     }
2647   }

```

```

2648     { \__unravel_do_simple_box:N #1 }
2649     { \__unravel_do_box_explicit:N #1 }
2650   }

```

(End 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_EX run the code and print what we have done. In the case of \shipout, check that \mag has a value between 1 and 32768.

```

2651 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2652 {
2653   \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2654   {
2655     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2656     \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2657     { \__unravel_prepare_mag: }
2658     \tl_use:N \l__unravel_head_tl \scan_stop:
2659     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2660     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2661   }
2662 }

```

(End definition for __unravel_do_simple_box:N.)

__unravel_do_leaders_fetch_skip:

```

2663 \cs_new_protected:Npn \__unravel_do_leaders_fetch_skip:
2664 {
2665   \__unravel_get_x_non_relax:
2666   \__unravel_set_cmd:
2667   \int_compare:nNnTF \l__unravel_head_cmd_int
2668   = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2669   {
2670     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2671     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2672     \__unravel_do_append_glue:
2673   }
2674   {
2675     \__unravel_back_input:
2676     \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2677     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2678     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2679   }
2680 }

```

(End 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_EX 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_EX 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_EX 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.

```

2681 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2682 {
2683   \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2684   { \__unravel_box_hook:N \tex_everyhbox:D }
2685   { \__unravel_box_hook:N \tex_everyvbox:D }
2686   % ^^A todo: TeX calls |normal_paragraph| here.
2687   \__unravel_scan_spec:
2688   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2689   \__unravel_set_action_text:x
2690   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2691   \seq_push:Nf \l__unravel_leaders_box_seq
2692   { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2693   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2694   \gtl_gconcat:NNN \g__unravel_output_gtl
2695   \g__unravel_output_gtl \c_group_begin_gtl
2696   \tl_use:N \l__unravel_head_tl
2697   \c_group_begin_token \__unravel_box_hook_end:
2698 }

```

(End 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_nothing:, used to avoid losing braces.

```

2699 \cs_new_protected:Npn \__unravel_box_hook:N #1
2700 {
2701   \tl_set:NV \l__unravel_tmpa_tl #1
2702   \str_if_eq:eeF
2703   { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2704   {
2705     \__unravel_exp_args:Nx #1
2706     {
2707       \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2708       \exp_not:V #1
2709     }
2710   }
2711   \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2712   {
2713     \exp_args:No #1 {##1}
2714     \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2715     \gtl_clear:N \l__unravel_after_group_gtl
2716     \__unravel_print_action:
2717     \__unravel_back_input:o {##1}
2718     \__unravel_set_action_text:x
2719     { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2720     \tl_if_empty:oF {##1} { \__unravel_print_action: }
2721   }
2722 }
2723 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2724 \cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:

```

(End 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.

```

2725 \cs_new_protected:Npn \__unravel_do_leaders_rule:
2726 {
2727   \__unravel_prev_input:V \l__unravel_head_tl
2728   \__unravel_scan_alt_rule:
2729   \__unravel_do_leaders_fetch_skip:
2730 }

```

(End definition for `__unravel_do_leaders_rule:.`)

2.9 Paragraphs

`__unravel_charcode_if_safe:nTF`

```

2731 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2732 {
2733   \bool_if:nTF
2734   {
2735     \int_compare_p:n { #1 = '!' }
2736     || \int_compare_p:n { ' ' <= #1 <= '[' }
2737     || \int_compare_p:n { #1 = ']' }
2738     || \int_compare_p:n { ' ' <= #1 <= 'z' }
2739   }
2740   { \prg_return_true: }
2741   { \prg_return_false: }
2742 }

```

(End definition for `__unravel_charcode_if_safe:nTF.`)

`__unravel_char:n`
`__unravel_char:V`
`__unravel_char:x`

```

2743 \cs_new_protected:Npn \__unravel_char:n #1
2744 {
2745   \tex_char:D #1 \scan_stop:
2746   \__unravel_charcode_if_safe:nTF {#1}
2747   { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2748   {
2749     \tl_set:Nx \l__unravel_tmpa_tl
2750     { \exp_not:N \char \int_eval:n {#1} ~ }
2751   }
2752   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2753   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2754 }
2755 \cs_generate_variant:Nn \__unravel_char:n { V }
2756 \cs_new_protected:Npn \__unravel_char:x
2757 { \__unravel_exp_args:Nx \__unravel_char:n }

```

(End definition for `__unravel_char:n.`)

`__unravel_char_in_mmode:n`
`__unravel_char_in_mmode:V`
`__unravel_char_in_mmode:x`

```

2758 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2759 {
2760   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2761   { % math active

```

```

2762         \_unravel_exp_args:NNx \gtl_set:Nn \l__unravel_head_gtl
2763         { \char_generate:nn {#1} { 12 } }
2764         \_unravel_back_input:
2765     }
2766     { \_unravel_char:n {#1} }
2767 }
2768 \cs_generate_variant:Nn \_unravel_char_in_mmode:n { V }
2769 \cs_new_protected:Npn \_unravel_char_in_mmode:x
2770 { \_unravel_exp_args:Nx \_unravel_char_in_mmode:n }

```

(End definition for _unravel_char_in_mmode:n.)

_unravel_mathchar:n

_unravel_mathchar:x

```

2771 \cs_new_protected:Npn \_unravel_mathchar:n #1
2772 {
2773     \tex_mathchar:D #1 \scan_stop:
2774     \tl_set:Nx \l__unravel_tmpa_tl
2775     { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2776     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2777     \_unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2778 }
2779 \cs_new_protected:Npn \_unravel_mathchar:x
2780 { \_unravel_exp_args:Nx \_unravel_mathchar:n }

```

(End 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.

```

2781 \cs_new_protected:Npn \_unravel_new_graf:N #1
2782 {
2783     \tl_set:NV \l__unravel_tmpa_tl \_unravel_everypar:w
2784     \_unravel_everypar:w { }
2785     \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2786     \exp_args:NV \_unravel_everypar:w \l__unravel_tmpa_tl
2787     \_unravel_back_input:V \l__unravel_tmpa_tl
2788     \_unravel_print_action:x
2789     {
2790         \g__unravel_action_text_str \c_space_tl : ~
2791         \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2792     }
2793 }

```

(End definition for _unravel_new_graf:N.)

_unravel_end_graf:

```

2794 \cs_new_protected:Npn \_unravel_end_graf:
2795 { \mode_if_horizontal:T { \_unravel_normal_paragraph: } }

```

(End definition for _unravel_end_graf:.)

`__unravel_normal_paragraph:`

```

2796 \cs_new_protected:Npn \__unravel_normal_paragraph:
2797 {
2798   \tex_par:D
2799   \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2800   \__unravel_print_action:x { Paragraph~end. }
2801 }

```

(End definition for __unravel_normal_paragraph:.)

`__unravel_build_page:`

```

2802 \cs_new_protected:Npn \__unravel_build_page:
2803 {
2804 }

```

(End definition for __unravel_build_page:.)

2.10 Groups

`__unravel_handle_right_brace:` When an end-group character is sensed, the result depends on the current group type.

```

2805 \cs_new_protected:Npn \__unravel_handle_right_brace:
2806 {
2807   \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
2808   {
2809     \gtl_gconcat:NNN \g__unravel_output_gtl
2810     \g__unravel_output_gtl \c_group_end_gtl
2811     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2812     \int_case:nn \__unravel_currentgrouptype:
2813     {
2814       { 1 } { \__unravel_end_simple_group: } % simple
2815       { 2 } { \__unravel_end_box_group: } % hbox
2816       { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2817       { 4 } { \__unravel_end_graf: \__unravel_end_box_group: } % vbox
2818       { 5 } { \__unravel_end_graf: \__unravel_end_box_group: } % vtop
2819       { 6 } { \__unravel_end_align_group: } % align
2820       { 7 } { \__unravel_end_no_align_group: } % no_align
2821       { 8 } { \__unravel_end_output_group: } % output
2822       { 9 } { \__unravel_end_simple_group: } % math
2823       { 10 } { \__unravel_end_disc_group: } % disc
2824       { 11 } { \__unravel_end_graf: \__unravel_end_simple_group: } % insert
2825       { 12 } { \__unravel_end_graf: \__unravel_end_simple_group: } % vcenter
2826       { 13 } { \__unravel_end_math_choice_group: } % math_choice
2827     }
2828   }
2829   { % bottom_level, semi_simple, math_shift, math_left
2830     \l__unravel_head_token
2831     \__unravel_print_action:
2832   }
2833 }

```

(End 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.

```

2834 \cs_new_protected:Npn \__unravel_end_simple_group:
2835 {
2836   \l__unravel_head_token
2837   \__unravel_print_action:
2838 }

```

(End 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.

```

2839 \cs_new_protected:Npn \__unravel_end_box_group:
2840 {
2841   \seq_pop:Nn \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2842   \exp_args:No \__unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
2843 }
2844 \cs_new_protected:Npn \__unravel_end_box_group_aux:n #1
2845 {
2846   \str_if_eq:eeTF {#1} { Z }
2847   { \__unravel_end_simple_group: }
2848   {
2849     \__unravel_get_x_non_relax:
2850     \__unravel_set_cmd:
2851     \int_compare:nNnTF \l__unravel_head_cmd_int
2852     = { \__unravel_tex_use:n { #1 skip } }
2853     {
2854       \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2855       \__unravel_do_append_glue:
2856     }
2857     {
2858       \__unravel_back_input:
2859       \c_group_end_token \group_begin: \group_end:
2860       \__unravel_print_action:
2861     }
2862   }
2863 }

```

(End definition for __unravel_end_box_group:.)

`__unravel_off_save:`

```

2864 \cs_new_protected:Npn \__unravel_off_save:
2865 {
2866   \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
2867   { % bottom-level
2868     \__unravel_error:nxxxx { extra-close }
2869     { \token_to_meaning:N \l__unravel_head_token } { } { } { }
2870   }
2871   {
2872     \__unravel_back_input:
2873     \int_case:nnF \__unravel_currentgrouptype:
2874     {
2875       { 14 } % semi_simple_group
2876       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }

```

```

2877         { 15 } % math_shift_group
2878         { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2879         { 16 } % math_left_group
2880         { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2881     }
2882     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2883     \__unravel_back_input:
2884     \__unravel_error:nxxxx { off-save }
2885     { \gtl_to_str:N \l__unravel_head_gtl } { } { } { }
2886 }
2887 }

```

(End definition for __unravel_off_save:.)

2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
2888 \cs_new_protected:Npn \__unravel_mode_math:n #1
2889 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
2890 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
2891 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
2892 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
2893 {
2894     \mode_if_math:TF
2895     { \__unravel_insert_dollar_error: }
2896     { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
2897 }
2898 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
2899 {
2900     \mode_if_vertical:TF
2901     { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
2902     {#1}
2903 }

```

(End definition for __unravel_mode_math:n, __unravel_mode_non_math:n, and __unravel_mode_vertical:n.)

__unravel_head_for_vmode: See T_EX's head_for_vmode.

```

2904 \cs_new_protected:Npn \__unravel_head_for_vmode:
2905 {
2906     \mode_if_inner:TF
2907     {
2908         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
2909         {
2910             \__unravel_error:nnnnn { hrulerule-bad-mode } { } { } { } { }
2911             \__unravel_print_action:
2912         }
2913         { \__unravel_off_save: }
2914     }
2915     {
2916         \__unravel_back_input:
2917         \gtl_set:Nn \l__unravel_head_gtl { \par }
2918         \__unravel_back_input:
2919     }
2920 }

```

(End definition for _unravel_head_for_vmode:.)

_unravel_goto_inner_math:

```

2921 \cs_new_protected:Npn \_unravel_goto_inner_math:
2922 {
2923   \_unravel_box_hook:N \tex_everymath:D
2924   $ % $
2925   \_unravel_box_hook_end:
2926 }

```

(End definition for _unravel_goto_inner_math:.)

_unravel_goto_display_math:

```

2927 \cs_new_protected:Npn \_unravel_goto_display_math:
2928 {
2929   \_unravel_box_hook:N \tex_everydisplay:D
2930   $ $
2931   \_unravel_box_hook_end:
2932 }

```

(End definition for _unravel_goto_display_math:.)

_unravel_after_math:

```

2933 \cs_new_protected:Npn \_unravel_after_math:
2934 {
2935   \mode_if_inner:TF
2936   {
2937     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2938     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2939     $ % $
2940   }
2941   {
2942     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2943     \_unravel_get_x_next:
2944     \token_if_eq_catcode:NNF
2945       \l__unravel_head_token \c_math_toggle_token
2946     {
2947       \_unravel_back_input:
2948       \tl_set:Nn \l__unravel_head_tl { $ } % $
2949       \_unravel_error:nnnnn { missing-dollar } { } { } { } { }
2950     }
2951     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2952     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2953     $ $
2954   }
2955   \_unravel_print_action:
2956 }

```

(End definition for _unravel_after_math:.)

2.12 One step

`__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.

```
2957 \cs_new_protected:Npn \__unravel_do_step:
2958 {
2959   \__unravel_set_action_text:
2960   \bool_if:NT \g__unravel_internal_debug_bool
2961     { \__unravel_exp_args:Nx \iow_term:n { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int
2962   \cs_if_exist_use:cF
2963     { \__unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2964     { \__unravel_error:nxxxx { internal } { unknown-command } { } { } { } }
2965 }
```

(End definition for __unravel_do_step:.)

2.13 Commands

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

2.13.1 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 `\crrc`.
- `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.

```

2966 \__unravel_new_tex_cmd:nn { relax } % 0
2967 {
2968   \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:
2969   {
2970     \exp_after:wN \__unravel_token_if_expandable:NNTF \l__unravel_head_tl
2971     {
2972       \__unravel_set_action_text:x
2973       { \iow_char:N \notexpanded: \g__unravel_action_text_str }
2974     }
2975   }
2976 }
2977 \__unravel_print_action:
2978 }

```

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.

```

2979 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
2980 {
2981   \gtl_gconcat:NNN \g__unravel_output_gtl
2982   \g__unravel_output_gtl \c_group_begin_gtl
2983   \__unravel_print_action:
2984   \l__unravel_head_token
2985   \gtl_clear:N \l__unravel_after_group_gtl
2986 }
2987 \__unravel_new_tex_cmd:nn { end-group_char } % 2
2988 { \__unravel_handle_right_brace: }

```

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

```

2989 \__unravel_new_tex_cmd:nn { math_char } % 3
2990 {
2991   \__unravel_mode_non_vertical:n
2992   {
2993     \mode_if_math:TF
2994     {
2995       \int_compare:nNnTF
2996       \__unravel_currentgrouptype: = { 15 } % math_shift_group
2997       { \__unravel_after_math: }
2998       { \__unravel_off_save: }
2999     }
3000     {
3001       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3002       \__unravel_get_next:
3003       \token_if_eq_catcode:NNTF
3004       \l__unravel_head_token \c_math_toggle_token
3005       {
3006         \mode_if_inner:TF
3007         { \__unravel_back_input: \__unravel_goto_inner_math: }
3008         {
3009           \gtl_gput_right:NV
3010           \g__unravel_output_gtl \l__unravel_head_tl
3011           \__unravel_goto_display_math:
3012         }
3013       }
3014     }
3015   }

```

```

3013     }
3014     { \_unravel_back_input: \_unravel_goto_inner_math: }
3015   }
3016 }
3017 }

```

Some commands are errors when they reach T_EX's stomach. Among others, `tab_mark=alignment_char`, `car_ret` and `macro_char`. We let T_EX insert the proper error.

```

3018 \_unravel_new_tex_cmd:nn { alignment_char } % 4
3019 { \l\_unravel_head_token \_unravel_print_action: }
3020 \_unravel_new_tex_cmd:nn { car_ret } % 5
3021 { \l\_unravel_head_token \_unravel_print_action: }
3022 \_unravel_new_tex_cmd:nn { macro_char } % 6
3023 { \l\_unravel_head_token \_unravel_print_action: }
3024 \_unravel_new_tex_cmd:nn { superscript_char } % 7
3025 { \_unravel_mode_math:n { \_unravel_sub_sup: } }
3026 \_unravel_new_tex_cmd:nn { subscript_char } % 8
3027 { \_unravel_mode_math:n { \_unravel_sub_sup: } }
3028 \cs_new_protected:Npn \_unravel_sub_sup:
3029 {
3030   \_unravel_prev_input_gpush:N \l\_unravel_head_tl
3031   \_unravel_print_action:
3032   \_unravel_get_x_non_relax:
3033   \_unravel_set_cmd:
3034   \int_case:nnTF \l\_unravel_head_cmd_int
3035   {
3036     { \_unravel_tex_use:n { the_char } }
3037     { \_unravel_prev_input:V \l\_unravel_head_tl }
3038     { \_unravel_tex_use:n { other_char } }
3039     { \_unravel_prev_input:V \l\_unravel_head_tl }
3040     { \_unravel_tex_use:n { char_given } }
3041     { \_unravel_prev_input:V \l\_unravel_head_tl }
3042     { \_unravel_tex_use:n { char_num } }
3043     {
3044       \_unravel_prev_input:V \l\_unravel_head_tl
3045       \_unravel_scan_int:
3046     }
3047     { \_unravel_tex_use:n { math_char_num } }
3048     {
3049       \_unravel_prev_input:V \l\_unravel_head_tl
3050       \_unravel_scan_int:
3051     }
3052     { \_unravel_tex_use:n { math_given } }
3053     { \_unravel_prev_input:V \l\_unravel_head_tl }
3054     { \_unravel_tex_use:n { delim_num } }
3055     { \_unravel_prev_input:V \l\_unravel_head_tl \_unravel_scan_int: }
3056   }
3057   {
3058     \_unravel_prev_input_gpop:N \l\_unravel_head_tl
3059     \gtl_gput_right:NV \g\_unravel_output_gtl \l\_unravel_head_tl
3060     \tl_use:N \l\_unravel_head_tl \scan_stop:
3061   }
3062   {

```

```

3063     \__unravel_back_input:
3064     \__unravel_scan_left_brace:
3065     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3066     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3067     \gtl_gconcat:NNN \g__unravel_output_gtl
3068     \g__unravel_output_gtl \c_group_begin_gtl
3069     \tl_use:N \l__unravel_head_tl \c_group_begin_token
3070   }
3071   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3072 }

```

```

3073 \__unravel_new_tex_cmd:nn { endv } % 9
3074 { \__unravel_not_implemented:n { alignments } }

```

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.

```

3075 \__unravel_new_tex_cmd:nn { blank_char } % 10
3076 {
3077   \mode_if_horizontal:T
3078   {
3079     \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
3080     \l__unravel_head_token
3081   }
3082   \__unravel_print_action:
3083 }

```

Letters and other characters leave vertical mode.

```

3084 \__unravel_new_tex_cmd:nn { the_char } % 11
3085 {
3086   \__unravel_mode_non_vertical:n
3087   {
3088     \tl_set:Nx \l__unravel_tmpa_tl
3089     { ' \__unravel_token_to_char:N \l__unravel_head_token }
3090     \mode_if_math:TF
3091     { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
3092     { \__unravel_char:V \l__unravel_tmpa_tl }
3093   }
3094 }

```

```

3095 \__unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12

```

```

3096 \__unravel_new_tex_cmd:nn { par_end } % 13
3097 {
3098   \__unravel_mode_non_math:n
3099   {
3100     \mode_if_vertical:TF
3101     { \__unravel_normal_paragraph: }
3102     {
3103       % if align_state<0 then off_save;
3104       \__unravel_end_graf:
3105       \mode_if_vertical:T
3106       { \mode_if_inner:F { \__unravel_build_page: } }
3107     }
3108   }
3109 }

```

```

3110 \__unravel_new_tex_cmd:nn { stop } % 14

```

```

3111 {
3112   \__unravel_mode_vertical:n
3113   {
3114     \mode_if_inner:TF
3115     { \__unravel_forbidden_case: }
3116     {
3117       % ^^A todo: unless its_all_over
3118       \int_gdecr:N \g__unravel_ends_int
3119       \int_compare:nNnTF \g__unravel_ends_int > 0
3120       {
3121         \__unravel_back_input:
3122         \__unravel_back_input:n
3123         {
3124           \__unravel_hbox:w to \tex_hsize:D { }
3125           \tex_vfill:D
3126           \tex_penalty:D - '10000000000 ~
3127         }
3128         \__unravel_build_page:
3129         \__unravel_print_action:x { End-everything! }
3130       }
3131       {
3132         \__unravel_print_outcome:
3133         \l__unravel_head_token
3134       }
3135     }
3136   }
3137 }
3138 \__unravel_new_tex_cmd:nn { delim_num } % 15
3139 {
3140   \__unravel_mode_math:n
3141   {
3142     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3143     \__unravel_print_action:
3144     \__unravel_scan_int:
3145     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3146     \tl_use:N \l__unravel_head_tl \scan_stop:
3147     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3148   }
3149 }

```

2.13.2 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`.

```

3150 \__unravel_new_tex_cmd:nn { char_num } % 16
3151 {
3152   \__unravel_mode_non_vertical:n
3153   {
3154     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3155     \__unravel_print_action:
3156     \__unravel_scan_int:
3157     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3158     \mode_if_math:TF
3159     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
3160     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
3161   }
3162 }

```

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.

```

3163 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3164 {
3165   \__unravel_mode_math:n
3166   {
3167     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3168     \__unravel_print_action:
3169     \__unravel_scan_int:
3170     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3171     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
3172   }
3173 }

3174 \__unravel_new_tex_cmd:nn { mark } % 18
3175 {
3176   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3177   \__unravel_print_action:

```

```

3178 \int_compare:nNnF \l__unravel_head_char_int = 0
3179 { \__unravel_scan_int: }
3180 \__unravel_prev_input_gpush:
3181 \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3182 \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3183 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3184 \__unravel_print_action:x
3185 { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3186 \tl_put_right:Nx \l__unravel_head_tl
3187 { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3188 \tl_use:N \l__unravel_head_tl
3189 }

```

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.

```

3190 \__unravel_new_tex_cmd:nn { xray } % 19
3191 {
3192   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3193   \__unravel_print_action:
3194   \int_case:nnF \l__unravel_head_char_int
3195   {
3196     { 0 }
3197     { % show
3198       \__unravel_get_next:
3199       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3200       \token_if_eq_meaning:NNTF
3201       \l__unravel_head_token \__unravel_special_relax:
3202       {
3203         \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
3204         \exp_after:wN \exp_not:N \l__unravel_head_tl
3205       }
3206       { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
3207     }
3208     { 2 }
3209     { % showthe
3210       \__unravel_get_x_next:
3211       \__unravel_scan_something_internal:n { 5 }
3212       \__unravel_prev_input_gpop:N \l__unravel_head_tl
3213       \__unravel_exp_args:Nx \use:n
3214       { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3215     }
3216   }
3217   { % no operand for showlists, showgroups, showifs
3218     \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
3219     { \__unravel_scan_int: }
3220     \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
3221     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3222     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3223     \tl_use:N \l__unravel_head_tl \scan_stop:
3224   }

```

```

3225 }
      make_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
3226 \__unravel_new_tex_cmd:nn { make_box } % 20
3227 {
3228   \__unravel_prev_input_gpush:
3229   \__unravel_back_input:
3230   \__unravel_do_box:N \c_false_bool
3231 }

```

__unravel_do_move: Scan a dimension and a box, and perform the shift, printing the appropriate action.

```

3232 \cs_new_protected:Npn \__unravel_do_move:
3233 {
3234   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3235   \__unravel_print_action:
3236   \__unravel_scan_normal_dimen:
3237   \__unravel_do_box:N \c_false_bool
3238 }

```

(End definition for __unravel_do_move:.)

hmove=21 for \moveright and \moveleft.

```

3239 \__unravel_new_tex_cmd:nn { hmove } % 21
3240 {
3241   \mode_if_vertical:TF
3242   { \__unravel_do_move: } { \__unravel_forbidden_case: }
3243 }

```

vmove=22 for \lower and \raise.

```

3244 \__unravel_new_tex_cmd:nn { vmove } % 22
3245 {
3246   \mode_if_vertical:TF
3247   { \__unravel_forbidden_case: } { \__unravel_do_move: }
3248 }

```

__unravel_do_unpackage:

```

3249 \cs_new_protected:Npn \__unravel_do_unpackage:
3250 {
3251   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3252   \__unravel_print_action:
3253   \__unravel_scan_int:
3254   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3255   \tl_use:N \l__unravel_head_tl \scan_stop:
3256   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3257 }

```

(End definition for __unravel_do_unpackage:.)

un_hbox=23 for \unhbox and \unhcopy.

```

3258 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3259 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

```

unvbox=24 for \unvbox, \unvcopy, \pagediscards, and \splittediscards. The latter two take no operands, so we just let T_EX do its thing, then we show the action.

```

3260 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3261 {
3262   \__unravel_mode_vertical:n

```

```

3263     {
3264         \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3265         { \l__unravel_head_token \__unravel_print_action: }
3266         { \__unravel_do_unpackage: }
3267     }
3268 }

```

remove_item=25 for \unpenalty (12), \unkern (11), \unskip (10). Those commands only act on TeX's box/glue data structures, which unravel does not (and cannot) care about.

```

3269 \__unravel_new_tex_cmd:nn { remove_item } % 25
3270 { \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.

```

3271 \cs_new_protected:Npn \__unravel_do_append_glue:
3272 {
3273     \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3274     { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3275     {
3276         \__unravel_prev_input_gpush:N \l__unravel_head_tl
3277         \__unravel_print_action:
3278         \exp_args:Nf \__unravel_scan_glue:n
3279         { \int_eval:n { \l__unravel_head_char_int - 2 } }
3280         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3281         \tl_use:N \l__unravel_head_tl \scan_stop:
3282         \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3283     }
3284 }

```

(End definition for __unravel_do_append_glue:.)

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

```

3285 \__unravel_new_tex_cmd:nn { hskip } % 26
3286 { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }

```

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

```

3287 \__unravel_new_tex_cmd:nn { vskip } % 27
3288 { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }

```

mskip=28 for \mskip (5).

```

3289 \__unravel_new_tex_cmd:nn { mskip } % 28
3290 { \__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.

```

3291 \cs_new_protected:Npn \__unravel_do_append_kern:
3292 {
3293     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3294     \__unravel_print_action:
3295     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
3296     { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
3297     { \__unravel_scan_dimen:nN { 3 } \c_false_bool }

```

```

3298     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3299     \tl_use:N \l__unravel_head_tl \scan_stop:
3300     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3301   }

(End definition for \__unravel_do_append_kern:.)
kern=29 for \kern (1).

3302 \__unravel_new_tex_cmd:nn { kern } % 29
3303 { \__unravel_do_append_kern: }

mkern=30 for \mkern (99).

3304 \__unravel_new_tex_cmd:nn { mkern } % 30
3305 { \__unravel_mode_math:n { \__unravel_do_append_kern: } }

leader_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).

3306 \__unravel_new_tex_cmd:nn { leader_ship } % 31
3307 {
3308     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3309     \__unravel_print_action:
3310     \__unravel_do_box:N \c_true_bool
3311 }

```

2.13.3 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

```

3312 \_unravel_new_tex_cmd:nn { halign } % 32
3313 { \_unravel_not_implemented:n { halign } }
3314 \_unravel_new_tex_cmd:nn { valign } % 33
3315 { \_unravel_not_implemented:n { valign } }
3316 \_unravel_new_tex_cmd:nn { no_align } % 34
3317 { \_unravel_not_implemented:n { noalign } }

3318 \_unravel_new_tex_cmd:nn { vrule } % 35
3319 { \_unravel_mode_non_vertical:n { \_unravel_do_rule: } }
3320 \_unravel_new_tex_cmd:nn { hrule } % 36
3321 { \_unravel_mode_vertical:n { \_unravel_do_rule: } }
3322 \cs_new_protected:Npn \_unravel_do_rule:
3323 {
3324   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3325   \_unravel_print_action:
3326   \_unravel_scan_alt_rule:
3327   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3328   \tl_use:N \l__unravel_head_tl \scan_stop:
3329   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3330 }

3331 \_unravel_new_tex_cmd:nn { insert } % 37
3332 { \_unravel_begin_insert_or_adjust: }
3333 \_unravel_new_tex_cmd:nn { vadjust } % 38
3334 {
3335   \mode_if_vertical:TF
3336   { \_unravel_forbidden_case: } { \_unravel_begin_insert_or_adjust: }
3337 }

3338 \_unravel_new_tex_cmd:nn { ignore_spaces } % 39
3339 {
3340   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3341   {
3342     \_unravel_print_action:
3343     \_unravel_get_x_non_blank:
3344     \_unravel_set_cmd:
3345     \_unravel_do_step:
3346   }
3347   { \_unravel_not_implemented:n { pdfprimitive } }
3348 }

3349 \_unravel_new_tex_cmd:nn { after_assignment } % 40
3350 {
3351   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3352   \_unravel_get_next:
3353   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3354   \_unravel_print_action:x
3355   {
3356     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3357     \gtl_to_str:N \l__unravel_head_gtl
3358   }
3359 }

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.

3360 \_unravel_new_tex_cmd:nn { after_group } % 41
3361 {

```

```

3362 \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3363 \__unravel_get_next:
3364 \int_compare:nNnTF \__unravel_currentgroupstype: = 0
3365 {
3366   \__unravel_print_action:x
3367   {
3368     Aftergroup~(level~0~=>~dropped):~
3369     \tl_to_str:N \l__unravel_tmpa_tl
3370     \gtl_to_str:N \l__unravel_head_gtl
3371   }
3372 }
3373 {
3374   \gtl_concat:NNN \l__unravel_after_group_gtl
3375   \l__unravel_after_group_gtl \l__unravel_head_gtl
3376   \__unravel_print_action:x
3377   {
3378     Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3379     \gtl_to_str:N \l__unravel_head_gtl
3380   }
3381 }
3382 }
See \__unravel_do_append_glue:.
3383 \__unravel_new_tex_cmd:nn { break_penalty } % 42
3384 {
3385   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3386   \__unravel_print_action:
3387   \__unravel_scan_int:
3388   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3389   \tl_use:N \l__unravel_head_tl \scan_stop:
3390   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3391 }
3392 \__unravel_new_tex_cmd:nn { start_par } % 43
3393 {
3394   \mode_if_vertical:TF
3395   {
3396     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3397     { \__unravel_new_graf:N \c_false_bool }
3398     { \__unravel_new_graf:N \c_true_bool }
3399   }
3400   {
3401     \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3402     {
3403       \__unravel_hbox:w width \tex_parindent:D { }
3404       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3405     }
3406     \__unravel_print_action:
3407   }
3408 }
3409 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3410 {
3411   \mode_if_vertical:TF { \__unravel_forbidden_case: }
3412   { \l__unravel_head_token \__unravel_print_action: }
3413 }

```

`__unravel_do_accent:`

```

3414 \cs_new_protected:Npn \__unravel_do_accent:
3415 {
3416   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3417   \__unravel_print_action:
3418   \__unravel_scan_int:
3419   \__unravel_do_assignments:
3420   \bool_if:nTF
3421     {
3422       \token_if_eq_catcode_p:NN
3423         \l__unravel_head_token \c_catcode_letter_token
3424       ||
3425       \token_if_eq_catcode_p:NN
3426         \l__unravel_head_token \c_catcode_other_token
3427       ||
3428       \int_compare_p:nNn
3429         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3430     }
3431   { \__unravel_prev_input:V \l__unravel_head_tl }
3432   {
3433     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3434     {
3435       \__unravel_prev_input:V \l__unravel_head_tl
3436       \__unravel_scan_int:
3437     }
3438     { \__unravel_break:w }
3439   }
3440   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3441   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3442   \tl_use:N \l__unravel_head_tl \scan_stop:
3443   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3444   \__unravel_break_point:
3445 }

```

(End definition for __unravel_do_accent:.)

`__unravel_do_math_accent:` `TeX` will complain if `\l__unravel_head_tl` happens to start with `\accent` (the user used `\accent` in math mode).

```

3446 \cs_new_protected:Npn \__unravel_do_math_accent:
3447 {
3448   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3449   \__unravel_print_action:
3450   \__unravel_scan_int:
3451   \__unravel_scan_math:
3452   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3453   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3454   \tl_use:N \l__unravel_head_tl \scan_stop:
3455   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3456 }

```

(End definition for __unravel_do_math_accent:.)

```

3457 \__unravel_new_tex_cmd:nn { accent } % 45
3458 {
3459   \__unravel_mode_non_vertical:n

```

```

3460     {
3461         \mode_if_math:TF
3462         { \__unravel_do_math_accent: } { \__unravel_do_accent: }
3463     }
3464 }

3465 \__unravel_new_tex_cmd:nn { math_accent } % 46
3466 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }

3467 \__unravel_new_tex_cmd:nn { discretionary } % 47
3468 { \__unravel_not_implemented:n { discretionary } }

```

2.13.4 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

```

3469 \__unravel_new_tex_cmd:nn { eq_no } % 48
3470 { \__unravel_not_implemented:n { eqno } }

3471 \__unravel_new_tex_cmd:nn { left_right } % 49
3472 { \__unravel_not_implemented:n { left/right } }

3473 \__unravel_new_tex_cmd:nn { math_comp } % 50
3474 { \__unravel_not_implemented:n { math~comp } }

3475 \__unravel_new_tex_cmd:nn { limit_switch } % 51
3476 { \__unravel_not_implemented:n { limits } }

3477 \__unravel_new_tex_cmd:nn { above } % 52
3478 { \__unravel_not_implemented:n { above } }

3479 \__unravel_new_tex_cmd:nn { math_style } % 53
3480 { \__unravel_not_implemented:n { math~style } }

3481 \__unravel_new_tex_cmd:nn { math_choice } % 54
3482 { \__unravel_not_implemented:n { math~choice } }

3483 \__unravel_new_tex_cmd:nn { non_script } % 55
3484 { \__unravel_not_implemented:n { non~script } }

3485 \__unravel_new_tex_cmd:nn { vcenter } % 56
3486 { \__unravel_not_implemented:n { vcenter } }

```

2.13.5 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

```

3487 \__unravel_new_tex_cmd:nn { case_shift } % 57
3488 {
3489   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3490   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3491   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3492   \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
3493 }
3494 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
3495 {
3496   #1 { \__unravel_back_input:n {#2} }
3497   \__unravel_print_action:x
3498   { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3499 }
3500 \__unravel_new_tex_cmd:nn { message } % 58
3501 {
3502   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3503   \__unravel_print_action:
3504   \__unravel_scan_toks_to_str:
3505   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3506   \tl_use:N \l__unravel_head_tl
3507   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3508 }

```

Extensions are implemented in a later section.

```

3509 \__unravel_new_tex_cmd:nn { extension } % 59
3510 {
3511   \__unravel_prev_input_gpush:N \l__unravel_head_tl

```

```

3512 \__unravel_print_action:
3513 \__unravel_scan_extension_operands:
3514 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3515 \tl_use:N \l__unravel_head_tl \scan_stop:
3516 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3517 }
3518 \__unravel_new_tex_cmd:nn { in_stream } % 60
3519 {
3520 \__unravel_prev_input_gpush:N \l__unravel_head_tl
3521 \__unravel_print_action:
3522 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3523 {
3524 \__unravel_scan_int:
3525 \__unravel_scan_optional_equals:
3526 \__unravel_scan_file_name:
3527 }
3528 { \__unravel_scan_int: }
3529 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3530 \tl_use:N \l__unravel_head_tl \scan_stop:
3531 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3532 }
3533 \__unravel_new_tex_cmd:nn { begin_group } % 61
3534 {
3535 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3536 \l__unravel_head_token
3537 \gtl_clear:N \l__unravel_after_group_gtl
3538 \__unravel_print_action:
3539 }
3540 \__unravel_new_tex_cmd:nn { end_group } % 62
3541 {
3542 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3543 \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3544 \l__unravel_head_token
3545 \__unravel_print_action:
3546 }
3547 \__unravel_new_tex_cmd:nn { omit } % 63
3548 { \__unravel_not_implemented:n { omit } }
3549 \__unravel_new_tex_cmd:nn { ex_space } % 64
3550 {
3551 \__unravel_mode_non_vertical:n
3552 { \l__unravel_head_token \__unravel_print_action: }
3553 }
3554 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3555 {
3556 \__unravel_mode_non_vertical:n
3557 { \l__unravel_head_token \__unravel_print_action: }
3558 }
3559 \__unravel_new_tex_cmd:nn { radical } % 66
3560 {
3561 \__unravel_mode_math:n
3562 {
3563 \__unravel_prev_input_gpush:N \l__unravel_head_tl

```

```

3564     \_unravel_print_action:
3565     \_unravel_scan_int:
3566     \_unravel_scan_math:
3567     \_unravel_prev_input_gpop:N \l__unravel_head_tl
3568     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3569     \tl_use:N \l__unravel_head_tl \scan_stop:
3570     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3571   }
3572 }

3573 \_unravel_new_tex_cmd:nn { end_cs_name } % 67
3574 {
3575   \_unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
3576   \_unravel_print_action:
3577 }

See the_char and other_char.

3578 \_unravel_new_tex_cmd:nn { char_given } % 68
3579 {
3580   \_unravel_mode_non_vertical:n
3581   {
3582     \mode_if_math:TF
3583     { \_unravel_char_in_mmode:V \l__unravel_head_char_int }
3584     { \_unravel_char:V \l__unravel_head_char_int }
3585   }
3586 }

See math_char_num.

3587 \_unravel_new_tex_cmd:nn { math_given } % 69
3588 {
3589   \_unravel_mode_math:n
3590   { \_unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3591 }

3592 \_unravel_new_tex_cmd:nn { last_item } % 70
3593 { \_unravel_forbidden_case: }

```

2.13.6 Extensions

_unravel_scan_extension_operands:

```

3594 \cs_new_protected:Npn \_unravel_scan_extension_operands:
3595 {
3596   \int_case:nnF \l__unravel_head_char_int
3597   {
3598     { 0 } % openout
3599     {
3600       \_unravel_scan_int:
3601       \_unravel_scan_optional_equals:
3602       \_unravel_scan_file_name:
3603     }
3604     { 1 } % write
3605     {
3606       \_unravel_scan_int:
3607       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3608     }
3609     { 2 } % closeout

```

```

3610     { \_unravel\_scan\_int: }
3611 { 3 } % special
3612     { \_unravel\_scan\_toks\_to\_str: }
3613 { 4 } % immediate
3614     { \_unravel\_scan\_immediate\_operands: }
3615 { 5 } % setlanguage
3616     {
3617         \mode\_if\_horizontal:TF
3618         { \_unravel\_scan\_int: }
3619         { \_unravel\_error:nnnnn { invalid-mode } { } { } { } { } }
3620     }
3621 { 6 } % pdfliteral
3622     {
3623         \_unravel\_scan\_keyword:nF { dDiIrReEcCtT }
3624         { \_unravel\_scan\_keyword:n { pPaAgGeE } }
3625         \_unravel\_scan\_pdf\_ext\_toks:
3626     }
3627 { 7 } % pdfobj
3628     {
3629         \_unravel\_scan\_keyword:nTF
3630         { rReEsSeErRvVeEoObBjJnNuUmM }
3631         { \_unravel\_skip\_optional\_space: }
3632         {
3633             \_unravel\_scan\_keyword:nF { uUsSeEoObBjJnNuUmM }
3634             { \_unravel\_scan\_int: }
3635             \_unravel\_scan\_keyword:nT { sStTrReEaAmM }
3636             {
3637                 \_unravel\_scan\_keyword:nT { aAtTtTrR }
3638                 { \_unravel\_scan\_pdf\_ext\_toks: }
3639             }
3640             \_unravel\_scan\_keyword:n { fFiIlLeE }
3641             \_unravel\_scan\_pdf\_ext\_toks:
3642         }
3643     }
3644 { 8 } % pdfrefobj
3645     { \_unravel\_scan\_int: }
3646 { 9 } % pdfxform
3647     {
3648         \_unravel\_scan\_keyword:nT { aAtTtTrR }
3649         { \_unravel\_scan\_pdf\_ext\_toks: }
3650         \_unravel\_scan\_keyword:nTF { rReEsSoOuUrRcCeEsS }
3651         { \_unravel\_scan\_pdf\_ext\_toks: }
3652         \_unravel\_scan\_int:
3653     }
3654 { 10 } % pdfrefxform
3655     { \_unravel\_scan\_int: }
3656 { 11 } % pdfximage
3657     { \_unravel\_scan\_image: }
3658 { 12 } % pdfrefximage
3659     { \_unravel\_scan\_int: }
3660 { 13 } % pdfannot
3661     {
3662         \_unravel\_scan\_keyword:nTF
3663         { rReEsSeErRvVeEoObBjJnNuUmM }

```

```

3664         { \__unravel_scan_optional_space: }
3665         {
3666             \__unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3667             { \__unravel_scan_int: }
3668             \__unravel_scan_alt_rule:
3669             \__unravel_scan_pdf_ext_toks:
3670         }
3671     }
3672 { 14 } % pdfstartlink
3673 {
3674     \mode_if_vertical:TF
3675     { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3676     {
3677         \__unravel_scan_rule_attr:
3678         \__unravel_scan_action:
3679     }
3680 }
3681 { 15 } % pdfendlink
3682 {
3683     \mode_if_vertical:T
3684     { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3685 }
3686 { 16 } % pdfoutline
3687 {
3688     \__unravel_scan_keyword:nT { aAtTtTrR }
3689     { \__unravel_scan_pdf_ext_toks: }
3690     \__unravel_scan_action:
3691     \__unravel_scan_keyword:nT { cCoOuUnNtT }
3692     { \__unravel_scan_int: }
3693     \__unravel_scan_pdf_ext_toks:
3694 }
3695 { 17 } % pdfdest
3696 { \__unravel_scan_pdfdest_operands: }
3697 { 18 } % pdfthread
3698 { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
3699 { 19 } % pdfstartthread
3700 { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
3701 { 20 } % pdfendthread
3702 { }
3703 { 21 } % pdfsavepos
3704 { }
3705 { 22 } % pdfinfo
3706 { \__unravel_scan_pdf_ext_toks: }
3707 { 23 } % pdfcatalog
3708 {
3709     \__unravel_scan_pdf_ext_toks:
3710     \__unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
3711     { \__unravel_scan_action: }
3712 }
3713 { 24 } % pdfnames
3714 { \__unravel_scan_pdf_ext_toks: }
3715 { 25 } % pdffontattr
3716 {
3717     \__unravel_scan_font_ident:

```

```

3718         \_unravel_scan_pdf_ext_toks:
3719     }
3720 { 26 } % pdfincludechars
3721 {
3722     \_unravel_scan_font_ident:
3723     \_unravel_scan_pdf_ext_toks:
3724 }
3725 { 27 } % pdfmapfile
3726 { \_unravel_scan_pdf_ext_toks: }
3727 { 28 } % pdfmapline
3728 { \_unravel_scan_pdf_ext_toks: }
3729 { 29 } % pdftrailer
3730 { \_unravel_scan_pdf_ext_toks: }
3731 { 30 } % pdfresettimer
3732 { }
3733 { 31 } % pdffontexpand
3734 {
3735     \_unravel_scan_font_ident:
3736     \_unravel_scan_optional_equals:
3737     \_unravel_scan_int:
3738     \_unravel_scan_int:
3739     \_unravel_scan_int:
3740     \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3741     { \_unravel_skip_optional_space: }
3742 }
3743 { 32 } % pdfsetrandomseed
3744 { \_unravel_scan_int: }
3745 { 33 } % pdfsnaprefpoint
3746 { }
3747 { 34 } % pdfsnapy
3748 { \_unravel_scan_normal_glue: }
3749 { 35 } % pdfsnapycomp
3750 { \_unravel_scan_int: }
3751 { 36 } % pdfglyphptounicode
3752 {
3753     \_unravel_scan_pdf_ext_toks:
3754     \_unravel_scan_pdf_ext_toks:
3755 }
3756 { 37 } % pdfcolorstack
3757 { \_unravel_scan_pdfcolorstack_operands: }
3758 { 38 } % pdfsetmatrix
3759 { \_unravel_scan_pdf_ext_toks: }
3760 { 39 } % pdfsave
3761 { }
3762 { 40 } % pdfrestore
3763 { }
3764 { 41 } % pdfnobuiltintounicode
3765 { \_unravel_scan_font_ident: }
3766 }
3767 { } % no other cases.
3768 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```
3769 \cs_new_protected:Npn \_unravel\_scan\_pdfcolorstack\_operands:
3770 {
3771   \_unravel\_scan\_int:
3772   \_unravel\_scan\_keyword:nF { sSeEtT }
3773   {
3774     \_unravel\_scan\_keyword:nF { pPuUsShH }
3775     {
3776       \_unravel\_scan\_keyword:nF { pPoOpP }
3777       {
3778         \_unravel\_scan\_keyword:nF { cCuUrRrReEnNtT }
3779         {
3780           \_unravel\_error:nnnnn { color-stack-action-missing }
3781           { } { } { } { } { }
3782         }
3783       }
3784     }
3785   }
3786 }
```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```
3787 \cs_new_protected:Npn \_unravel\_scan\_rule\_attr:
3788 {
3789   \_unravel\_scan\_alt\_rule:
3790   \_unravel\_scan\_keyword:nT { aAtTtTrR }
3791   { \_unravel\_scan\_pdf\_ext\_toks: }
3792 }
```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```
3793 \cs_new_protected:Npn \_unravel\_scan\_action:
3794 {
3795   \_unravel\_scan\_keyword:nTF { uUsSeErR }
3796   { \_unravel\_scan\_pdf\_ext\_toks: }
3797   {
3798     \_unravel\_scan\_keyword:nF { gGoOtToO }
3799     {
3800       \_unravel\_scan\_keyword:nF { tThHrReEaAdD }
3801       { \_unravel\_error:nnnnn { action-type-missing } { } { } { } { } { } }
3802     }
3803   }
3804   \_unravel\_scan\_keyword:nT { fFiIlLeE }
3805   { \_unravel\_scan\_pdf\_ext\_toks: }
3806   \_unravel\_scan\_keyword:nTF { pPaAgGeE }
3807   {
3808     \_unravel\_scan\_int:
3809     \_unravel\_scan\_pdf\_ext\_toks:
3810   }
3811   {
3812     \_unravel\_scan\_keyword:nTF { nNaAmMeE }
3813     { \_unravel\_scan\_pdf\_ext\_toks: }
```

```

3814         {
3815             \__unravel_scan_keyword:nTF { nNuUmM }
3816             { \__unravel_scan_int: }
3817             { \__unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
3818         }
3819     }
3820     \__unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3821     { \__unravel_skip_optional_space: }
3822     {
3823         \__unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3824         { \__unravel_skip_optional_space: }
3825     }
3826 }

```

(End definition for __unravel_scan_action:.)

__unravel_scan_image: Used by \pdfximage.

```

3827 \cs_new_protected:Npn \__unravel_scan_image:
3828 {
3829     \__unravel_scan_rule_attr:
3830     \__unravel_scan_keyword:nTF { nNaAmMeEdD }
3831     { \__unravel_scan_pdf_ext_toks: }
3832     {
3833         \__unravel_scan_keyword:nT { pPaAgGeE }
3834         { \__unravel_scan_int: }
3835     }
3836     \__unravel_scan_keyword:nT { cCoOlLoOrRsSpPaAcCeE }
3837     { \__unravel_scan_int: }
3838     \__unravel_scan_pdf_ext_toks:
3839 }

```

(End definition for __unravel_scan_image:.)

__unravel_scan_immediate_operands:

```

3840 \cs_new_protected:Npn \__unravel_scan_immediate_operands:
3841 {
3842     \__unravel_get_x_next:
3843     \__unravel_set_cmd:
3844     \int_compare:nNnTF
3845     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
3846     {
3847         \int_compare:nNnTF
3848         \l__unravel_head_char_int < { 3 } % openout, write, closeout
3849         { \__unravel_scan_immediate_operands_aux: }
3850         {
3851             \int_case:nnF \l__unravel_head_char_int
3852             {
3853                 { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
3854                 { 9 } { \__unravel_scan_extension_operands_aux: } % pdfxform
3855                 { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
3856             }
3857             { \__unravel_scan_immediate_operands_bad: }
3858         }
3859     }
3860     { \__unravel_scan_immediate_operands_bad: }

```

```

3861 }
3862 \cs_new_protected:Npn \__unravel_scan_immediate_operands_aux:
3863 {
3864   \__unravel_prev_input:V \l__unravel_head_tl
3865   \__unravel_scan_extension_operands:
3866 }
3867 \cs_new_protected:Npn \__unravel_scan_immediate_operands_bad:
3868 {
3869   \__unravel_back_input:
3870   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3871   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3872   \__unravel_prev_input_gpush:
3873 }
3874

```

(End definition for __unravel_scan_immediate_operands:.)

__unravel_scan_pdfdest_operands:

```

3875 \cs_new_protected:Npn \__unravel_scan_pdfdest_operands:
3876 {
3877   \__unravel_scan_keyword:nTF { nNuUmM }
3878   { \__unravel_scan_int: }
3879   {
3880     \__unravel_scan_keyword:nTF { nNaAmMeE }
3881     { \__unravel_scan_pdf_ext_toks: }
3882     { \__unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
3883   }
3884   \__unravel_scan_keyword:nTF { xXyYzZ }
3885   {
3886     \__unravel_scan_keyword:nT { zZoOoOmM }
3887     { \__unravel_scan_int: }
3888   }
3889   {
3890     \__unravel_scan_keyword:nF { fFiItTbBhH }
3891     {
3892       \__unravel_scan_keyword:nF { fFiItTbBvV }
3893       {
3894         \__unravel_scan_keyword:nF { fFiItTbB }
3895         {
3896           \__unravel_scan_keyword:nF { fFiItThHhH }
3897           {
3898             \__unravel_scan_keyword:nF { fFiItTvV }
3899             {
3900               \__unravel_scan_keyword:nTF
3901               { fFiItTrR }
3902               {
3903                 \__unravel_skip_optional_space:
3904                 \__unravel_scan_alt_rule:
3905                 \use_none:n
3906               }
3907             }
3908             \__unravel_scan_keyword:nF
3909             { fFiItT }
3910           }

```

```

3911                                     \_unravel_error:nnnn { destination-type-missing }
3912                                     { } { } { } { }
3913                                     }
3914                                 }
3915                            }
3916                       }
3917                   }
3918               }
3919           }
3920       }
3921   \_unravel_skip_optional_space:
3922 }

```

(End definition for _unravel_scan_pdfdest_operands:.)

2.13.7 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`.

```

3923 \cs_set_protected:Npn \_unravel_tmp:w
3924 {
3925     \_unravel_prev_input_gpush:
3926     \_unravel_prefixed_command:
3927 }
3928 \int_step_inline:nnnn
3929 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
3930 { 1 }
3931 { \_unravel_tex_use:n { max_command } }
3932 { \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.

```

3933 \cs_new_protected:Npn \_unravel_prefixed_command:
3934 {
3935     \int_while_do:nNnn
3936     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { prefix } }
3937     {
3938         \_unravel_prev_input:V \l__unravel_head_tl
3939         \_unravel_get_x_non_relax:
3940         \_unravel_set_cmd:
3941         \int_compare:nNnF \l__unravel_head_cmd_int
3942         > { \_unravel_tex_use:n { max_non_prefixed_command } }
3943         {
3944             \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3945             \_unravel_error:nxxxx { erroneous-prefixes }
3946             { \tl_to_str:N \l__unravel_tmpa_tl }
3947             { \tl_to_str:N \l__unravel_head_tl }
3948             { } { }
3949             \_unravel_back_input:
3950             \_unravel_omit_after_assignment:w
3951         }
3952     }

```

```

3953 % ^^A todo: Discard non-\global prefixes if they are irrelevant
3954 % ^^A todo: Adjust for the setting of \globaldefs
3955 \cs_if_exist_use:cF
3956 { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3957 {
3958   __unravel_error:nnnnn { internal } { prefixed } { } { } { }
3959   __unravel_omit_after_assignment:w
3960 }
3961 \__unravel_after_assignment:
3962 }

```

(End 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
3963 \cs_new_protected:Npn \__unravel_after_assignment:
3964 {
3965   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3966   \gtl_gclear:N \g__unravel_after_assignment_gtl
3967 }
3968 \cs_new_protected:Npn \__unravel_omit_after_assignment:w
3969 #1 \__unravel_after_assignment: { }

```

(End definition for __unravel_after_assignment: and __unravel_omit_after_assignment:w.)

```

\__unravel_prefixed_new:nn
3970 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3971 {
3972   \cs_new_protected:cpn
3973     { __unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3974 }

```

(End definition for __unravel_prefixed_new:nn.)

```

\__unravel_assign_token:n
3975 \cs_new_protected:Npn \__unravel_assign_token:n #1
3976 {
3977   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3978   #1
3979   \tl_use:N \l__unravel_head_tl \scan_stop:
3980   \__unravel_print_assigned_token:
3981 }

```

(End definition for __unravel_assign_token:n.)

```

\__unravel_assign_register:
3982 \cs_new_protected:Npn \__unravel_assign_register:
3983 {
3984   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3985   \tl_use:N \l__unravel_head_tl \scan_stop:
3986   \__unravel_print_assigned_register:
3987 }

```

(End definition for _unravel_assign_register:.)

_unravel_assign_value:nn

```

3988 \cs_new_protected:Npn \_unravel_assign_value:nn #1#2
3989 {
3990   \tl_if_empty:nF {#1}
3991   {
3992     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3993     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3994     #1
3995     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3996   }
3997   \__unravel_prev_input:V \l__unravel_head_tl
3998   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3999   \__unravel_scan_optional_equals:
4000   #2
4001   \_unravel_assign_register:
4002 }

```

(End definition for _unravel_assign_value:nn.)

_unravel_assign_toks:

```

4003 \_unravel_prefixed_new:nn { toks_register } % 71
4004 {
4005   \int_compare:nNnT \l__unravel_head_char_int = 0
4006   { % \toks
4007     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4008     \__unravel_print_action:
4009     \__unravel_scan_int:
4010     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4011   }
4012   \_unravel_assign_toks:
4013 }
4014 \_unravel_prefixed_new:nn { assign_toks } % 72
4015 { \_unravel_assign_toks: }
4016 \cs_new_protected:Npn \_unravel_assign_toks:
4017 {
4018   \__unravel_prev_input_silent:V \l__unravel_head_tl
4019   \__unravel_print_action:
4020   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4021   \__unravel_scan_optional_equals:
4022   \__unravel_get_x_non_relax:
4023   \__unravel_set_cmd:
4024   \int_compare:nNnTF
4025   \l__unravel_head_cmd_int = { \_unravel_tex_use:n { toks_register } }
4026   {
4027     \__unravel_prev_input:V \l__unravel_head_tl
4028     \int_compare:nNnT \l__unravel_head_char_int = 0
4029     { \_unravel_scan_int: }
4030   }
4031   {
4032     \int_compare:nNnTF
4033     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { assign_toks } }
4034     { \_unravel_prev_input:V \l__unravel_head_tl }

```

```

4035         {
4036             \_unravel_back_input:
4037             \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4038         }
4039     }
4040     \_unravel_assign_register:
4041 }

(End definition for \_unravel_assign_toks:.)

4042 \_unravel_prefixed_new:nn { assign_int } % 73
4043 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4044 \_unravel_prefixed_new:nn { assign_dimen } % 74
4045 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4046 \_unravel_prefixed_new:nn { assign_glue } % 75
4047 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_glue: } }
4048 \_unravel_prefixed_new:nn { assign_mu_glue } % 76
4049 { \_unravel_assign_value:nn { } { \_unravel_scan_mu_glue: } }
4050 \_unravel_prefixed_new:nn { assign_font_dimen } % 77
4051 {
4052     \_unravel_assign_value:nn
4053     { \_unravel_scan_int: \_unravel_scan_font_ident: }
4054     { \_unravel_scan_normal_dimen: }
4055 }
4056 \_unravel_prefixed_new:nn { assign_font_int } % 78
4057 {
4058     \_unravel_assign_value:nn
4059     { \_unravel_scan_font_int: } { \_unravel_scan_int: }
4060 }
4061 \_unravel_prefixed_new:nn { set_aux } % 79
4062 { % prevdepth = 1, spacefactor = 102
4063     \int_compare:nNnTF \l_unravel_head_char_int = 1
4064     { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4065     { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4066 }
4067 \_unravel_prefixed_new:nn { set_prev_graf } % 80
4068 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4069 \_unravel_prefixed_new:nn { set_page_dimen } % 81
4070 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4071 \_unravel_prefixed_new:nn { set_page_int } % 82
4072 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4073 \_unravel_prefixed_new:nn { set_box_dimen } % 83
4074 {
4075     \_unravel_assign_value:nn
4076     { \_unravel_scan_int: } { \_unravel_scan_normal_dimen: }
4077 }
4078 \_unravel_prefixed_new:nn { set_shape } % 84
4079 {
4080     \_unravel_assign_value:nn { \_unravel_scan_int: }
4081     {
4082         \prg_replicate:nn
4083         {
4084             \tl_if_head_eq_meaning:VNT
4085             \l_unravel_defined_tl \tex_parshape:D { 2 * }
4086             \tl_tail:N \l_unravel_defined_tl

```

```

4087     }
4088     { \__unravel_scan_int: }
4089   }
4090 }
4091 \__unravel_prefixed_new:nn { def_code } % 85
4092 {
4093   \__unravel_assign_value:nn
4094   { \__unravel_scan_int: } { \__unravel_scan_int: }
4095 }
4096 \__unravel_prefixed_new:nn { def_family } % 86
4097 {
4098   \__unravel_assign_value:nn
4099   { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
4100 }
4101 \__unravel_prefixed_new:nn { set_font } % 87
4102 {
4103   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4104   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4105   \tl_use:N \l__unravel_head_tl \scan_stop:
4106   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
4107   \__unravel_print_action:
4108 }
4109 \__unravel_prefixed_new:nn { def_font } % 88
4110 {
4111   \__unravel_prev_input_silent:V \l__unravel_head_tl
4112   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4113   \__unravel_scan_r_token:
4114   \__unravel_print_action:x
4115   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4116   \__unravel_scan_optional_equals:
4117   \__unravel_scan_file_name:
4118   \bool_gset_true:N \g__unravel_name_in_progress_bool
4119   \__unravel_scan_keyword:nTF { aAtT }
4120   { \__unravel_scan_normal_dimen: }
4121   {
4122     \__unravel_scan_keyword:nT { sScCaAlLeEdD }
4123     { \__unravel_scan_int: }
4124   }
4125   \bool_gset_false:N \g__unravel_name_in_progress_bool
4126   \__unravel_assign_token:n { }
4127 }

```

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

let, futurelet

```

4128 \__unravel_prefixed_new:nn { let } % 94
4129 {
4130   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4131   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4132   { % |let|
4133     \__unravel_scan_r_token:
4134     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4135     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4136     \__unravel_get_next:

```

```

4137 \bool_while_do:nn
4138 { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4139 { \__unravel_get_next: }
4140 \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4141 { \__unravel_get_next: }
4142 \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4143 { \__unravel_get_next: }
4144 }
4145 { % |futurelet|
4146 \__unravel_scan_r_token:
4147 \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4148 \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4149 \__unravel_get_next:
4150 \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4151 \__unravel_get_next:
4152 \__unravel_back_input:
4153 \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4154 \__unravel_back_input:
4155 }
4156 \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4157 \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4158 \__unravel_prev_input_gpop:N \l__unravel_head_tl
4159 \__unravel_exp_args:Nx \use:n
4160 {
4161 \exp_not:V \l__unravel_head_tl
4162 \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4163 }
4164 \__unravel_print_assigned_token:
4165 }
4166 \__unravel_prefixed_new:nn { shorthand_def } % 95
4167 {
4168 \__unravel_prev_input_silent:V \l__unravel_head_tl
4169 \tl_set:Nx \l__unravel_prev_action_tl
4170 { \tl_to_str:N \l__unravel_head_tl }
4171 \__unravel_scan_r_token:
4172 \__unravel_print_action:x
4173 { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4174 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4175 \__unravel_scan_optional_equals:
4176 \__unravel_scan_int:
4177 \__unravel_assign_token:n { }
4178 }

```

`__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.

```

4179 \__unravel_prefixed_new:nn { read_to_cs } % 96
4180 {
4181 \__unravel_prev_input_silent:V \l__unravel_head_tl
4182 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }

```

```

4183 \__unravel_scan_int:
4184 \__unravel_scan_to:
4185 \__unravel_scan_r_token:
4186 \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4187 \__unravel_read_to_cs_safe:fTF
4188 { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
4189 { \__unravel_assign_token:n { } }
4190 {
4191   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4192   \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4193 }
4194 }
4195 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
4196 {
4197   \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
4198   { \prg_return_true: }
4199   {
4200     \int_compare:nNnTF {#1} < { 0 }
4201     { \prg_return_false: }
4202     {
4203       \int_compare:nNnTF {#1} > { 15 }
4204       { \prg_return_false: }
4205       {
4206         \tex_ifeof:D #1 \exp_stop_f:
4207         \prg_return_false:
4208         \else:
4209         \prg_return_true:
4210         \fi:
4211       }
4212     }
4213   }
4214 }
4215 \cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }

(End definition for \__unravel_read_to_cs_safe:nTF.)

4216 \__unravel_prefixed_new:nn { def } % 97
4217 {
4218   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4219   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4220   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4221   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4222   \int_compare:nNnTF \l__unravel_head_char_int < 2
4223   { % def/gdef
4224     \__unravel_scan_r_token:
4225     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4226     \__unravel_scan_toks:NN \c_true_bool \c_false_bool
4227   }
4228   { % edef/xdef
4229     \__unravel_scan_r_token:
4230     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4231     \__unravel_scan_toks:NN \c_true_bool \c_true_bool
4232   }
4233   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4234   \__unravel_prev_input:V \l__unravel_head_tl

```

```

4235     \__unravel_assign_token:n
4236     { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4237 }

```

\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.

```

4238 \__unravel_prefixed_new:nn { set_box } % 98
4239 {
4240     \__unravel_prev_input:V \l__unravel_head_tl
4241     \__unravel_scan_int:
4242     \__unravel_scan_optional_equals:
4243     \bool_if:NTF \g__unravel_set_box_allowed_bool
4244     { \__unravel_do_box:N \c_false_bool }
4245     {
4246         \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
4247         \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4248         \__unravel_omit_after_assignment:w
4249     }
4250 }

```

\hyphenation and \patterns

```

4251 \__unravel_prefixed_new:nn { hyph_data } % 99
4252 {
4253     \__unravel_prev_input:V \l__unravel_head_tl
4254     \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4255     \__unravel_assign_token:n { }
4256 }

```

```

4257 \__unravel_prefixed_new:nn { set_interaction } % 100
4258 {
4259     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4260     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4261     \tl_use:N \l__unravel_head_tl \scan_stop:
4262     \__unravel_print_assignment:x { \tl_to_str:N \l__unravel_head_tl }
4263 }

```

```

4264 \__unravel_prefixed_new:nn { letterspace_font } % 101
4265 {
4266     \__unravel_prev_input_silent:V \l__unravel_head_tl
4267     \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4268     \__unravel_scan_r_token:
4269     \__unravel_print_action:x
4270     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4271     \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4272     \__unravel_scan_optional_equals:
4273     \__unravel_scan_font_ident:
4274     \__unravel_scan_int:
4275     \__unravel_assign_token:n { }
4276 }

```

```

4277 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4278 {
4279     \__unravel_prev_input_silent:V \l__unravel_head_tl
4280     \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4281     \__unravel_scan_r_token:

```

```

4282 \__unravel_print_action:x
4283 { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4284 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4285 \__unravel_scan_optional_equals:
4286 \__unravel_scan_font_ident:
4287 \__unravel_assign_token:n { }
4288 }

```

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

```

4289 \__unravel_prefixed_new:nn { register } % 89
4290 { \__unravel_do_register:N 0 }
4291 \__unravel_prefixed_new:nn { advance } % 90
4292 { \__unravel_do_operation:N 1 }
4293 \__unravel_prefixed_new:nn { multiply } % 91
4294 { \__unravel_do_operation:N 2 }
4295 \__unravel_prefixed_new:nn { divide } % 92
4296 { \__unravel_do_operation:N 3 }

```

__unravel_do_operation:N

__unravel_do_operation_fail:w

```

4297 \cs_new_protected:Npn \__unravel_do_operation:N #1
4298 {
4299 \__unravel_prev_input_silent:V \l__unravel_head_tl
4300 \__unravel_print_action:
4301 \__unravel_get_x_next:
4302 \__unravel_set_cmd:
4303 \int_compare:nNnTF
4304 \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4305 {
4306 \int_compare:nNnTF
4307 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4308 { \__unravel_do_register:N #1 }
4309 { \__unravel_do_operation_fail:w }
4310 }
4311 {
4312 \int_compare:nNnTF
4313 \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
4314 { \__unravel_do_operation_fail:w }
4315 {
4316 \__unravel_prev_input:V \l__unravel_head_tl
4317 \exp_args:NNf \__unravel_do_register_set:Nn #1
4318 {
4319 \int_eval:n
4320 {
4321 \l__unravel_head_cmd_int
4322 - \__unravel_tex_use:n { assign_toks }
4323 }
4324 }
4325 }
4326 }
4327 }
4328 \cs_new_protected:Npn \__unravel_do_operation_fail:w
4329 {
4330 \__unravel_error:nnnnn { after-advance } { } { } { } { } { }

```

```

4331     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4332     \__unravel_omit_after_assignment:w
4333 }

```

(End definition for __unravel_do_operation:N and __unravel_do_operation_fail:w.)

__unravel_do_register:N

```

\__unravel_do_register_aux:Nn
4334 \cs_new_protected:Npn \__unravel_do_register:N #1
4335 {
4336     \exp_args:NNV \__unravel_do_register_aux:Nn #1
4337     \l__unravel_head_char_int
4338 }
4339 \cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
4340 {
4341     \int_compare:nNnTF { \tl_tail:n {#2} } = 0
4342     {
4343         \__unravel_prev_input_gpush:N \l__unravel_head_tl
4344         \__unravel_print_assignment:
4345         \__unravel_scan_int:
4346         \__unravel_prev_input_gpop:N \l__unravel_head_tl
4347         \__unravel_prev_input_silent:V \l__unravel_head_tl
4348     }
4349     {
4350         \__unravel_prev_input_silent:V \l__unravel_head_tl
4351         \__unravel_print_assignment:
4352     }
4353     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4354     \exp_args:NNf \__unravel_do_register_set:Nn #1
4355     { \int_eval:n { #2 / 1 000 000 } }
4356 }

```

(End definition for __unravel_do_register:N and __unravel_do_register_aux:Nn.)

__unravel_do_register_set:Nn

```

4357 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
4358 {
4359     \int_compare:nNnTF {#1} = 0
4360     { % truly register command
4361         \__unravel_scan_optional_equals:
4362     }
4363     { % \advance, \multiply, \divide
4364         \__unravel_scan_keyword:nF { bByY }
4365         { \__unravel_prev_input_silent:n { by } }
4366     }
4367     \int_compare:nNnTF {#1} < 2
4368     {
4369         \int_case:nnF {#2}
4370         {
4371             { 1 } { \__unravel_scan_int: } % count
4372             { 2 } { \__unravel_scan_normal_dimen: } % dim
4373             { 3 } { \__unravel_scan_normal_glue: } % glue
4374             { 4 } { \__unravel_scan_mu_glue: } % muglue
4375         }
4376         { \__unravel_error:nxxxx { internal } { do-reg=#2 } { } { } { } }
4377     }

```

```

4378     { \_unravel\_scan\_int: }
4379     \_unravel\_assign\_register:
4380 }

(End definition for \_unravel\_do\_register\_set:Nn.)
The following is used for instance when making accents.
4381 \cs\_new\_protected:Npn \_unravel\_do\_assignments:
4382 {
4383   \_unravel\_get\_x\_non\_relax:
4384   \_unravel\_set\_cmd:
4385   \int\_compare:nNnT
4386   \l\_unravel\_head\_cmd\_int
4387   > { \_unravel\_tex\_use:n { max\_non\_prefixed\_command } }
4388   {
4389     \bool\_gset\_false:N \g\_unravel\_set\_box\_allowed\_bool
4390     \_unravel\_prev\_input\_gpush:
4391     \_unravel\_prefixed\_command:
4392     \bool\_gset\_true:N \g\_unravel\_set\_box\_allowed\_bool
4393     \_unravel\_do\_assignments:
4394   }
4395 }

```

2.14 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`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifpdfprimitive`, `\ifpdfabsnum`, and `\ifpdfabsdim`.
- `fi_or_else=108` for `\fi`, `\else` and `\or`.
- `cs_name=109` for `\csname`.
- `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`, `\pdfinserttht`, `\pdfximagebbox`, `\jobname`, and in LuaTeX `\directlua`, `\expanded`, `\luaescapestring`.
- `the=111` for `\the`, `\unexpanded`, and `\detokenize`.
- `top_bot_mark=112` for `\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.

```

4396 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
4397 { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }

\__unravel_expandafter:
  \__unravel_unless:
\__unravel_unless_bad:
4398 \__unravel_new_tex_expandable:nn { expand_after } % 104
4399 {
4400   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4401   { \__unravel_expandafter: } { \__unravel_unless: }
4402 }
4403 \cs_new_protected:Npn \__unravel_expandafter:
4404 {
4405   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4406   \__unravel_get_next:
4407   \gtl_concat:NNN \l__unravel_head_gtl
4408   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4409   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4410   \__unravel_print_expansion:x { \gtl_to_str:N \l__unravel_head_gtl }
4411   \__unravel_get_next:
4412   \__unravel_token_if_expandable:NNTF \l__unravel_head_token
4413   { \__unravel_expand_do:N \prg_do_nothing: }
4414   { \__unravel_back_input: }
4415   \__unravel_prev_input_gpop:N \l__unravel_head_gtl
4416   \__unravel_set_action_text:x
4417   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4418   \gtl_pop_left:N \l__unravel_head_gtl
4419   \__unravel_back_input:
4420   \__unravel_print_expansion:
4421 }
4422 \cs_new_protected:Npn \__unravel_unless:
4423 {
4424   \__unravel_get_token:
4425   \int_compare:nNnTF
4426   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4427   {
4428     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4429     { \__unravel_unless_bad: }
4430     {
4431       \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4432       % \int_add:Nn \l__unravel_head_char_int { 32 }
4433       \__unravel_expand_nonmacro:
4434     }
4435   }
4436   { \__unravel_unless_bad: }
4437 }
4438 \cs_new_protected:Npn \__unravel_unless_bad:
4439 {
4440   \__unravel_error:nnnnn { bad-unless } { } { } { } { }
4441   \__unravel_back_input:
4442 }

```

(End 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` 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`.
`_unravel_pdfprimitive:`

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_EX for expandable commands.

```

4443 \_unravel\_new\_tex\_expandable:nn { no\_expand } % 105
4444 {
4445   \token\_if\_eq\_meaning:NNTF \l\_unravel\_head\_token \tex\_noexpand:D
4446   { \_unravel\_noexpand:N }
4447   { \_unravel\_pdfprimitive: }
4448 }
4449 \cs\_new\_protected:Npn \_unravel\_noexpand:N #1
4450 {
4451   \_unravel\_get\_token:
4452   \cs\_if\_eq:NNTF #1 \prg\_do\_nothing:
4453   {
4454     \tl\_if\_empty:NNTF \l\_unravel\_head\_tl
4455     { \_unravel\_back\_input: }
4456     {
4457       \exp\_after:wN \_unravel\_token\_if\_definable:NNTF \l\_unravel\_head\_tl
4458       { \_unravel\_noexpand\_after: }
4459       { \_unravel\_back\_input: }
4460     }
4461   }
4462   {
4463     \_unravel\_back\_input:
4464     \_unravel\_get\_next:
4465     \_unravel\_token\_if\_expandable:NT \l\_unravel\_head\_token
4466     { \cs\_set\_eq:NN \l\_unravel\_head\_token \_unravel\_special\_relax: }
4467   }
4468 }
4469 \cs\_new\_protected:Npn \_unravel\_noexpand\_after:
4470 {
4471   \group\_begin:
4472   \_unravel\_set\_escapechar:n { 92 }
4473   \exp\_args:NNc
4474   \group\_end:
4475   \_unravel\_noexpand\_after:N
4476   { notexpanded: \exp\_after:wN \token\_to\_str:N \l\_unravel\_head\_tl }
4477 }

```

```

4478 \cs_new_protected:Npn \__unravel_noexpand_after:N #1
4479 {
4480   \cs_gset_eq:NN #1 \__unravel_special_relax:
4481   \__unravel_back_input:n {#1}
4482 }
4483 \cs_new_protected:Npn \__unravel_pdfprimitive:
4484 { \__unravel_not_implemented:n { pdfprimitive } }

(End definition for \__unravel_noexpand:N, \__unravel_noexpand_after:, and \__unravel_pdfprimitive:.)

```

```

\__unravel_endinput:
\__unravel_scantokens: 4485 \__unravel_new_tex_expandable:nn { input } % 106
\__unravel_input:      4486 {
4487   \int_case:nnF \l__unravel_head_char_int
4488   {
4489     { 1 } { \__unravel_endinput: } % \endinput
4490     { 2 } { \__unravel_scantokens: } % \scantokens
4491   }
4492   { % 0=\input
4493     \bool_if:NTF \g__unravel_name_in_progress_bool
4494     { \__unravel_insert_relax: } { \__unravel_input: }
4495   }
4496 }
4497 \cs_new_protected:Npn \__unravel_endinput:
4498 {
4499   \group_begin:
4500   \msg_warning:nn { unravel } { endinput-ignored }
4501   \group_end:
4502   \__unravel_print_expansion:
4503 }
4504 \cs_new_protected:Npn \__unravel_scantokens:
4505 {
4506   \__unravel_prev_input_gpush:
4507   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4508   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4509   \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4510   \__unravel_back_input:V \l__unravel_head_tl
4511   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_tmpa_tl }
4512 }
4513 \cs_new_protected:Npn \__unravel_input:
4514 {
4515   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4516   \__unravel_scan_file_name:
4517   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4518   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4519   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4520   \__unravel_back_input:V \l__unravel_tmpa_tl
4521   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
4522 }

(End definition for \__unravel_endinput:, \__unravel_scantokens:, and \__unravel_input:.)

```

```

\__unravel_csname_loop:
4523 \__unravel_new_tex_expandable:nn { cs_name } % 109
4524 {

```

```

4525 \__unravel_prev_input_gpush:N \l__unravel_head_tl
4526 \__unravel_print_expansion:
4527 \__unravel_csname_loop:
4528 \__unravel_prev_input_silent:V \l__unravel_head_tl
4529 \__unravel_prev_input_gpop:N \l__unravel_head_tl
4530 \__unravel_back_input_tl_o:
4531 }
4532 \cs_new_protected:Npn \__unravel_csname_loop:
4533 {
4534 \__unravel_get_x_next:
4535 \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4536 {
4537 \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4538 {
4539 \__unravel_back_input:
4540 \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4541 \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4542 }
4543 }
4544 {
4545 \__unravel_prev_input_silent:x
4546 { \__unravel_token_to_char:N \l__unravel_head_token }
4547 \__unravel_csname_loop:
4548 }
4549 }

```

(End definition for __unravel_csname_loop:.)

```

4550 \__unravel_new_tex_expandable:nn { convert } % 110
4551 {
4552 \__unravel_prev_input_gpush:N \l__unravel_head_tl
4553 \__unravel_print_expansion:
4554 \int_case:nn \l__unravel_head_char_int
4555 {
4556 0 \__unravel_scan_int:
4557 1 \__unravel_scan_int:
4558 2 \__unravel_convert_string:
4559 3 \__unravel_convert_meaning:
4560 4 \__unravel_scan_font_ident:
4561 8 \__unravel_scan_font_ident:
4562 9 \__unravel_scan_font_ident:
4563 { 10 } \__unravel_scan_font_ident:
4564 { 11 } \__unravel_scan_int:
4565 { 12 } \__unravel_scan_int:
4566 { 13 } \__unravel_scan_pdf_ext_toks:
4567 { 14 } \__unravel_scan_pdf_ext_toks:
4568 { 15 } \__unravel_scan_int:
4569 { 16 } \__unravel_scan_int:
4570 { 17 } \__unravel_scan_pdfstrcmp:
4571 { 18 } \__unravel_scan_pdfcolorstackinit:
4572 { 19 } \__unravel_scan_pdf_ext_toks:
4573 { 20 } \__unravel_scan_pdf_ext_toks:
4574 { 22 } \__unravel_scan_pdf_ext_toks:
4575 { 23 } \__unravel_scan_pdf_ext_toks:
4576 { 24 }

```

```

4577         {
4578             \__unravel_scan_keyword:n { fFillLeE }
4579             \__unravel_scan_pdf_ext_toks:
4580         }
4581         { 25 } \__unravel_scan_pdffileddump:
4582         { 26 } \__unravel_scan_pdfmatch:
4583         { 27 } \__unravel_scan_int:
4584         { 28 } \__unravel_scan_int:
4585         { 30 } \__unravel_scan_int:
4586         { 31 } \__unravel_scan_pdfximagebbox:
4587         { 33 } \__unravel_scan_directlua:
4588         { 34 } \__unravel_scan_pdf_ext_toks:
4589         { 35 } \__unravel_scan_pdf_ext_toks:
4590         { 40 }
4591         {
4592             \__unravel_scan_int:
4593             \__unravel_prev_input_silent:n { ~ }
4594             \__unravel_scan_int:
4595         }
4596     }
4597     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4598     \__unravel_back_input_tl_o:
4599 }
4600 \cs_new_protected:Npn \__unravel_convert_string:
4601 {
4602     \__unravel_get_next:
4603     \tl_if_empty:NTF \l__unravel_head_tl
4604     { \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_head_gtl } }
4605     { \__unravel_prev_input:V \l__unravel_head_tl }
4606 }
4607 \cs_new_protected:Npn \__unravel_convert_meaning:
4608 {
4609     \__unravel_get_next:
4610     \tl_if_empty:NTF \l__unravel_head_tl
4611     { \__unravel_prev_input:n { \l__unravel_head_token } }
4612     { \__unravel_prev_input:V \l__unravel_head_tl }
4613 }
4614 \cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
4615 {
4616     \__unravel_scan_toks_to_str:
4617     \__unravel_scan_toks_to_str:
4618 }
4619 \cs_new_protected:Npn \__unravel_scan_pdfximagebbox:
4620 { \__unravel_scan_int: \__unravel_scan_int: }
4621 \cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:
4622 {
4623     \__unravel_scan_keyword:nTF { pPaAgGeE }
4624     { \bool_set_true:N \l__unravel_tmpa_bool }
4625     { \bool_set_false:N \l__unravel_tmpb_bool }
4626     \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4627     { \__unravel_scan_keyword:n { pPaAgGeE } }
4628     \__unravel_scan_toks_to_str:
4629 }
4630 \cs_new_protected:Npn \__unravel_scan_pdffileddump:

```

```

4631 {
4632   \_unravel\_scan\_keyword:nT { oOfFfFsSeEtT } \_unravel\_scan\_int:
4633   \_unravel\_scan\_keyword:nT { lLeEnNgGtThH } \_unravel\_scan\_int:
4634   \_unravel\_scan\_pdf\_ext\_toks:
4635 }
4636 \cs\_new\_protected:Npn \_unravel\_scan\_pdfmatch:
4637 {
4638   \_unravel\_scan\_keyword:n { iIcCaAsSeE }
4639   \_unravel\_scan\_keyword:nT { sSuUbBcCoOuUnNtT }
4640   { \_unravel\_scan\_int: }
4641   \_unravel\_scan\_pdf\_ext\_toks:
4642   \_unravel\_scan\_pdf\_ext\_toks:
4643 }
4644 \sys\_if\_engine\_luatex:T
4645 {
4646   \cs\_new\_protected:Npn \_unravel\_scan\_directlua:
4647   {
4648     \_unravel\_get\_x\_non\_relax:
4649     \token\_if\_eq\_catcode:NNTF \l\_unravel\_head\_token \c\_group\_begin\_token
4650     { \_unravel\_back\_input: }
4651     {
4652       \_unravel\_scan\_int:
4653       \_unravel\_get\_x\_non\_relax:
4654     }
4655     \_unravel\_scan\_pdf\_ext\_toks:
4656   }
4657 }

```

_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.

```

4658 \_unravel\_new\_tex\_expandable:nn { the } % 111
4659 { \_unravel\_get\_the:N }
4660 \cs\_new\_protected:Npn \_unravel\_get\_the:N #1
4661 {
4662   \_unravel\_prev\_input\_gpush:N \l\_unravel\_head\_tl
4663   \_unravel\_print\_expansion:
4664   \int\_if\_odd:nTF \l\_unravel\_head\_char\_int
4665   { % \unexpanded, \detokenize
4666     \_unravel\_scan\_toks:NN \c\_false\_bool \c\_false\_bool
4667     \_unravel\_prev\_input\_gpop:N \l\_unravel\_head\_tl
4668     \_unravel\_set\_action\_text:x { \tl\_to\_str:N \l\_unravel\_head\_tl }
4669   }
4670   { % \the
4671     \_unravel\_get\_x\_next:
4672     \_unravel\_scan\_something\_internal:n { 5 }
4673     \_unravel\_prev\_input\_gpop:N \l\_unravel\_head\_tl
4674     \_unravel\_set\_action\_text:x
4675     {
4676       \tl\_head:N \l\_unravel\_head\_tl
4677       => \tl\_tail:N \l\_unravel\_head\_tl
4678     }
4679     \tl\_set:Nx \l\_unravel\_head\_tl
4680     { \exp\_not:N \exp\_not:n { \tl\_tail:N \l\_unravel\_head\_tl } }
4681   }

```

```

4682 \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
4683 {
4684   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4685   \__unravel_prev_input:V \l__unravel_head_tl
4686 }
4687 {
4688   \cs_if_eq:NNTF #1 \__unravel_get_token_x:
4689   {
4690     \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
4691     \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
4692   }
4693   {
4694     \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4695     \__unravel_back_input:V \l__unravel_tmpa_tl
4696   }
4697   \__unravel_print_expansion:
4698 }
4699 #1
4700 }

```

(End definition for __unravel_get_the:N.)

```

4701 \__unravel_new_tex_expandable:nn { top_bot_mark } % 112
4702 { \__unravel_back_input_tl_o: }
4703 \__unravel_new_tex_expandable:nn { end_template } % 117
4704 {
4705   \__unravel_not_implemented:n { end-template } { } { } { }
4706   \__unravel_back_input_tl_o:
4707 }

```

2.14.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w
4708 \cs_new_protected:Npn \__unravel_pass_text:
4709 {
4710   \__unravel_input_if_empty:TF
4711   { \__unravel_pass_text_empty: }
4712   {
4713     \__unravel_input_get:N \l__unravel_tmpb_gtl
4714     \if_true:
4715       \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
4716       \exp_after:wN \__unravel_pass_text_done:w
4717       \fi:
4718       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4719       \exp_after:wN \__unravel_pass_text:
4720     \else:
4721       \use:c { fi: }
4722       \int_set:Nn \l__unravel_if_nesting_int { 1 }
4723       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4724       \exp_after:wN \__unravel_pass_text_nested:
4725       \fi:
4726   }
4727 }
4728 \cs_new_protected:Npn \__unravel_pass_text_done:w

```

```

4729 {
4730   \__unravel_get_next:
4731   \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4732   \else:
4733 }

```

(End 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:.

```

4734 \cs_new_protected:Npn \__unravel_pass_text_nested:
4735 {
4736   \__unravel_input_if_empty:TF
4737   { \__unravel_pass_text_empty: }
4738   {
4739     \__unravel_input_get:N \l__unravel_tmpb_gtl
4740     \if_true:
4741     \if_true:
4742     \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4743     \int_decr:N \l__unravel_if_nesting_int
4744     \use_none:nnnnn
4745     \fi:
4746     \use_none:nnn
4747     \fi:
4748     \int_incr:N \l__unravel_if_nesting_int
4749     \fi:
4750     \__unravel_input_gpop:N \l__unravel_unused_gtl
4751     \int_compare:nNnTF \l__unravel_if_nesting_int = 0
4752     { \__unravel_pass_text: }
4753     { \__unravel_pass_text_nested: }
4754   }
4755 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

4756 \cs_new_protected:Npn \__unravel_pass_text_empty:
4757 {
4758   \__unravel_error:nnnnn { runaway-if } { } { } { } { } { }
4759   \__unravel_exit:w
4760 }

```

(End definition for __unravel_pass_text_empty:.)

```

\__unravel_cond_push:
\__unravel_cond_pop:
4761 \cs_new_protected:Npn \__unravel_cond_push:
4762 {
4763   \tl_gput_left:Nx \g__unravel_if_limit_tl
4764   { { \int_use:N \g__unravel_if_limit_int } }
4765   \int_gincr:N \g__unravel_if_depth_int
4766   \int_gzero:N \g__unravel_if_limit_int
4767 }
4768 \cs_new_protected:Npn \__unravel_cond_pop:
4769 {
4770   \int_gset:Nn \g__unravel_if_limit_int
4771   { \tl_head:N \g__unravel_if_limit_tl }
4772   \tl_gset:Nx \g__unravel_if_limit_tl
4773   { \tl_tail:N \g__unravel_if_limit_tl }
4774   \int_gdecr:N \g__unravel_if_depth_int
4775 }

(End definition for \__unravel_cond_push: and \__unravel_cond_pop:.)

\__unravel_change_if_limit:nn
4776 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
4777 {
4778   \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4779   { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4780   {
4781     \tl_clear:N \l__unravel_tmpa_tl
4782     \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
4783     {
4784       \tl_put_right:Nx \l__unravel_tmpa_tl
4785       { { \tl_head:N \g__unravel_if_limit_tl } }
4786       \tl_gset:Nx \g__unravel_if_limit_tl
4787       { \tl_tail:N \g__unravel_if_limit_tl }
4788     }
4789     \tl_gset:Nx \g__unravel_if_limit_tl
4790     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4791   }
4792 }

(End definition for \__unravel_change_if_limit:nn.)

4793 \__unravel_new_tex_expandable:nn { if_test } % 107
4794 {
4795   \__unravel_cond_push:
4796   \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
4797 }

\__unravel_cond_aux:nn
4798 \cs_new_protected:Npn \__unravel_cond_aux:n #1
4799 {
4800   \int_case:nnF \l__unravel_head_char_int
4801   {
4802     { 12 } { \__unravel_test_ifx:n {#1} }
4803     { 16 } { \__unravel_test_case:n {#1} }
4804     { 21 } { \__unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4805   }

```

```

4806 {
4807   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4808   \__unravel_print_expansion:
4809   \int_case:nn \l__unravel_head_char_int
4810   {
4811     { 0 } { \__unravel_test_two_chars: } % if
4812     { 1 } { \__unravel_test_two_chars: } % ifcat
4813     { 2 } % ifnum
4814     { \__unravel_test_two_vals:N \__unravel_scan_int: }
4815     { 3 } % ifdim
4816     { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4817     { 4 } { \__unravel_scan_int: } % ifodd
4818     % { 5 } { } % ifvmode
4819     % { 6 } { } % ifhmode
4820     % { 7 } { } % ifmmode
4821     % { 8 } { } % ifinner
4822     { 9 } { \__unravel_scan_int: } % ifvoid
4823     { 10 } { \__unravel_scan_int: } % ifhbox
4824     { 11 } { \__unravel_scan_int: } % ifvbox
4825     { 13 } { \__unravel_scan_int: } % ifeof
4826     % { 14 } { } % iftrue
4827     % { 15 } { } % iffalse
4828     { 17 } { \__unravel_test_ifdefined: } % ifdefined
4829     { 18 } { \__unravel_test_ifcsname: } % ifcsname
4830     { 19 } % iffontchar
4831     { \__unravel_scan_font_ident: \__unravel_scan_int: }
4832     % { 20 } { } % ifincsname % ^^A todo: something?
4833     { 22 } % ifpdfabsnum
4834     { \__unravel_test_two_vals:N \__unravel_scan_int: }
4835     { 23 } % ifpdfabsdim
4836     { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4837   }
4838   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4839   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4840   \l__unravel_head_tl \scan_stop:
4841   \exp_after:wN \__unravel_cond_true:n
4842   \else:
4843     \exp_after:wN \__unravel_cond_false:n
4844   \fi:
4845   {#1}
4846 }
4847 }

```

(End definition for __unravel_cond_aux:nn.)

__unravel_cond_true:n

```

4848 \cs_new_protected:Npn \__unravel_cond_true:n #1
4849 {
4850   \__unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4851   \__unravel_print_expansion:x { \g__unravel_action_text_str = true }
4852 }

```

(End definition for __unravel_cond_true:n.)

```

    \_unravel_cond_false:n
\_unravel_cond_false_loop:n
    \_unravel_cond_false_common:
4853 \cs_new_protected:Npn \_unravel_cond_false:n #1
4854 {
4855     \_unravel_cond_false_loop:n {#1}
4856     \_unravel_cond_false_common:
4857     \_unravel_print_expansion:x
4858     {
4859         \g_unravel_action_text_str = false ~
4860         => ~ skipped ~ to ~ \iow_char:N\fi
4861     }
4862 }
4863 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4864 {
4865     \_unravel_pass_text:
4866     \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
4867     {
4868         \token_if_eq_meaning:NNT \l_unravel_head_token \or:
4869         {
4870             \_unravel_error:nnnnn { extra-or } { } { } { } { }
4871             \_unravel_cond_false_loop:n {#1}
4872         }
4873     }
4874     {
4875         \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
4876         { \_unravel_cond_pop: }
4877         \_unravel_cond_false_loop:n {#1}
4878     }
4879 }
4880 \cs_new_protected:Npn \_unravel_cond_false_common:
4881 {
4882     \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
4883     { \_unravel_cond_pop: }
4884     { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
4885 }

(End definition for \_unravel_cond_false:n, \_unravel_cond_false_loop:n, and \_unravel_
cond_false_common:.)

\_unravel_test_two_vals:N
4886 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
4887 {
4888     #1
4889     \_unravel_get_x_non_blank:
4890     \_unravel_tl_if_in:ooTF { < = > } \l_unravel_head_tl { }
4891     {
4892         \_unravel_error:nnnnn { missing-equals } { } { } { } { }
4893         \_unravel_back_input:
4894         \tl_set:Nn \l_unravel_head_tl { = }
4895     }
4896     \_unravel_prev_input:V \l_unravel_head_tl
4897     #1
4898 }

(End definition for \_unravel_test_two_vals:N.)

```

```

\__unravel_test_two_chars:
  \__unravel_test_two_chars_aux:
4899   \cs_new_protected:Npn \__unravel_test_two_chars:
4900   {
4901     \__unravel_test_two_chars_aux:
4902     \__unravel_prev_input:V \l__unravel_head_tl
4903     \__unravel_test_two_chars_aux:
4904     \__unravel_prev_input:V \l__unravel_head_tl
4905   }
4906   \cs_new_protected:Npn \__unravel_test_two_chars_aux:
4907   {
4908     \__unravel_get_x_next:
4909     \gtl_if_tl:NF \l__unravel_head_gtl
4910     {
4911       \tl_set:Nx \l__unravel_head_tl
4912       {
4913         \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
4914         { \c_group_begin_token } { \c_group_end_token }
4915       }
4916     }
4917     \tl_put_left:Nn \l__unravel_head_tl { \exp_not:N } % ^^A todo: prettify.
4918   }

```

(End definition for __unravel_test_two_chars: and __unravel_test_two_chars_aux:.)

```

\__unravel_test_ifx:n
\__unravel_test_ifx_str:NN
\__unravel_test_ifx_aux:NNN
\__unravel_test_ifx_aux:w

```

The token equal to \ifx is pushed as a previous input to show an action nicely, then retrieved as \l__unravel_tmpa_tl after getting the next two tokens as tmpb and head. Then we call \l__unravel_tmpa_tl followed by these two tokens. A previous implementation 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.

```

4919   \cs_new_protected:Npn \__unravel_test_ifx:n #1
4920   {
4921     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4922     \__unravel_print_expansion:
4923     \__unravel_get_next:
4924     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4925     \cs_set_eq:NN \l__unravel_tmpb_token \l__unravel_head_token
4926     \__unravel_get_next:
4927     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4928     \__unravel_set_action_text:x
4929     {
4930       Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4931       \__unravel_test_ifx_str:NN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
4932       \__unravel_test_ifx_str:NN \l__unravel_head_token \l__unravel_head_gtl
4933     }
4934     \__unravel_test_ifx_aux:NNN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
4935     \__unravel_test_ifx_aux:w
4936     \exp_after:wN \__unravel_cond_true:n
4937   \else:
4938     \exp_after:wN \__unravel_cond_false:n

```

```

4939     \fi:
4940     {#1}
4941   }
4942   \cs_new:Npn \__unravel_test_ifx_str:NN #1#2
4943   {
4944     \token_if_eq_meaning:NNT #1 \__unravel_special_relax:
4945     { \iow_char:N \notexpanded: }
4946     \gtl_to_str:N #2
4947   }
4948   \cs_new_protected:Npn \__unravel_test_ifx_aux:NNN #1#2#3
4949   {
4950     \token_if_eq_meaning:NNTF #1 \__unravel_special_relax:
4951     {
4952       \gtl_head_do:NN #2 \__unravel_token_if_expandable:NNTF
4953       { #3 #1 } { \gtl_head_do:NN #2 #3 }
4954     }
4955     { \gtl_head_do:NN #2 #3 }
4956   }
4957   \cs_new:Npn \__unravel_test_ifx_aux:w
4958   {
4959     \__unravel_test_ifx_aux:NNN \l__unravel_head_token \l__unravel_head_gtl
4960     \l__unravel_tmpa_tl
4961   }

```

(End definition for __unravel_test_ifx:n and others.)

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn
4962 \cs_new_protected:Npn \__unravel_test_case:n #1
4963 {
4964   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4965   \__unravel_print_expansion:
4966   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
4967   \__unravel_scan_int:
4968   \__unravel_prev_input_get:N \l__unravel_head_tl
4969   \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
4970   % ^^A does text_case_aux use prev_input_seq?
4971   \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
4972   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4973   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
4974 }
4975 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
4976 {
4977   \int_compare:nNnTF {#1} = 0
4978   { \__unravel_change_if_limit:nn { 4 } {#2} }
4979   {
4980     \__unravel_pass_text:
4981     \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
4982     {
4983       \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
4984       {
4985         \exp_args:Nf \__unravel_test_case_aux:nn
4986         { \int_eval:n { #1 - 1 } } {#2}
4987       }
4988       { \__unravel_cond_false_common: }

```

```

4989     }
4990     {
4991         \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4992         { \__unravel_cond_pop: }
4993         \__unravel_test_case_aux:nn {#1} {#2}
4994     }
4995 }
4996 }

```

(End definition for __unravel_test_case:n and __unravel_test_case_aux:nn.)

__unravel_test_ifdefined:

```

4997 \cs_new_protected:Npn \__unravel_test_ifdefined:
4998 {
4999     \__unravel_input_if_empty:TF
5000     { \__unravel_pass_text_empty: }
5001     {
5002         \__unravel_input_gpop:N \l__unravel_tmpb_gtl
5003         \__unravel_set_action_text:x
5004         {
5005             Conditional:~ \tl_to_str:N \l__unravel_head_tl
5006             \gtl_to_str:N \l__unravel_tmpb_gtl
5007         }
5008         \__unravel_prev_input:x
5009         {
5010             \gtl_if_tl:NNTF \l__unravel_tmpb_gtl
5011             { \gtl_head:N \l__unravel_tmpb_gtl }
5012             { \gtl_to_str:N \l__unravel_tmpb_gtl }
5013         }
5014     }
5015 }

```

(End definition for __unravel_test_ifdefined:.)

__unravel_test_ifcsname:

```

5016 \cs_new_protected:Npn \__unravel_test_ifcsname:
5017 {
5018     \__unravel_csname_loop:
5019     \__unravel_prev_input:V \l__unravel_head_tl
5020 }

```

(End definition for __unravel_test_ifcsname:.)

```

5021 \__unravel_new_tex_expandable:nn { fi_or_else } % 108
5022 {
5023     \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
5024     {
5025         \int_compare:nNnTF \g__unravel_if_limit_int = 0
5026         {
5027             \int_compare:nNnTF \g__unravel_if_depth_int = 0
5028             { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
5029             { \__unravel_insert_relax: }
5030         }
5031         { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
5032     }

```

```

5033     {
5034         \__unravel_set_action_text:
5035         \int_compare:nNnF \l__unravel_head_char_int = 2
5036         {
5037             \__unravel_fi_or_else_loop:
5038             \__unravel_set_action_text:x
5039             {
5040                 \g__unravel_action_text_str \c_space_tl
5041                 => ~ skipped ~ to ~ \tl_to_str:N \l__unravel_head_tl
5042             }
5043         }
5044         % ^^A todo: in the terminal output the token itself is missing.
5045         \__unravel_print_expansion:
5046         \__unravel_cond_pop:
5047     }
5048 }
5049 \cs_new_protected:Npn \__unravel_fi_or_else_loop:
5050 {
5051     \int_compare:nNnF \l__unravel_head_char_int = 2
5052     {
5053         \__unravel_pass_text:
5054         \__unravel_set_cmd:
5055         \__unravel_fi_or_else_loop:
5056     }
5057 }

```

2.15 User interaction

2.15.1 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.

```

5058 \tl_new:N \l__unravel_print_tl
5059 \sys_if_engine luatex:TF
5060 {
5061     \cs_new_protected:Npx \__unravel_print_normalize_null:
5062     {
5063         \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
5064         { \char_generate:nn { 0 } { 12 } }
5065         { \tl_to_str:n { ^^@ } }
5066     }
5067 }
5068 { \cs_new_protected:Npn \__unravel_print_normalize_null: { } }

```

(End definition for `__unravel_print_normalize_null:` and `\l__unravel_print_tl`.)

```

\__unravel_print:n
\__unravel_print:x
\__unravel_log:n
5069 \cs_new_protected:Npn \__unravel_print:n #1
5070 {
5071     \tl_set:Nn \l__unravel_print_tl {#1}
5072     \__unravel_print_normalize_null:

```

```

5073     \__unravel_exp_args:Nx \iow_term:n { \l__unravel_print_tl }
5074   }
5075   \cs_new_protected:Npn \__unravel_print:x
5076     { \__unravel_exp_args:Nx \__unravel_print:n }
5077   \cs_new_protected:Npn \__unravel_log:n #1
5078     {
5079       \tl_set:Nn \l__unravel_print_tl {#1}
5080       \__unravel_print_normalize_null:
5081       \__unravel_exp_args:Nx \iow_log:n { \l__unravel_print_tl }
5082     }

```

(End definition for __unravel_print:n and __unravel_log: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_online_int.

```

5083   \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
5084   {
5085     \int_compare:nNnF \g__unravel_online_int < 0
5086     {
5087       \int_compare:nNnTF \g__unravel_online_int = 0
5088         { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_log:n }
5089         { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_print:n }
5090     }
5091   }

```

(End definition for __unravel_print_message:nn.)

__unravel_set_action_text:x

```

5092   \cs_new_protected:Npn \__unravel_set_action_text:x #1
5093   {
5094     \group_begin:
5095     \__unravel_set_escapechar:n { 92 }
5096     \str_gset:Nx \g__unravel_action_text_str {#1}
5097     \group_end:
5098   }

```

(End definition for __unravel_set_action_text:x.)

__unravel_set_action_text:

```

5099   \cs_new_protected:Npn \__unravel_set_action_text:
5100   {
5101     \__unravel_set_action_text:x
5102     {
5103       \tl_to_str:N \l__unravel_head_tl
5104       \tl_if_single_token:VT \l__unravel_head_tl
5105       { = ~ \token_to_meaning:N \l__unravel_head_token }
5106     }
5107   }

```

(End definition for __unravel_set_action_text:.)

_unravel_print_state:

```
5108 \cs_new_protected:Npn \_unravel_print_state:
5109 {
5110   \group_begin:
5111     \_unravel_set_escapechar:n { 92 }
5112     \tl_use:N \g__unravel_before_print_state_tl
5113     \int_compare:nNnT \g__unravel_online_int > 0
5114     {
5115       \_unravel_print_state_output:
5116       \_unravel_print_state_prev:
5117       \_unravel_print_state_input:
5118     }
5119   \group_end:
5120 }
```

(End definition for _unravel_print_state:.)

_unravel_print_state_output:

_unravel_print_state_output:n

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.

```
5121 \cs_new_protected:Npn \_unravel_print_state_output:
5122 {
5123   \_unravel_exp_args:Nx \_unravel_print_state_output:n
5124   { \gtl_to_str:N \g__unravel_output_gtl }
5125 }
5126 \cs_new_protected:Npn \_unravel_print_state_output:n #1
5127 {
5128   \tl_if_empty:nF {#1}
5129   {
5130     \_unravel_print_message:nn { <| ~ }
5131     { \_unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
5132   }
5133 }
```

(End definition for _unravel_print_state_output: and _unravel_print_state_output:n.)

_unravel_print_state_prev: Never trim ##1.

```
5134 \cs_new_protected:Npn \_unravel_print_state_prev:
5135 {
5136   \seq_set_map:NnN \l__unravel_tmpa_seq \g__unravel_prev_input_seq
5137   { \_unravel_to_str:n {##1} }
5138   \seq_remove_all:Nn \l__unravel_tmpa_seq { }
5139   \seq_if_empty:NTF \l__unravel_tmpa_seq
5140   { \_unravel_print_message:nn { || ~ } { } }
5141   {
5142     \seq_map_inline:Nn \l__unravel_tmpa_seq
5143     {
5144       \_unravel_print_message:nn { || ~ } {##1}
5145     }
5146   }
5147 }
```

(End definition for _unravel_print_state_prev:.)

`__unravel_print_state_input:` Print #1 with each line starting with |>~. The `__unravel_str_truncate_right:nn` function trims #1 if needed, to fit in a maximum of `\g__unravel_max_input_int` characters.

```

5148 \cs_new_protected:Npn \__unravel_print_state_input:
5149 {
5150   \__unravel_exp_args:Nx \__unravel_print_state_input:n
5151   { \__unravel_input_to_str: }
5152 }
5153 \cs_new_protected:Npn \__unravel_print_state_input:n #1
5154 {
5155   \__unravel_print_message:nn { |> ~ }
5156   { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
5157 }

```

(End definition for `__unravel_print_state_input:` and `__unravel_print_state_input:n`.)

`__unravel_print_meaning:`

```

5158 \cs_new_protected:Npn \__unravel_print_meaning:
5159 {
5160   \__unravel_input_if_empty:TF
5161   { \__unravel_print_message:nn { } { Empty-input! } }
5162   {
5163     \__unravel_input_get:N \l__unravel_tmpb_gtl
5164     \__unravel_print_message:nn { }
5165     {
5166       \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
5167       = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
5168     }
5169   }
5170 }

```

(End definition for `__unravel_print_meaning:.`)

`__unravel_print_action:` Some of these commands are currently synonyms but we may decide to make some options act differently on them.

```

\__unravel_print_action:x
\__unravel_print_assignment:
\__unravel_print_expansion:
\__unravel_print_expansion:x
\__unravel_print_action_aux:N
5171 \cs_new_protected:Npn \__unravel_print_action:
5172 { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
5173 \cs_new_protected:Npn \__unravel_print_action:x #1
5174 {
5175   \__unravel_set_action_text:x {#1}
5176   \__unravel_print_action:
5177 }
5178 \cs_new_protected:Npn \__unravel_print_assignment:
5179 { \__unravel_print_action_aux:N \g__unravel_trace_assigns_bool }
5180 \cs_new_protected:Npn \__unravel_print_assignment:x #1
5181 {
5182   \__unravel_set_action_text:x {#1}
5183   \__unravel_print_assignment:
5184 }
5185 \cs_new_protected:Npn \__unravel_print_expansion:
5186 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
5187 \cs_new_protected:Npn \__unravel_print_expansion:x #1
5188 {
5189   \__unravel_set_action_text:x {#1}

```

```

5190     \__unravel_print_expansion:
5191   }
5192   \cs_new_protected:Npn \__unravel_print_action_aux:N #1
5193   {
5194     \int_gdecr:N \g__unravel_nonstop_int
5195     \int_gincr:N \g__unravel_step_int
5196     \bool_if:NT #1
5197     {
5198       \__unravel_print:x
5199       {
5200         [====
5201         \bool_if:NT \g__unravel_number_steps_bool
5202         { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
5203         =====]~
5204         \int_compare:nNnTF
5205         { \str_count:N \g__unravel_action_text_str }
5206         > { \g__unravel_max_action_int }
5207         {
5208           \str_range:Nnn \g__unravel_action_text_str
5209           { 1 } { \g__unravel_max_action_int - 3 } ...
5210         }
5211         { \g__unravel_action_text_str }
5212       }
5213       \__unravel_print_state:
5214       \__unravel_prompt:
5215     }
5216   }

```

(End definition for __unravel_print_action: and others.)

__unravel_print_assigned_token:
__unravel_print_assigned_register:

```

5217   \cs_new_protected:Npn \__unravel_print_assigned_token:
5218   {
5219     \__unravel_after_assignment: % ^^A todo: simplify
5220     \__unravel_print_assignment:x
5221     {
5222       Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5223       = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
5224     }
5225     \__unravel_omit_after_assignment:w
5226   }
5227   \cs_new_protected:Npn \__unravel_print_assigned_register:
5228   {
5229     \__unravel_after_assignment: % ^^A todo: simplify
5230     \__unravel_exp_args:Nx \__unravel_print_assignment:x
5231     {
5232       \exp_not:n
5233       {
5234         Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5235         \tl_if_single:NT \l__unravel_defined_tl
5236         { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
5237       }
5238       = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
5239     }

```

```

5240     \__unravel_omit_after_assignment:w
5241   }

(End definition for \__unravel_print_assigned_token: and \__unravel_print_assigned_register:.)

```

```

\__unravel_print_welcome: Welcome message.

5242 \cs_new_protected:Npn \__unravel_print_welcome:
5243 {
5244   \__unravel_print_message:nn { }
5245   {
5246     \bool_if:NTF \g__unravel_welcome_message_bool
5247     {
5248       \\\
5249       =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5250       \iow_indent:n
5251       {
5252         "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\
5253         "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\
5254         "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
5255         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
5256       }
5257     }
5258     { [=====Start=====] }
5259   }
5260   \__unravel_print_state:
5261   \__unravel_prompt:
5262 }

(End definition for \__unravel_print_welcome:.)

```

```

\__unravel_print_outcome: Final message.

5263 \cs_new_protected:Npn \__unravel_print_outcome:
5264 { \__unravel_print_message:nn { } { [=====End=====] } }

(End definition for \__unravel_print_outcome:.)

```

2.15.2 Prompt

```

\__unravel_prompt:

5265 \cs_new_protected:Npn \__unravel_prompt:
5266 {
5267   \int_compare:nNnF \g__unravel_nonstop_int > 0
5268   {
5269     \group_begin:
5270     \__unravel_set_escapechar:n { -1 }
5271     \int_set:Nn \tex_endlinechar:D { -1 }
5272     \tl_use:N \g__unravel_before_prompt_tl
5273     \__unravel_prompt_aux:
5274     \group_end:
5275   }
5276 }
5277 \cs_new_protected:Npn \__unravel_prompt_aux:
5278 {
5279   \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
5280   {

```

```

5281         \bool_if:NTF \g__unravel_explicit_prompt_bool
5282         { \ior_str_get:Nc \c__unravel_prompt_ior }
5283         { \ior_str_get:Nc \c__unravel_noprompt_ior }
5284         { Your~input }
5285         \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5286     }
5287 }
5288 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5289 {
5290     \tl_if_empty:nF {#1}
5291     {
5292         \__unravel_exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
5293         {
5294             { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
5295             { q }
5296             {
5297                 \int_gset:Nn \g__unravel_online_int { -1 }
5298                 \int_gzero:N \g__unravel_nonstop_int
5299             }
5300             { x }
5301             {
5302                 \group_end:
5303                 \exp_after:wN \__unravel_exit:w \__unravel_exit:w
5304             }
5305             { X } { \tex_batchmode:D \tex_end:D }
5306             { s } { \__unravel_prompt_scan_int:nn {#1}
5307                 \__unravel_prompt_silent_steps:n }
5308             { o } { \__unravel_prompt_scan_int:nn {#1}
5309                 { \int_gset:Nn \g__unravel_online_int } }
5310             { C }
5311             {
5312                 \__unravel_exp_args:Nx \use:n
5313                 {
5314                     \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5315                     { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5316                 }
5317                 \tl_gput_left:Nn \g__unravel_tmpc_tl
5318                 { \tl_gclear:N \g__unravel_tmpc_tl }
5319                 \group_insert_after:N \g__unravel_tmpc_tl
5320             }
5321             { | } { \__unravel_prompt_scan_int:nn {#1}
5322                 \__unravel_prompt_vert:n }
5323             { a } { \__unravel_prompt_all: }
5324         }
5325         { \__unravel_prompt_help: }
5326     }
5327 }
5328 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
5329 {
5330     \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
5331     \l__unravel_prompt_tmpa_int =
5332     \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5333     \use_ii:nn #1 \scan_stop:
5334 }

```

```

5335 \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5336 {
5337   #2 \l__unravel_prompt_tmpa_int
5338   \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
5339 }
5340 \cs_new_protected:Npn \__unravel_prompt_help:
5341 {
5342   \__unravel_print:n { "m":~meaning-of~first~token }
5343   \__unravel_print:n { "q":~semi-quiet~(same-as~"o-1") }
5344   \__unravel_print:n { "x"/"X":~exit~this~instance-of~unravel/TeX }
5345   \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
5346   \__unravel_print:n
5347   { "o<num>":~1~==>~log~and~terminal,~0~==>~only~log,~-1~==>~neither.}
5348   \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
5349   \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
5350   \__unravel_print:n { "a":~print~state~again,~without~truncating }
5351   \__unravel_prompt_aux:
5352 }
5353 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
5354 {
5355   \int_compare:nNnF {#1} < 0
5356   {
5357     \int_gset:Nn \g__unravel_online_int { -1 }
5358     \tl_gset:Nn \g__unravel_before_prompt_tl
5359     {
5360       \int_gset:Nn \g__unravel_online_int { 1 }
5361       \tl_gclear:N \g__unravel_before_prompt_tl
5362     }
5363     \int_gset:Nn \g__unravel_nonstop_int {#1}
5364   }
5365 }
5366 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5367 {
5368   \int_compare:nNnTF {#1} < { 0 }
5369   { \__unravel_prompt_vert:Nn > {#1} }
5370   { \__unravel_prompt_vert:Nn < {#1} }
5371 }
5372 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5373 {
5374   \int_gset:Nn \g__unravel_online_int { -1 }
5375   \tl_gset:Nf \g__unravel_before_print_state_tl
5376   {
5377     \exp_args:NNf \exp_stop_f: \int_compare:nNnTF
5378     { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5379     #1 { \__unravel_prev_input_count: }
5380     {
5381       \int_gset:Nn \g__unravel_nonstop_int
5382       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5383     }
5384     {
5385       \int_gset:Nn \g__unravel_online_int { 1 }
5386       \tl_gclear:N \g__unravel_before_print_state_tl
5387     }
5388   }

```

```

5389 }
5390 \cs_new_protected:Npn \__unravel_prompt_all:
5391 {
5392   \tl_gset:Nx \g__unravel_tmpc_tl
5393   {
5394     \exp_not:n
5395     {
5396       \tl_gclear:N \g__unravel_tmpc_tl
5397       \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5398       \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5399       \__unravel_print_state:
5400       \int_gdecr:N \g__unravel_nonstop_int
5401       \__unravel_prompt:
5402     }
5403     \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5404     \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5405   }
5406   \group_insert_after:N \g__unravel_tmpc_tl
5407 }
5408 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5409 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for __unravel_prompt:.)

2.15.3 Errors

__unravel_not_implemented:n

```

5410 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5411 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End definition for __unravel_not_implemented:n.)

__unravel_error:nnnnn
__unravel_error:nxxxx

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.

```

5412 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5413 {
5414   \group_begin:
5415   \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5416   \group_end:
5417 }
5418 \cs_new_protected:Npn \__unravel_error:nxxxx #1#2#3#4#5
5419 {
5420   \group_begin:
5421   \msg_error:nnxxxx { unravel } {#1} {#2} {#3} {#4} {#5}
5422   \group_end:
5423 }

```

(End definition for __unravel_error:nnnnn.)

__unravel_tex_msg_new:nnn

This stores a \TeX error message.

```

5424 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5425 {
5426   \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
5427   \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
5428 }

```

(End 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.

```

5429 \cs_new_protected:Npn \_unravel\_tex\_error:nn #1#2
5430 {
5431   \group_begin:
5432   \msg_error:nnxxx { unravel } { tex-error }
5433   { \tl_to_str:n {#2} }
5434   { \use:c { \_unravel\_tex\_msg\_error\_#1: } }
5435   { \use:c { \_unravel\_tex\_msg\_help\_#1: } }
5436   \group_end:
5437 }
5438 \cs_generate_variant:Nn \_unravel\_tex\_error:nn { nV }

```

(End 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.

```

5439 \cs_new_protected:Npn \_unravel\_tex\_fatal\_error:nn #1#2
5440 {
5441   \_unravel\_error:nxxxx { tex-fatal }
5442   { \tl_to_str:n {#2} }
5443   { \use:c { \_unravel\_tex\_msg\_error\_#1: } }
5444   { \use:c { \_unravel\_tex\_msg\_help\_#1: } }
5445   { }
5446 }
5447 \cs_generate_variant:Nn \_unravel\_tex\_fatal\_error:nn { nV }

```

(End definition for `_unravel_tex_fatal_error:nn`.)

2.16 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.

```

5448 \keys_define:nn { unravel/defaults }
5449 {
5450   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5451   internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5452   max-action .int_gset:N = \g__unravel_default_max_action_int ,
5453   max-output .int_gset:N = \g__unravel_default_max_output_int ,
5454   max-input .int_gset:N = \g__unravel_default_max_input_int ,
5455   number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
5456   online .int_gset:N = \g__unravel_default_online_int ,
5457   trace-assigns .bool_gset:N = \g__unravel_default_trace_assign_bool ,
5458   trace-expansion .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
5459   trace-other .bool_gset:N = \g__unravel_default_trace_other_bool ,
5460   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5461 }
5462 \keys_define:nn { unravel }
5463 {
5464   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,

```

```

5465     internal-debug    .bool_gset:N = \g__unravel_internal_debug_bool ,
5466     max-action        .int_gset:N = \g__unravel_max_action_int ,
5467     max-output        .int_gset:N = \g__unravel_max_output_int ,
5468     max-input         .int_gset:N = \g__unravel_max_input_int ,
5469     number-steps      .bool_gset:N = \g__unravel_number_steps_bool ,
5470     online            .int_gset:N = \g__unravel_online_int ,
5471     trace-assigns     .bool_gset:N = \g__unravel_trace_assigns_bool ,
5472     trace-expansion   .bool_gset:N = \g__unravel_trace_expansion_bool ,
5473     trace-other       .bool_gset:N = \g__unravel_trace_other_bool ,
5474     welcome-message   .bool_gset:N = \g__unravel_welcome_message_bool ,
5475 }

```

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.

```

5476 \tl_map_inline:nn { { /defaults } { } }
5477 {
5478   \keys_define:nn { unravel #1 }
5479   {
5480     machine      .meta:nn =
5481     { unravel #1 }
5482     {
5483       explicit-prompt = false ,
5484       internal-debug  = false ,
5485       max-action      = \c_max_int ,
5486       max-output      = \c_max_int ,
5487       max-input       = \c_max_int ,
5488       number-steps    = false ,
5489       welcome-message = false ,
5490     } ,
5491     mute         .meta:nn =
5492     { unravel #1 }
5493     {
5494       trace-assigns = false ,
5495       trace-expansion = false ,
5496       trace-other = false ,
5497       welcome-message = false ,
5498       online = -1 ,
5499     }
5500   }
5501 }

```

2.17 Main command

`\unravel` Simply call an underlying code-level command.

```

5502 \NewDocumentCommand \unravel { 0 { } +m } { \unravel:nn {#1} {#2} }

```

(End definition for `\unravel`. This function is documented on page 2.)

`\unravelsetup` Simply call an underlying code-level command.

```

5503 \NewDocumentCommand \unravelsetup { m } { \unravel_setup:n {#1} }

```

(End definition for `\unravelsetup`. This function is documented on page 2.)

\unravel_setup:n Set keys, updating both default values and current values.

```

5504 \cs_new_protected:Npn \unravel_setup:n #1
5505 {
5506   \keys_set:nn { unravel/defaults } {#1}
5507   \keys_set:nn { unravel } {#1}
5508 }

```

(End 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

__unravel:nn in \unravel and avoid re-initializing important variables. Initialize and setup keys.

__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.

```

5509 \cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
5510 \cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
5511 \cs_new_protected:Npn \__unravel:nn #1#2
5512 {
5513   \__unravel_init_key_vars:
5514   \keys_set:nn { unravel } {#1}
5515   \__unravel_init_vars:
5516   \__unravel_input_gset:n {#2}
5517   \__unravel_print_welcome:
5518   \__unravel_main_loop:
5519   \__unravel_exit_point:
5520   \__unravel_print_outcome:
5521   \__unravel_final_test:
5522   \__unravel_exit_point:
5523 }
5524 \cs_new_protected:Npn \unravel_get:nnN #1#2#3
5525 {
5526   \unravel:nn {#1} {#2}
5527   \tl_set:Nx #3 { \gtl_left_tl:N \g__unravel_output_gtl }
5528 }

```

(End 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).

```

5529 \cs_new_protected:Npn \__unravel_init_key_vars:
5530 {
5531   \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bo
5532   \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5533   \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5534   \int_gset_eq:NN \g__unravel_online_int \g__unravel_default_online_int
5535   \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
5536   \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
5537   \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
5538   \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
5539   \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5540   \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5541   \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int

```

```

5542     \int_gzero:N \g__unravel_nonstop_int
5543 }

```

(End 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.

```

5544 \cs_new_protected:Npn \__unravel_init_vars:
5545 {
5546     \seq_gclear:N \g__unravel_prev_input_seq
5547     \gtl_gclear:N \g__unravel_output_gtl
5548     \int_gzero:N \g__unravel_step_int
5549     \tl_gclear:N \g__unravel_if_limit_tl
5550     \int_gzero:N \g__unravel_if_limit_int
5551     \int_gzero:N \g__unravel_if_depth_int
5552     \gtl_gclear:N \g__unravel_after_assignment_gtl
5553     \bool_gset_true:N \g__unravel_set_box_allowed_bool
5554     \bool_gset_false:N \g__unravel_name_in_progress_bool
5555     \gtl_clear:N \l__unravel_after_group_gtl
5556 }

```

(End definition for __unravel_init_vars:.)

__unravel_main_loop: Loop forever, getting the next token (with expansion) and performing the corresponding command.

```

5557 \cs_new_protected:Npn \__unravel_main_loop:
5558 {
5559     \__unravel_get_x_next:
5560     \__unravel_set_cmd:
5561     \__unravel_do_step:
5562     \__unravel_main_loop:
5563 }

```

(End definition for __unravel_main_loop:.)

__unravel_final_test: Make sure that the \unravel finished correctly. The error message is a bit primitive.

```

\__unravel_final_bad:
5564 \cs_new_protected:Npn \__unravel_final_test:
5565 {
5566     \bool_if:nTF
5567     {
5568         \tl_if_empty_p:N \g__unravel_if_limit_tl
5569         && \int_compare_p:nNn \g__unravel_if_limit_int = 0
5570         && \int_compare_p:nNn \g__unravel_if_depth_int = 0
5571         && \seq_if_empty_p:N \g__unravel_prev_input_seq
5572     }
5573     { \__unravel_input_if_empty:TF { } { \__unravel_final_bad: } }
5574     { \__unravel_final_bad: }
5575 }
5576 \cs_new_protected:Npn \__unravel_final_bad:
5577 {
5578     \__unravel_error:nnnnn { internal }
5579     { the-last-unravel-finished-badly } { } { } { }
5580 }

```

(End definition for __unravel_final_test: and __unravel_final_bad:.)

2.18 Messages

```
5581 \msg_new:nnn { unravel } { unknown-primitive }
5582   { Internal~error:~the~primitive~'#1'~is~not~known. }
5583 \msg_new:nnn { unravel } { extra-fi-or-else }
5584   { Extra~fi,~or,~or~else. }
5585 \msg_new:nnn { unravel } { missing-dollar }
5586   { Missing~dollar~inserted. }
5587 \msg_new:nnn { unravel } { unknown-expandable }
5588   { Internal~error:~the~expandable~command~'#1'~is~not~known. }
5589 \msg_new:nnn { unravel } { missing-font-id }
5590   { Missing~font~identifier.~\iow_char:N\~nullfont~inserted. }
5591 \msg_new:nnn { unravel } { missing-rparen }
5592   { Missing~right~parenthesis~inserted~for~expression. }
5593 \msg_new:nnn { unravel } { missing-cs }
5594   { Missing~control~sequence.~\iow_char:N\~inaccessible~inserted. }
5595 \msg_new:nnn { unravel } { missing-box }
5596   { Missing~box~inserted. }
5597 \msg_new:nnn { unravel } { missing-to }
5598   { Missing~keyword~'to'~inserted. }
5599 \msg_new:nnn { unravel } { improper-leaders }
5600   { Leaders~not~followed~by~proper~glue. }
5601 \msg_new:nnn { unravel } { extra-close }
5602   { Extra~right~brace~or~\iow_char:N\~endgroup. }
5603 \msg_new:nnn { unravel } { off-save }
5604   { Something~is~wrong~with~groups. }
5605 \msg_new:nnn { unravel } { hrule-bad-mode }
5606   { \iow_char\~hrule~used~in~wrong~mode. }
5607 \msg_new:nnn { unravel } { invalid-mode }
5608   { Invalid~mode~for~this~command. }
5609 \msg_new:nnn { unravel } { color-stack-action-missing }
5610   { Missing~color~stack~action. }
5611 \msg_new:nnn { unravel } { action-type-missing }
5612   { Missing~action~type. }
5613 \msg_new:nnn { unravel } { identifier-type-missing }
5614   { Missing~identifier~type. }
5615 \msg_new:nnn { unravel } { destination-type-missing }
5616   { Missing~destination~type. }
5617 \msg_new:nnn { unravel } { erroneous-prefixes }
5618   { Prefixes~applied~to~non~assignment~command. }
5619 \msg_new:nnn { unravel } { improper-setbox }
5620   { \iow_char:N\~setbox~while~fetching~base~of~an~accent. }
5621 \msg_new:nnn { unravel } { after-advance }
5622   {
5623     Missing~register~after~\iow_char:N\~advance,~
5624     \iow_char:N\~multiply,~or~\iow_char:N\~divide.
5625   }
5626 \msg_new:nnn { unravel } { bad-unless }
5627   { \iow_char:N\~unless~not~followed~by~conditional. }
5628 \msg_new:nnn { unravel } { runaway-if }
5629   { Runaway~\iow_char:N\~if... }
5630 \msg_new:nnn { unravel } { runaway-macro-parameter }
5631   {
5632     Runaway~macro~parameter~\# #2~after \\\
```

```

5633     \iow_indent:n {#1}
5634   }
5635   \msg_new:nnn { unravel } { extra-or }
5636     { Extra~\iow_char:N\or. }
5637   \msg_new:nnn { unravel } { missing-equals }
5638     { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
5639   \msg_new:nnn { unravel } { internal }
5640     { Internal~error:~'#1'.~\ Please-report. }
5641   \msg_new:nnn { unravel } { not-implemented }
5642     { The~following~feature~is~not~implemented:~'#1'. }
5643   \msg_new:nnn { unravel } { endinput-ignored }
5644     { The~primitive~\iow_char:N\endinput~was~ignored. }
5645   \msg_new:nnn { unravel } { missing-something }
5646     { Something~is~missing,~sorry! }
5647   \msg_new:nnn { unravel } { nested-unravel }
5648     { The~\iow_char:N\unravel~command~may~not~be~nested. }
5649   \msg_new:nnnn { unravel } { tex-error }
5650     { TeX~sees~"#1"~and~throws~an~error:~\\\ \iow_indent:n {#2} }
5651     {
5652       \tl_if_empty:nTF {#3}
5653         { TeX~provides~no~further~help~for~this~error. }
5654         { TeX's~advice~is:~\\\ \iow_indent:n {#3} }
5655     }
5656   \msg_new:nnnn { unravel } { tex-fatal }
5657     { TeX~sees~"#1"~and~throws~a~fatal~error:~\\\ \iow_indent:n {#2} }
5658     {
5659       \tl_if_empty:nTF {#3}
5660         { TeX~provides~no~further~help~for~this~error. }
5661         { TeX's~advice~is:~\\\ \iow_indent:n {#3} }
5662     }

```

Some error messages from T_EX itself.

```

5663   \_unravel_tex_msg_new:nnn { incompatible-mag }
5664   {
5665     Incompatible~magnification~
5666     ( \int_to_arabic:n { \_unravel_mag: } );~
5667     the~previous~value~will~be~retained
5668   }
5669   {
5670     I~can~handle~only~one~magnification~ratio~per~job.~So~I've~
5671     reverted~to~the~magnification~you~used~earlier~on~this~run.
5672   }
5673   \_unravel_tex_msg_new:nnn { illegal-mag }
5674   {
5675     Illegal~magnification~has~been~changed~to~1000~
5676     ( \int_to_arabic:n { \_unravel_mag: } )
5677   }
5678   { The~magnification~ratio~must~be~between~1~and~32768. }
5679   \_unravel_tex_msg_new:nnn { missing-number }
5680   { Missing~number,~treated~as~zero }
5681   {
5682     A~number~should~have~been~here;~I~inserted~'0'.~
5683     If~you~can't~figure~out~why~I~needed~to~see~a~number,~
5684     look~up~'weird-error'~in~the~index~to~The~TeXbook.
5685   }

```

```

5686 \_unravel_tex_msg_new:nnn { the-cannot }
5687 { You~can't~use~'\tl_to_str:N\l\_unravel_head_tl'~after~\iow_char:N\the }
5688 { I'm~forgetting~what~you~said~and~using~zero~instead. }
5689 \_unravel_tex_msg_new:nnn { incompatible-units }
5690 { Incompatible~glue~units }
5691 { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }
5692 \_unravel_tex_msg_new:nnn { missing-mu }
5693 { Illegal~unit~of~measure~(mu~inserted) }
5694 {
5695   The~unit~of~measurement~in~math~glue~must~be~mu.~
5696   To~recover~gracefully~from~this~error,~it's~best~to~
5697   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5698   two~letters.~(See~Chapter~27~of~The~TeXbook.)
5699 }
5700 \_unravel_tex_msg_new:nnn { missing-pt }
5701 { Illegal~unit~of~measure~(pt~inserted) }
5702 {
5703   Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
5704   cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
5705   I'll~assume~that~you~meant~to~say~pt,~for~printer's~points.~
5706   To~recover~gracefully~from~this~error,~it's~best~to~
5707   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5708   two~letters.~(See~Chapter~27~of~The~TeXbook.)
5709 }
5710 \_unravel_tex_msg_new:nnn { missing-lbrace }
5711 { Missing~\iow_char:N\{~inserted }
5712 {
5713   A~left~brace~was~mandatory~here,~so~I've~put~one~in.~
5714   You~might~want~to~delete~and/or~insert~some~corrections~
5715   so~that~I~will~find~a~matching~right~brace~soon.~
5716   (If~you're~confused~by~all~this,~try~typing~'\iow_char:N\}'~now.)
5717 }
5718 \_unravel_tex_msg_new:nnn { extra-endcsname }
5719 { Extra~\token_to_str:c{endcsname} }
5720 { I'm~ignoring~this,~since~I~wasn't~doing~a~\token_to_str:c{csname}. }
5721 \_unravel_tex_msg_new:nnn { missing-endcsname }
5722 { Missing~\token_to_str:c{endcsname}~inserted }
5723 {
5724   The~control~sequence~marked~<to~be~read~again>~should~
5725   not~appear~between~\token_to_str:c{csname}~and~
5726   \token_to_str:c{endcsname}.
5727 }

```

Fatal T_EX error messages.

```

5728 \_unravel_tex_msg_new:nnn { cannot-read }
5729 { ***~(cannot~\iow_char:N\read~from~terminal~in~nonstop~modes) }
5730 { }
5731 \_unravel_tex_msg_new:nnn { file-error }
5732 { ***~(job~aborted,~file~error~in~nonstop~mode) }
5733 { }
5734 \_unravel_tex_msg_new:nnn { interwoven-preambles }
5735 { (interwoven~alignment~preambles~are~not~allowed) }
5736 { }

```

Restore catcodes to their original values.

```
5737 \_unravel_setup_restore:
5738 </package>
```