

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2019/03/23

Contents

1	unravel 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	unravel 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	15
2.1.6	Helpers concerning tokens	15
2.1.7	Helpers for previous input	18
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	50
2.6	Expand next token	51
2.7	Basic scanning subroutines	53
2.8	Working with boxes	72

*This file has version number 0.2g, last revised 2019/03/23.

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	86
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	98
2.13.7	Assignments	104
2.14	Expandable primitives	114
2.14.1	Conditionals	121
2.15	User interaction	130
2.15.1	Print	130
2.15.2	Prompt	135
2.15.3	Errors	138
2.16	Keys	140
2.17	Main command	141
2.18	Messages	143

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` [*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` {*options*}

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

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

See `\unravel`.

<code>\unravel_get:nnN</code>	<code>\unravel_get:nnN {<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.
<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

explicit-prompt

Boolean option (default `false`) determining whether to give an explicit prompt. If `true`, the text “Your input=” will appear at the beginning of lines where user input is expected.

internal-debug

Boolean option (default `false`) used to debug `unravel` itself.

machine

Option which takes no value and makes `unravel` produce an output that is somewhat more suitable for automatic processing. In particular, it sets `max-action`, `max-output`, `max-input` to very large values, and `number-steps` to `false`.

max-action
max-output
max-input

Integer options (defaults 50, 300, 300) determining the maximum number of characters displayed for the action, the output part, and the input part.

mute

Make none of the steps produce any output, by setting `trace-assigns`, `trace-expansion`, `trace-other`, `welcome-message` to `false`. This is only useful with `\unravel_get:nnN` or when other options change some of these settings.

number-steps

Boolean option (default `true`) determining whether to number steps.

<code>online</code>	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> <code>trace-expansion</code> <code>trace-other</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>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`, `\crrc`, `&`), many math mode primitives, and `\pdfprimitive`, `\discretionary`, as well as all primitives specific to engines other than `pdfTEX`. This list may sadly be incomplete!
- `\aftergroup` is only partially implemented.
- `\everyhbox`, `\everyvbox`, `\everymath`, `\everydisplay`, `\lastkern`, `\lastnodetype`, `\lastpenalty`, `\lastskip`, `\currentiflevel` and `\currentiftyp` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgroup` 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 %
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:nmn \expandafter
50     { \the \numexpr #1 + 1 } {#2} {#3}
51     \fi
52   }

```

(End definition for _unravel_setup_catcodes:nmn.)

_unravel_setup_latex: This saves the catcodes and related parameters, then sets them to the value they normally have in a L^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:nmn {0} {8} {15}
57   \catcode 9 = 10 ~
58   \catcode 10 = 12 ~
59   \catcode 11 = 15 ~
60   \catcode 12 = 13 ~
61   \catcode 13 = 5 ~
62   \_unravel_setup_catcodes:nmn {14} {31} {15}
63   \catcode 32 = 10 ~
64   \catcode 33 = 12 ~
65   \catcode 34 = 12 ~
66   \catcode 35 = 6 ~
67   \catcode 36 = 3 ~
68   \catcode 37 = 14 ~
69   \catcode 38 = 4 ~
70   \_unravel_setup_catcodes:nmn {39} {63} {12}
71   \_unravel_setup_catcodes:nmn {64} {90} {11}
72   \catcode 91 = 12 ~
73   \catcode 92 = 0 ~
74   \catcode 93 = 12 ~
75   \catcode 94 = 7 ~
76   \catcode 95 = 8 ~
77   \catcode 96 = 12 ~
78   \_unravel_setup_catcodes:nmn {97} {122} {11}
79   \catcode 123 = 1 ~
80   \catcode 124 = 12 ~
81   \catcode 125 = 2 ~
82   \catcode 126 = 13 ~
83   \catcode 127 = 15 ~
84   \_unravel_setup_catcodes:nmn {128} {255} {12}
85   \endlinechar = 13 ~
86   \escapechar = 92 ~
87   \newlinechar = 10 ~
88 }

```

(End definition for _unravel_setup_latex:.)

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

```

89 \protected \gdef \_unravel_setup_unravel:
90 {
91   \_unravel_setup_save:

```



```

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

```

(End 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:nmn`. 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:nmn}{\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 `TEX` 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} {2019/03/23} {0.2g} {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_everyeof:w
\__unravel_everypar:w
\__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 \gtl_if_empty:NTF { c }
161 \cs_generate_variant:Nn \gtl_if_tl:NT { 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 }
169 \cs_generate_variant:Nn \gtl_left_tl:N { c }

```

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

```

170 \cs_new_protected:Npn \__unravel_tl_if_in:ooTF #1#2#3#4
171 {
172   \group_begin:
173   \exp_args:Noo \tl_if_in:nnTF {#1} {#2}
174   { \group_end: #3 } { \group_end: #4 }
175 }

```

(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
176 \tl_new:N \l__unravel_exp_tl
177 \cs_new_protected:Npn \__unravel_exp_args:Nx #1#2
178 {
179   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 {#2} }
180   \l__unravel_exp_tl
181 }
182 \cs_new_protected:Npn \__unravel_exp_args:NNx #1#2#3
183 {
184   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 \exp_not:N #2 {#3} }
185   \l__unravel_exp_tl
186 }

```

(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.

```

187 \cs_new_protected:Npn \__unravel_tmp:w { }

```

(End definition for __unravel_tmp:w.)

```

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

```

```

196         \tex_input:D ##1 \scan_stop:
197     }
198     \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
199     {
200         \group_end:
201         \tl_set:Nx ##2
202         { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
203     }
204 }
205 \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
 __unravel_tl_first_int_aux:Nn

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

```

206 \cs_new:Npn \__unravel_tl_first_int:N #1
207 {
208     \int_eval:n
209     {
210         \exp_after:wN \__unravel_tl_first_int_aux:Nn
211         \exp_after:wN \__unravel_tl_first_int_aux:Nn
212         #1 ? 0 ? \q_stop
213     }
214 }
215 \cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
216 {
217     \tl_if_single:nT {#2}
218     {
219         \token_if_eq_catcode:NNT + #2
220         {
221             \if_int_compare:w 1 < 1 #2 \exp_stop_f:
222             #2
223             \exp_after:wN \use_i_ii:nnn
224             \exp_after:wN \__unravel_tl_first_int_aux:Nn
225             \exp_after:wN \use_none_delimit_by_q_stop:w
226             \fi:
227         }
228     }
229     #1
230 }

```

(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`.

```

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

```

```

235     \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
236     {
237         \__unravel_tex_error:nn { incompatible-mag } { }
238         \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
239     }
240 }
241 \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
242 {
243     \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
244     \int_gset:Nn \__unravel_mag: { 1000 }
245 }
246 \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
247 }

```

(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.

```

248 \cs_new:Npn \__unravel_strip_escape:w
249 {
250     \tex_romannumeral:D
251     \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
252     \__unravel_strip_escape_aux:N
253 }
254 \cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero_int }
255 \cs_new:Npn \__unravel_strip_escape_aux:w #1#2
256 { - \__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
257 \cs_new:Npn \__unravel_to_str:n #1
258 {
259     \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
260     { \__unravel_to_str_auxi:w #1 ? \q_stop }
261     { \tl_to_str:n }
262     {#1}
263 }
264 \cs_set:Npn \__unravel_tmp:w #1
265 {
266     \cs_new:Npn \__unravel_to_str_auxi:w ##1##2 \q_stop
267     {

```

```

268     \exp_after:wN \__unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark
269     #1 t1 \q_mark \q_stop
270   }
271   \cs_new:Npn \__unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
272   { \cs_if_exist_use:cF { __unravel_ ##2 _to_str:n } { \t1_to_str:n } }
273 }
274 \exp_args:No \__unravel_tmp:w { \t1_to_str:n { s _ _ } }
275 \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.

```

276 \cs_new:Npn \__unravel_str_truncate_left:nn #1#2
277 {
278   \exp_args:Nf \__unravel_str_truncate_left_aux:nnn
279   { \str_count:n {#1} } {#1} {#2}
280 }
281 \cs_new:Npn \__unravel_str_truncate_left_aux:nnn #1#2#3
282 {
283   \int_compare:nNnTF {#1} > {#3}
284   {
285     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
286     \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
287   }
288   { \t1_to_str:n {#2} }
289 }

```

(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.

```

290 \cs_new:Npn \__unravel_str_truncate_right:nn #1#2
291 {
292   \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
293   { \str_count:n {#1} } {#1} {#2}
294 }
295 \cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
296 {
297   \int_compare:nNnTF {#1} > {#3}
298   {
299     \str_range:nnn {#2} { 1 } { #3 - 25 } ~
300     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
301   }
302   { \t1_to_str:n {#2} }
303 }

```

(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`.

```

304 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
305 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }
306 \cs_new:Npn \__unravel_exit_error:w #1 \__unravel_exit_point: #2 \__unravel_final_bad:
307 {
308   \__unravel_error:nnnnn { runaway-unravel } { } { } { } { }
309   #2
310 }
311 \cs_new:Npn \__unravel_exit_hard:w #1 \__unravel_exit_point: #2 \__unravel_exit_point: { }

```

(End definition for `__unravel_exit:w` and others.)

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

```

312 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
313 \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).

```

314 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
315 {
316   \int_compare:nNnTF
317     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
318     { \prg_return_false: }
319     {
320       \int_compare:nNnTF
321         \l__unravel_head_cmd_int
322         > { \__unravel_tex_use:n { max_internal } }
323         { \prg_return_false: }
324         { \prg_return_true: }
325     }
326 }

```

(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), extract the character itself (with string category codes). This is somewhat robust against wrong input.

```

327 \cs_new:Npn \__unravel_meaning_to_char:n #1
328 { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
329 \cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
330 { \__unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
331 \cs_new:Npn \__unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
332 { \tl_if_empty:nTF {#2} { ~ } {#2} }
333 \cs_generate_variant:Nn \__unravel_meaning_to_char:n { o }
334 \cs_new:Npn \__unravel_token_to_char:N #1
335 { \__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:NTF` because the `expl3` one does not think that `undefined` is expandable.

```

336 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
337 { p , T , F , TF }
338 {
339   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
340   \prg_return_false:
341   \else:
342     \prg_return_true:
343   \fi:
344 }

```

(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.

```

345 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
346 { p , T , F , TF }
347 {
348   \_unravel_token_if_expandable:NTF #1
349   {
350     \token_if_protected_macro:NTF #1
351     { \prg_return_true: }
352     {
353       \token_if_protected_long_macro:NTF #1
354       { \prg_return_true: }
355       { \prg_return_false: }
356     }
357   }
358   { \prg_return_true: }
359 }

```

(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.

```

360 \group_begin:
361   \char_set_catcode_active:n { 'Z }
362   \prg_new_protected_conditional:Npnn \_unravel_token_if_active_char:N #1
363   { TF }
364   {
365     \group_begin:
366       \_unravel_exp_args:Nx \char_set_lccode:nn
367       { ' \exp_args:No \str_head:n { \token_to_str:N #1 } }
368       { ' Z }
369       \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
370       { \group_end: \prg_return_true: }
371       { \group_end: \prg_return_false: }
372     }
373   \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.

```

374 \group_begin:
375   \char_set_catcode_active:n { 'Z }
376   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
377     { TF }
378     {
379       \group_begin:
380         \_unravel_set_escapechar:n { 92 }
381         \tl_set:Nx \l__unravel_tmpa_tl
382           { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
383         \tl_if_empty:NTF \l__unravel_tmpa_tl
384           {
385             \_unravel_token_if_active_char:NTF #1
386             { \group_end: \prg_return_true: }
387             { \group_end: \prg_return_false: }
388           }
389           { \group_end: \prg_return_true: }
390       }
391   \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.

```

392 \prg_new_protected_conditional:Npnn \_unravel_gtl_if_head_is_definable:N #1
393   { TF , F }
394   {
395     \gtl_if_single_token:NTF #1
396     {
397       \gtl_if_head_is_N_type:NTF #1
398       {
399         \gtl_head_do:NN #1 \_unravel_token_if_definable:NTF
400         { \prg_return_true: }
401         { \prg_return_false: }
402       }
403       { \prg_return_false: }
404     }
405     { \prg_return_false: }
406   }

```

(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
407 \cs_new:Npn \__unravel_prev_input_count:
408 {
409   \int_eval:n
410   {
411     0
412     \seq_map_function:NN \g__unravel_prev_input_seq
413     \__unravel_prev_input_count_aux:n
414   }
415 }
416 \cs_new:Npn \__unravel_prev_input_count_aux:n #1
417 { \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
418 \cs_new_protected:Npn \__unravel_prev_input_get:N
419 { \seq_get_right:NN \g__unravel_prev_input_seq }
420 \cs_new_protected:Npn \__unravel_prev_input_gpush:
421 { \seq_gput_right:Nn \g__unravel_prev_input_seq { } }
422 \cs_new_protected:Npn \__unravel_prev_input_gpush:N
423 { \seq_gput_right:NV \g__unravel_prev_input_seq }
424 \cs_new_protected:Npn \__unravel_prev_input_gpop:N
425 { \seq_gpop_right:NN \g__unravel_prev_input_seq }
426 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
427 { \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl }
428 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
429 { \seq_gput_right:NV \g__unravel_prev_input_seq }
430 \cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
431 { \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
432 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
433 {
434   \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
435   \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
436   \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
437 }
438 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V }
439 \cs_new_protected:Npn \__unravel_prev_input_silent:x
440 { \__unravel_exp_args:Nx \__unravel_prev_input_silent:n }
441 \cs_new_protected:Npn \__unravel_prev_input:n #1
442 {
443   \__unravel_prev_input_silent:n {#1}
444   \__unravel_print_action:x { \tl_to_str:n {#1} }
445 }
446 \cs_generate_variant:Nn \__unravel_prev_input:n { V }
447 \cs_new_protected:Npn \__unravel_prev_input:x
448 { \__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`

```
449 \cs_new_protected:Npn \_unravel_prev_input_gtl:N #1
450 {
451   \_unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
452   \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
453   \_unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
454 }
```

(End definition for _unravel_prev_input_gtl:N.)

`_unravel_prev_input_join_get:nN`
`_unravel_join_get_aux:NNN`

Pops the previous-input sequence twice to get some value in `\l__unravel_head_tl` and some sign or decimal number in `\l__unravel_tmpa_tl`. Combines them into a value, using the appropriate evaluation function, determined based on #1.

```
455 \cs_new_protected:Npn \_unravel_prev_input_join_get:nN #1
456 {
457   \int_case:nnF {#1}
458   {
459     { 2 } { \_unravel_join_get_aux:NNN \skip_eval:n \tex_glueexpr:D }
460     { 3 } { \_unravel_join_get_aux:NNN \muskip_eval:n \tex_muexpr:D }
461   }
462   {
463     \_unravel_error:nnnnn { internal } { join-factor } { } { } { }
464     \_unravel_join_get_aux:NNN \use:n \prg_do_nothing:
465   }
466 }
467 \cs_new_protected:Npn \_unravel_join_get_aux:NNN #1#2#3
468 {
469   \_unravel_prev_input_gpop:N \l__unravel_head_tl
470   \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
471   \tl_set:Nx #3 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl } }
472 }
```

(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.

```
473 \tl_new:N \g_unravel_before_print_state_tl
474 \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`

```
475 \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.

```
476 \int_new:N \g_unravel_nonstop_int
```

(End definition for \g_unravel_nonstop_int.)

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

```

\g__unravel_default_explicit_prompt_bool 477 \bool_new:N \g__unravel_default_explicit_prompt_bool
\g__unravel_default_internal_debug_bool 478 \bool_new:N \g__unravel_default_internal_debug_bool
\g__unravel_default_number_steps_bool 479 \bool_new:N \g__unravel_default_number_steps_bool
\g__unravel_default_online_int 480 \int_new:N \g__unravel_default_online_int
\g__unravel_online_int 481 \bool_new:N \g__unravel_default_trace_assigns_bool
\g__unravel_default_trace_assigns_bool 482 \bool_new:N \g__unravel_default_trace_expansion_bool
\g__unravel_trace_assigns_bool 483 \bool_new:N \g__unravel_default_trace_other_bool
\g__unravel_default_trace_expansion_bool 484 \bool_new:N \g__unravel_default_welcome_message_bool
\g__unravel_trace_expansion_bool 485 \bool_gset_true:N \g__unravel_default_number_steps_bool
\g__unravel_default_trace_other_bool 486 \int_gset:Nn \g__unravel_default_online_int { 1 }
\g__unravel_trace_other_bool 487 \bool_gset_true:N \g__unravel_default_trace_assigns_bool
\g__unravel_default_welcome_message_bool 488 \bool_gset_true:N \g__unravel_default_trace_expansion_bool
\g__unravel_welcome_message_bool 489 \bool_gset_true:N \g__unravel_default_trace_other_bool
490 \bool_gset_true:N \g__unravel_default_welcome_message_bool
491 \bool_new:N \g__unravel_explicit_prompt_bool
492 \bool_new:N \g__unravel_internal_debug_bool
493 \bool_new:N \g__unravel_number_steps_bool
494 \int_new:N \g__unravel_online_int
495 \bool_new:N \g__unravel_trace_assigns_bool
496 \bool_new:N \g__unravel_trace_expansion_bool
497 \bool_new:N \g__unravel_trace_other_bool
498 \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.

```
499 \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.

```
500 \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 501 \int_new:N \g__unravel_default_max_action_int
\g__unravel_default_max_input_int 502 \int_new:N \g__unravel_default_max_output_int
\g__unravel_max_action_int 503 \int_new:N \g__unravel_default_max_input_int
\g__unravel_max_output_int 504 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
\g__unravel_max_input_int 505 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
506 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
507 \int_new:N \g__unravel_max_action_int
508 \int_new:N \g__unravel_max_output_int
509 \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.

```
510 \bool_new:N \g_unravel_speedup_macros_bool
511 \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.

```
512 \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 \TeX 's input stack, and allows us to manipulate smaller token lists, speeding up processing. The total number $\langle n \rangle$ of lists is `\g_unravel_input_int`. The highest numbered `gtl` represents input that comes to the left of lower numbered ones.

```
513 \int_new:N \g_unravel_input_int
```

(End definition for \g_unravel_input_int.)

`\g_unravel_input_tmpa_int`

`\l_unravel_input_tmpa_tl`

```
514 \int_new:N \g_unravel_input_tmpa_int
```

```
515 \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.

```
516 \seq_new:N \g_unravel_prev_input_seq
```

```
517 \tl_new:N \l_unravel_prev_input_tl
```

```
518 \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 \TeX 's digestion.

```
519 \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 \TeX .

```
520 \gtl_new:N \l_unravel_head_gtl
```

```
521 \tl_new:N \l_unravel_head_tl
```

```
522 \cs_new_eq:NN \l_unravel_head_token ?
```

```
523 \int_new:N \l_unravel_head_cmd_int
```

```
524 \int_new:N \l_unravel_head_char_int
```

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

`\l__unravel_head_meaning_tl`

525 `\tl_new:N \l__unravel_head_meaning_tl`

(End definition for `\l__unravel_head_meaning_tl`.)

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

`\l__unravel_argii_tl`

526 `\tl_new:N \l__unravel_argi_tl`

527 `\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
`\l__unravel_tmpb_gtl` unwanted tokens.

`\g__unravel_tmpc_tl`

528 `\tl_new:N \l__unravel_tmpa_tl`

`\l__unravel_tmpa_seq`

529 `\gtl_new:N \l__unravel_unused_gtl`

`\l__unravel_unused_gtl`

530 `\gtl_new:N \l__unravel_tmpb_gtl`

`\l__unravel_tmpb_token`

531 `\tl_new:N \g__unravel_tmpc_tl`

532 `\seq_new:N \l__unravel_tmpa_seq`

533 `\cs_new_eq:NN \l__unravel_tmpb_token ?`

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

`\l__unravel_defined_tl`

`\l__unravel_defining_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.

534 `\tl_new:N \l__unravel_defined_tl`

535 `\tl_new:N \l__unravel_defining_tl`

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

`__unravel_inaccessible:w`

536 `\cs_new_eq:NN __unravel_inaccessible:w ?`

(End definition for `__unravel_inaccessible:w`.)

`\g__unravel_after_assignment_gtl`

`\g__unravel_set_box_allowed_bool`

`\g__unravel_name_in_progress_bool`

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

537 `\gtl_new:N \g__unravel_after_assignment_gtl`

538 `\bool_new:N \g__unravel_set_box_allowed_bool`

539 `\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.

540 `\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.

```
541 \group_begin:
542   \cs_set:Npx \__unravel_tmp:w #1 { \c_hash_str #1 }
543   \tl_const:Nx \c__unravel_parameters_tl
544     { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
545 \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.

```
546 \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 547 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 548 \int_new:N \g__unravel_if_limit_int
549 \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`

```
550 \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.

```
551 \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.

```
552 \int_new:N \g__unravel_ends_int
553 \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 554 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 555 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 556 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 557 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 558 \tl_const:Nn \c__unravel_lq_tl { ‘ }
\c__unravel_dq_tl 559 \tl_const:Nn \c__unravel_rq_tl { ’ }
\c__unravel_lp_tl 560 \tl_const:Nn \c__unravel_dq_tl { " }
\c__unravel_rp_tl 561 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 562 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 563 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 564 \tl_const:Nn \c__unravel_comma_tl { , }
565 \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:.
566 \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`.

```

567 \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 568 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
569 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
570 \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
571 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
572 {
573   \tl_const:cx { c__unravel_tex_#1_tl }
574   { { \__unravel_tex_use:n {#2} } {#3} }
575 }

```

(End definition for `__unravel_tex_primitive:nnn`.)


```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
576 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
577 {
578   \cs_new_protected:cpn
579     { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
580 }
581 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
582 {
583   \cs_new_eq:cc
584     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
585     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
586 }

```

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

```

\__unravel_new_tex_expandable:nn
587 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
588 {
589   \cs_new_protected:cpn
590     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
591 }

```

(End definition for `__unravel_new_tex_expandable:nn`.)

Contrarily to \TeX , all macros are call, no `long_call` and the like.

```

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

```

```

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

```

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

980 \_unravel_tex_primitive:nnn { uchyph } { assign_int } { 38 }
981 \_unravel_tex_primitive:nnn { outputpenalty } { assign_int } { 39 }
982 \_unravel_tex_primitive:nnn { maxdeadcycles } { assign_int } { 40 }
983 \_unravel_tex_primitive:nnn { hangafter } { assign_int } { 41 }
984 \_unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
985 \_unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
986 \_unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
987 \_unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
988 \_unravel_tex_primitive:nnn { defaultshyphenchar } { assign_int } { 46 }
989 \_unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
990 \_unravel_tex_primitive:nnn { endlinesechar } { assign_int } { 48 }
991 \_unravel_tex_primitive:nnn { newlinesechar } { assign_int } { 49 }
992 \_unravel_tex_primitive:nnn { language } { assign_int } { 50 }
993 \_unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
994 \_unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
995 \_unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
996 \_unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
997 \_unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
998 \_unravel_tex_primitive:nnn { pdfcompresslevel } { assign_int } { 56 }
999 \_unravel_tex_primitive:nnn { pdfdecimaldigits } { assign_int } { 57 }
1000 \_unravel_tex_primitive:nnn { pdfmovechars } { assign_int } { 58 }
1001 \_unravel_tex_primitive:nnn { pdfimageresolution } { assign_int } { 59 }
1002 \_unravel_tex_primitive:nnn { pdfpkresolution } { assign_int } { 60 }
1003 \_unravel_tex_primitive:nnn { pdfuniqueresname } { assign_int } { 61 }
1004 \_unravel_tex_primitive:nnn
1005 { pdfoptionalwaysusepdfpagebox } { assign_int } { 62 }
1006 \_unravel_tex_primitive:nnn
1007 { pdfoptionpdfinclusionerrorlevel } { assign_int } { 63 }
1008 \_unravel_tex_primitive:nnn
1009 { pdfoptionpdfminorversion } { assign_int } { 64 }
1010 \_unravel_tex_primitive:nnn { pdfminorversion } { assign_int } { 64 }
1011 \_unravel_tex_primitive:nnn { pdfforcepagebox } { assign_int } { 65 }
1012 \_unravel_tex_primitive:nnn { pdfpagebox } { assign_int } { 66 }
1013 \_unravel_tex_primitive:nnn
1014 { pdfinclusionerrorlevel } { assign_int } { 67 }
1015 \_unravel_tex_primitive:nnn { pdfgamma } { assign_int } { 68 }
1016 \_unravel_tex_primitive:nnn { pdfimagegamma } { assign_int } { 69 }
1017 \_unravel_tex_primitive:nnn { pdfimagehicolor } { assign_int } { 70 }
1018 \_unravel_tex_primitive:nnn { pdfimageapplygamma } { assign_int } { 71 }
1019 \_unravel_tex_primitive:nnn { pdfadjustspacing } { assign_int } { 72 }
1020 \_unravel_tex_primitive:nnn { pdfprotrudechars } { assign_int } { 73 }
1021 \_unravel_tex_primitive:nnn { pdftracingfonts } { assign_int } { 74 }
1022 \_unravel_tex_primitive:nnn { pdfobjcompresslevel } { assign_int } { 75 }
1023 \_unravel_tex_primitive:nnn
1024 { pdfadjustinterwordglue } { assign_int } { 76 }
1025 \_unravel_tex_primitive:nnn { pdfprependkern } { assign_int } { 77 }
1026 \_unravel_tex_primitive:nnn { pdfappendkern } { assign_int } { 78 }
1027 \_unravel_tex_primitive:nnn { pdfgentounicode } { assign_int } { 79 }
1028 \_unravel_tex_primitive:nnn { pdfdraftmode } { assign_int } { 80 }
1029 \_unravel_tex_primitive:nnn { pdfinclusioncopyfonts } { assign_int } { 81 }
1030 \_unravel_tex_primitive:nnn { tracingassigns } { assign_int } { 82 }
1031 \_unravel_tex_primitive:nnn { tracinggroups } { assign_int } { 83 }
1032 \_unravel_tex_primitive:nnn { tracingifs } { assign_int } { 84 }
1033 \_unravel_tex_primitive:nnn { tracingscantokens } { assign_int } { 85 }

```

```

1034 \_unravel_tex_primitive:nnn { tracingnesting } { assign_int } { 86 }
1035 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
1036 \_unravel_tex_primitive:nnn { lastlinefit } { assign_int } { 88 }
1037 \_unravel_tex_primitive:nnn { savingdiscards } { assign_int } { 89 }
1038 \_unravel_tex_primitive:nnn { savinghyphcodes } { assign_int } { 90 }
1039 \_unravel_tex_primitive:nnn { TeXxETstate } { assign_int } { 91 }
1040 \_unravel_tex_primitive:nnn { parindent } { assign_dimen } { 0 }
1041 \_unravel_tex_primitive:nnn { mathsurround } { assign_dimen } { 1 }
1042 \_unravel_tex_primitive:nnn { lineskiplimit } { assign_dimen } { 2 }
1043 \_unravel_tex_primitive:nnn { hsize } { assign_dimen } { 3 }
1044 \_unravel_tex_primitive:nnn { vsize } { assign_dimen } { 4 }
1045 \_unravel_tex_primitive:nnn { maxdepth } { assign_dimen } { 5 }
1046 \_unravel_tex_primitive:nnn { splitmaxdepth } { assign_dimen } { 6 }
1047 \_unravel_tex_primitive:nnn { boxmaxdepth } { assign_dimen } { 7 }
1048 \_unravel_tex_primitive:nnn { hfuzz } { assign_dimen } { 8 }
1049 \_unravel_tex_primitive:nnn { vfuzz } { assign_dimen } { 9 }
1050 \_unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
1051 \_unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
1052 \_unravel_tex_primitive:nnn { scriptspace } { assign_dimen } { 12 }
1053 \_unravel_tex_primitive:nnn { predisplaysize } { assign_dimen } { 13 }
1054 \_unravel_tex_primitive:nnn { displaywidth } { assign_dimen } { 14 }
1055 \_unravel_tex_primitive:nnn { displayindent } { assign_dimen } { 15 }
1056 \_unravel_tex_primitive:nnn { overfullrule } { assign_dimen } { 16 }
1057 \_unravel_tex_primitive:nnn { hangindent } { assign_dimen } { 17 }
1058 \_unravel_tex_primitive:nnn { hoffset } { assign_dimen } { 18 }
1059 \_unravel_tex_primitive:nnn { voffset } { assign_dimen } { 19 }
1060 \_unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
1061 \_unravel_tex_primitive:nnn { pdfhorigin } { assign_dimen } { 21 }
1062 \_unravel_tex_primitive:nnn { pdfvorigin } { assign_dimen } { 22 }
1063 \_unravel_tex_primitive:nnn { pdfpagewidth } { assign_dimen } { 23 }
1064 \_unravel_tex_primitive:nnn { pdfpageheight } { assign_dimen } { 24 }
1065 \_unravel_tex_primitive:nnn { pdflinkmargin } { assign_dimen } { 25 }
1066 \_unravel_tex_primitive:nnn { pdfdestmargin } { assign_dimen } { 26 }
1067 \_unravel_tex_primitive:nnn { pdfthreadmargin } { assign_dimen } { 27 }
1068 \_unravel_tex_primitive:nnn { pdffirstlineheight } { assign_dimen } { 28 }
1069 \_unravel_tex_primitive:nnn { pdflastlinedepth } { assign_dimen } { 29 }
1070 \_unravel_tex_primitive:nnn { pdfeachlineheight } { assign_dimen } { 30 }
1071 \_unravel_tex_primitive:nnn { pdfeachlinedepth } { assign_dimen } { 31 }
1072 \_unravel_tex_primitive:nnn { pdfignoreddimen } { assign_dimen } { 32 }
1073 \_unravel_tex_primitive:nnn { pdfpxdimen } { assign_dimen } { 33 }
1074 \_unravel_tex_primitive:nnn { lineskip } { assign_glue } { 0 }
1075 \_unravel_tex_primitive:nnn { baselineskip } { assign_glue } { 1 }
1076 \_unravel_tex_primitive:nnn { parskip } { assign_glue } { 2 }
1077 \_unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue } { 3 }
1078 \_unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue } { 4 }
1079 \_unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
1080 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1081 \_unravel_tex_primitive:nnn { leftskip } { assign_glue } { 7 }
1082 \_unravel_tex_primitive:nnn { rightskip } { assign_glue } { 8 }
1083 \_unravel_tex_primitive:nnn { topskip } { assign_glue } { 9 }
1084 \_unravel_tex_primitive:nnn { splittopskip } { assign_glue } { 10 }
1085 \_unravel_tex_primitive:nnn { tabskip } { assign_glue } { 11 }
1086 \_unravel_tex_primitive:nnn { spaceskip } { assign_glue } { 12 }
1087 \_unravel_tex_primitive:nnn { xspaceskip } { assign_glue } { 13 }

```

```

1088 \_unravel_tex_primitive:nnn { parfillskip           } { assign_glue } { 14 }
1089 \_unravel_tex_primitive:nnn { thinmuskip         } { assign_mu_glue } { 15 }
1090 \_unravel_tex_primitive:nnn { medmuskip         } { assign_mu_glue } { 16 }
1091 \_unravel_tex_primitive:nnn { thickmuskip      } { assign_mu_glue } { 17 }
1092 \_unravel_tex_primitive:nnn { fontdimen        } { assign_font_dimen } { 0 }
1093 \_unravel_tex_primitive:nnn { hyphenchar       } { assign_font_int } { 0 }
1094 \_unravel_tex_primitive:nnn { skewchar         } { assign_font_int } { 1 }
1095 \_unravel_tex_primitive:nnn { lpcode           } { assign_font_int } { 2 }
1096 \_unravel_tex_primitive:nnn { rpcode           } { assign_font_int } { 3 }
1097 \_unravel_tex_primitive:nnn { efcodes          } { assign_font_int } { 4 }
1098 \_unravel_tex_primitive:nnn { tagcode          } { assign_font_int } { 5 }
1099 \_unravel_tex_primitive:nnn { pdfnoligatures  } { assign_font_int } { 6 }
1100 \_unravel_tex_primitive:nnn { knbscode         } { assign_font_int } { 7 }
1101 \_unravel_tex_primitive:nnn { stbscode         } { assign_font_int } { 8 }
1102 \_unravel_tex_primitive:nnn { shbscode         } { assign_font_int } { 9 }
1103 \_unravel_tex_primitive:nnn { knbccode         } { assign_font_int } { 10 }
1104 \_unravel_tex_primitive:nnn { knaccode         } { assign_font_int } { 11 }
1105 \_unravel_tex_primitive:nnn { spacefactor     } { set_aux } { 102 }
1106 \_unravel_tex_primitive:nnn { prevdepth       } { set_aux } { 1 }
1107 \_unravel_tex_primitive:nnn { prevgraf        } { set_prev_graf } { 0 }
1108 \_unravel_tex_primitive:nnn { pagegoal        } { set_page_dimen } { 0 }
1109 \_unravel_tex_primitive:nnn { pagetotal       } { set_page_dimen } { 1 }
1110 \_unravel_tex_primitive:nnn { pagestretch     } { set_page_dimen } { 2 }
1111 \_unravel_tex_primitive:nnn { pagefilstretch  } { set_page_dimen } { 3 }
1112 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 4 }
1113 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 5 }
1114 \_unravel_tex_primitive:nnn { pageshrink      } { set_page_dimen } { 6 }
1115 \_unravel_tex_primitive:nnn { pagedepth      } { set_page_dimen } { 7 }
1116 \_unravel_tex_primitive:nnn { deadcycles      } { set_page_int } { 0 }
1117 \_unravel_tex_primitive:nnn { insertpenalties } { set_page_int } { 1 }
1118 \_unravel_tex_primitive:nnn { interactionmode } { set_page_int } { 2 }
1119 \_unravel_tex_primitive:nnn { wd              } { set_box_dimen } { 1 }
1120 \_unravel_tex_primitive:nnn { dp              } { set_box_dimen } { 2 }
1121 \_unravel_tex_primitive:nnn { ht              } { set_box_dimen } { 3 }
1122 \_unravel_tex_primitive:nnn { parshape        } { set_shape } { 0 }
1123 \_unravel_tex_primitive:nnn { interlinepenalties } { set_shape } { 1 }
1124 \_unravel_tex_primitive:nnn { clubpenalties  } { set_shape } { 2 }
1125 \_unravel_tex_primitive:nnn { widowpenalties } { set_shape } { 3 }
1126 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
1127 \_unravel_tex_primitive:nnn { catcode         } { def_code } { 0 }
1128 \_unravel_tex_primitive:nnn { lccode          } { def_code } { 256 }
1129 \_unravel_tex_primitive:nnn { uccode          } { def_code } { 512 }
1130 \_unravel_tex_primitive:nnn { sfcodes         } { def_code } { 768 }
1131 \_unravel_tex_primitive:nnn { mathcode        } { def_code } { 1024 }
1132 \_unravel_tex_primitive:nnn { delcode         } { def_code } { 1591 }
1133 \_unravel_tex_primitive:nnn { textfont        } { def_family } { -48 }
1134 \_unravel_tex_primitive:nnn { scriptfont      } { def_family } { -32 }
1135 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family } { -16 }
1136 \_unravel_tex_primitive:nnn { nullfont        } { set_font } { 0 }
1137 \_unravel_tex_primitive:nnn { font            } { def_font } { 0 }
1138 \_unravel_tex_primitive:nnn { count           } { register } { 1 000 000 }
1139 \_unravel_tex_primitive:nnn { dimen           } { register } { 2 000 000 }
1140 \_unravel_tex_primitive:nnn { skip            } { register } { 3 000 000 }
1141 \_unravel_tex_primitive:nnn { muskip          } { register } { 4 000 000 }

```

```

1142 \_unravel_tex_primitive:nnn { advance } { advance } { 0 }
1143 \_unravel_tex_primitive:nnn { multiply } { multiply } { 0 }
1144 \_unravel_tex_primitive:nnn { divide } { divide } { 0 }
1145 \_unravel_tex_primitive:nnn { long } { prefix } { 1 }
1146 \_unravel_tex_primitive:nnn { outer } { prefix } { 2 }
1147 \_unravel_tex_primitive:nnn { global } { prefix } { 4 }
1148 \_unravel_tex_primitive:nnn { protected } { prefix } { 8 }
1149 \_unravel_tex_primitive:nnn { let } { let } { 0 }
1150 \_unravel_tex_primitive:nnn { futurelet } { let } { 1 }
1151 \_unravel_tex_primitive:nnn { chardef } { shorthand_def } { 0 }
1152 \_unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
1153 \_unravel_tex_primitive:nnn { countdef } { shorthand_def } { 2 }
1154 \_unravel_tex_primitive:nnn { dimendef } { shorthand_def } { 3 }
1155 \_unravel_tex_primitive:nnn { skipdef } { shorthand_def } { 4 }
1156 \_unravel_tex_primitive:nnn { muskipdef } { shorthand_def } { 5 }
1157 \_unravel_tex_primitive:nnn { toksdef } { shorthand_def } { 6 }
1158 \_unravel_tex_primitive:nnn { read } { read_to_cs } { 0 }
1159 \_unravel_tex_primitive:nnn { readline } { read_to_cs } { 1 }
1160 \_unravel_tex_primitive:nnn { def } { def } { 0 }
1161 \_unravel_tex_primitive:nnn { gdef } { def } { 1 }
1162 \_unravel_tex_primitive:nnn { edef } { def } { 2 }
1163 \_unravel_tex_primitive:nnn { xdef } { def } { 3 }
1164 \_unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1165 \_unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1166 \_unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1167 \_unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1168 \_unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1169 \_unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1170 \_unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1171 \_unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1172 \_unravel_tex_primitive:nnn { pdfcopyfont } { pdf_copy_font } { 0 }
1173 \_unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1174 \_unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1175 \_unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1176 \_unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1177 \_unravel_tex_primitive:nnn { pdfprimitive } { no_expand } { 1 }
1178 \_unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1179 \_unravel_tex_primitive:nnn { input } { input } { 0 }
1180 \_unravel_tex_primitive:nnn { endinput } { input } { 1 }
1181 \_unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1182 \_unravel_tex_primitive:nnn { if } { if_test } { 0 }
1183 \_unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }
1184 \_unravel_tex_primitive:nnn { ifnum } { if_test } { 2 }
1185 \_unravel_tex_primitive:nnn { ifdim } { if_test } { 3 }
1186 \_unravel_tex_primitive:nnn { ifodd } { if_test } { 4 }
1187 \_unravel_tex_primitive:nnn { ifvmode } { if_test } { 5 }
1188 \_unravel_tex_primitive:nnn { ifhmode } { if_test } { 6 }
1189 \_unravel_tex_primitive:nnn { ifmmode } { if_test } { 7 }
1190 \_unravel_tex_primitive:nnn { ifinner } { if_test } { 8 }
1191 \_unravel_tex_primitive:nnn { ifvoid } { if_test } { 9 }
1192 \_unravel_tex_primitive:nnn { ifhbox } { if_test } { 10 }
1193 \_unravel_tex_primitive:nnn { ifvbox } { if_test } { 11 }
1194 \_unravel_tex_primitive:nnn { ifx } { if_test } { 12 }
1195 \_unravel_tex_primitive:nnn { ifeof } { if_test } { 13 }

```

```

1196 \__unravel_tex_primitive:nnn { iftrue } { if_test } { 14 }
1197 \__unravel_tex_primitive:nnn { iffalse } { if_test } { 15 }
1198 \__unravel_tex_primitive:nnn { ifcase } { if_test } { 16 }
1199 \__unravel_tex_primitive:nnn { ifdefined } { if_test } { 17 }
1200 \__unravel_tex_primitive:nnn { ifcsname } { if_test } { 18 }
1201 \__unravel_tex_primitive:nnn { iffontchar } { if_test } { 19 }
1202 \__unravel_tex_primitive:nnn { ifincsname } { if_test } { 20 }
1203 \__unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1204 \__unravel_tex_primitive:nnn { ifpdfabsnum } { if_test } { 22 }
1205 \__unravel_tex_primitive:nnn { ifpdfabsdim } { if_test } { 23 }
1206 \__unravel_tex_primitive:nnn { fi } { fi_or_else } { 2 }
1207 \__unravel_tex_primitive:nnn { else } { fi_or_else } { 3 }
1208 \__unravel_tex_primitive:nnn { or } { fi_or_else } { 4 }
1209 \__unravel_tex_primitive:nnn { csname } { cs_name } { 0 }
1210 \__unravel_tex_primitive:nnn { number } { convert } { 0 }
1211 \__unravel_tex_primitive:nnn { romannumeral } { convert } { 1 }
1212 \__unravel_tex_primitive:nnn { string } { convert } { 2 }
1213 \__unravel_tex_primitive:nnn { meaning } { convert } { 3 }
1214 \__unravel_tex_primitive:nnn { fontname } { convert } { 4 }
1215 \__unravel_tex_primitive:nnn { eTeXrevision } { convert } { 5 }
1216 \__unravel_tex_primitive:nnn { pdftexrevision } { convert } { 6 }
1217 \__unravel_tex_primitive:nnn { pdftexbanner } { convert } { 7 }
1218 \__unravel_tex_primitive:nnn { pdffontname } { convert } { 8 }
1219 \__unravel_tex_primitive:nnn { pdffontobjnum } { convert } { 9 }
1220 \__unravel_tex_primitive:nnn { pdffontsize } { convert } { 10 }
1221 \__unravel_tex_primitive:nnn { pdfpageref } { convert } { 11 }
1222 \__unravel_tex_primitive:nnn { pdfxformname } { convert } { 12 }
1223 \__unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
1224 \__unravel_tex_primitive:nnn { pdfescapename } { convert } { 14 }
1225 \__unravel_tex_primitive:nnn { leftmarginkern } { convert } { 15 }
1226 \__unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
1227 \__unravel_tex_primitive:nnn
1228 { \sys_if_engine_xetex:F { pdf } strcmp } { convert } { 17 }
1229 \__unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
1230 \__unravel_tex_primitive:nnn { pdfescapehex } { convert } { 19 }
1231 \__unravel_tex_primitive:nnn { pdfunescapehex } { convert } { 20 }
1232 \__unravel_tex_primitive:nnn { pdfcreationdate } { convert } { 21 }
1233 \__unravel_tex_primitive:nnn { pdffilemoddate } { convert } { 22 }
1234 \__unravel_tex_primitive:nnn { pdffilesize } { convert } { 23 }
1235 \__unravel_tex_primitive:nnn { pdfmdfivesum } { convert } { 24 }
1236 \__unravel_tex_primitive:nnn { pdffiledump } { convert } { 25 }
1237 \__unravel_tex_primitive:nnn { pdfmatch } { convert } { 26 }
1238 \__unravel_tex_primitive:nnn { pdflastmatch } { convert } { 27 }
1239 \__unravel_tex_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
1240 \__unravel_tex_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
1241 \__unravel_tex_primitive:nnn { pdfinsertht } { convert } { 30 }
1242 \__unravel_tex_primitive:nnn { pdfximagebbox } { convert } { 31 }
1243 \__unravel_tex_primitive:nnn { jobname } { convert } { 32 }
1244 \sys_if_engine_luatex:T
1245 {
1246 \__unravel_tex_primitive:nnn { directlua } { convert } { 33 }
1247 \__unravel_tex_primitive:nnn { expanded } { convert } { 34 }
1248 \__unravel_tex_primitive:nnn { luaescapestring } { convert } { 35 }
1249 }

```

```

1250 \sys_if_engine_xetex:T
1251 {
1252   \__unravel_tex_primitive:nnn { Ucharcat          } { convert } { 40 }
1253 }
1254 \__unravel_tex_primitive:nnn { the                } { the } { 0 }
1255 \__unravel_tex_primitive:nnn { unexpanded        } { the } { 1 }
1256 \__unravel_tex_primitive:nnn { detokenize        } { the } { 5 }
1257 \__unravel_tex_primitive:nnn { topmark           } { top_bot_mark } { 0 }
1258 \__unravel_tex_primitive:nnn { firstmark         } { top_bot_mark } { 1 }
1259 \__unravel_tex_primitive:nnn { botmark           } { top_bot_mark } { 2 }
1260 \__unravel_tex_primitive:nnn { splitfirstmark    } { top_bot_mark } { 3 }
1261 \__unravel_tex_primitive:nnn { splitbotmark      } { top_bot_mark } { 4 }
1262 \__unravel_tex_primitive:nnn { topmarks          } { top_bot_mark } { 5 }
1263 \__unravel_tex_primitive:nnn { firstmarks        } { top_bot_mark } { 6 }
1264 \__unravel_tex_primitive:nnn { botmarks          } { top_bot_mark } { 7 }
1265 \__unravel_tex_primitive:nnn { splitfirstmarks    } { top_bot_mark } { 8 }
1266 \__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.

```

1267 \cs_new_protected:Npn \__unravel_get_next:
1268 {
1269   \__unravel_input_if_empty:TF
1270   { \__unravel_exit_error:w }
1271   {
1272     \__unravel_input_gpop:N \l__unravel_head_gtl
1273     \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1274     \gtl_if_tl:NNTF \l__unravel_head_gtl
1275     {
1276       \tl_set:Nx \l__unravel_head_tl
1277       { \gtl_head:N \l__unravel_head_gtl }
1278       \token_if_eq_meaning:NNT
1279       \l__unravel_head_token \__unravel_special_relax:
1280       \__unravel_get_next_notexpanded:
1281     }
1282     { \tl_clear:N \l__unravel_head_tl }
1283   }
1284 }
1285 \cs_new_protected:Npn \__unravel_get_next_aux:w
1286 { \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 cases where `\noexpand` acts on a control sequence or an active character. To make sure
`_unravel_notexpanded_test:w` 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
`_unravel_notexpanded_expand:NN` 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.

```

1287 \cs_set_protected:Npn \_unravel_tmp:w #1
1288   {
1289     \cs_new_protected:Npn \_unravel_get_next_notexpanded:
1290       {
1291         \tl_if_eq:onTF { \l_unravel_head_tl } { \_unravel_unravel_marker: }
1292         { \_unravel_get_next_marker: }
1293         {
1294           \_unravel_exp_args:NNx \use:nn \_unravel_notexpanded_test:w
1295           { \scan_stop: \exp_after:wN \cs_to_str:N \l_unravel_head_tl Z }
1296           \q_mark \_unravel_notexpanded_expand:n
1297           #1 Z \q_mark \use_none:n
1298           \q_stop
1299         }
1300       }
1301     \cs_new_protected:Npn \_unravel_notexpanded_test:w
1302       ##1 #1 ##2 Z \q_mark ##3##4 \q_stop
1303     { ##3 {##2} }
1304   }
1305 \exp_args:Nx \_unravel_tmp:w { \scan_stop: \tl_to_str:n { notexpanded: } }
1306 \group_begin:
1307   \char_set_catcode_active:n { 0 }
1308   \cs_new_protected:Npn \_unravel_notexpanded_expand:n #1
1309     {
1310       \_unravel_exp_args:Nx \tl_if_empty:nTF { \str_tail:n {#1} }
1311       {
1312         \group_begin:
1313         \char_set_lccode:nm { 0 } { '#1 }
1314         \tex_lowercase:D
1315         {
1316           \group_end:
1317           \_unravel_notexpanded_expand:N ^^@
1318         }
1319       }
1320     {
1321       \group_begin: \exp_args:NNc \group_end:
1322       \_unravel_notexpanded_expand:N { \use_none:n #1 }
1323     }
1324   }
1325 \group_end:
1326 \cs_new_protected:Npn \_unravel_notexpanded_expand:N #1

```

```

1327 {
1328   \gtl_set:Nn \l__unravel_head_gtl {#1}
1329   \tl_set:Nn \l__unravel_head_tl {#1}
1330   \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
1331 }

```

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

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

```

1332 \cs_new_protected:Npn \__unravel_get_next_marker:
1333 {
1334   \__unravel_get_next:
1335   \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
1336     { \__unravel_error:nxxxx { nested-unravel } { } { } { } { } }
1337     { \__unravel_error:nxxxx { internal } { marker~unknown } { } { } { } }
1338   \__unravel_input_gpop_item:NF \l__unravel_argi_tl
1339     { \__unravel_error:nxxxx { internal } { marker~1 } { } { } { } }
1340   \__unravel_input_gpop_item:NF \l__unravel_argii_tl
1341     { \__unravel_error:nxxxx { internal } { marker~2 } { } { } { } }
1342   \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
1343   \__unravel_exp_args:Nx \__unravel_back_input:n
1344     { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
1345   \__unravel_get_next:
1346 }

```

(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`.

```

1347 \cs_new_protected:Npn \__unravel_get_token:
1348 {
1349   \__unravel_get_next:
1350   \__unravel_set_cmd:
1351 }

```

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

```

1352 \cs_new_protected:Npn \__unravel_set_cmd:
1353 {
1354   \__unravel_set_cmd_aux_meaning:
1355   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1356     { }
1357     {
1358       \__unravel_token_if_expandable:NTF \l__unravel_head_token
1359         {
1360           \token_if_macro:NTF \l__unravel_head_token

```



```

1361         { \__unravel_set_cmd_aux_macro: }
1362         { \__unravel_set_cmd_aux_unknown: }
1363     }
1364     {
1365     \token_if_cs:NTF \l__unravel_head_token
1366     { \__unravel_set_cmd_aux_cs: }
1367     { \__unravel_set_cmd_aux_char: }
1368     }
1369 }
1370 }

```

(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.

```

1371 \cs_new_protected:Npn \__unravel_set_cmd_aux_meaning:
1372 {
1373     \tl_set:Nx \l__unravel_head_meaning_tl
1374     {
1375     \exp_after:wN \__unravel_strip_escape:w
1376     \token_to_meaning:N \l__unravel_head_token
1377     \tl_to_str:n { : }
1378     }
1379     \tl_set:Nx \l__unravel_head_meaning_tl
1380     {
1381     \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1382     \l__unravel_head_meaning_tl \q_stop
1383     }
1384 }
1385 \use:x
1386 {
1387     \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1388     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1389 }

```

(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.

```

1390 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
1391 {
1392     \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1393     {
1394     \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1395     { c__unravel_tex_#1_tl }
1396     #2
1397     }
1398 }
1399 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
1400 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
1401 {
1402     \int_set:Nn \l__unravel_head_cmd_int {#1}

```

```

1403     \int_set:Nn \l__unravel_head_char_int {#2}
1404   }

```

(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.

```

1405 \cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
1406   {
1407     \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1408     \int_zero:N \l__unravel_head_char_int
1409   }

```

(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`.

```

1410 \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1411   {
1412     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1413     \c__unravel_tex_relax_tl
1414     \__unravel_error:nxxxx { unknown-primitive }
1415     { \l__unravel_head_meaning_tl } { } { } { }
1416   }

```

(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.

```

1417 \cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
1418   {
1419     \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1420     { \tl_to_str:n { elect-font } }
1421     {
1422       \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1423       \c__unravel_tex_nullfont_tl
1424     }
1425     { \__unravel_set_cmd_aux_numeric: }
1426   }

```

(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.

```

1427 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:
1428   {

```

```

1429 \tl_set:Nx \l__unravel_tmpa_tl
1430 {
1431   \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1432   \l__unravel_head_meaning_tl + 0
1433 }
1434 \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1435 \l__unravel_tmpa_tl \q_stop
1436 }
1437 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1438 {
1439   \if_int_compare:w '#1 < 'A \exp_stop_f:
1440     \exp_not:N \q_mark
1441     \exp_after:wN \use_i:nn
1442   \fi:
1443   #1 \__unravel_set_cmd_aux_numeric:N
1444 }
1445 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1446 {
1447   \str_case:nnF {#1}
1448   {
1449     { char }      { \__unravel_set_cmd_aux_given:n { char_given } }
1450     { kchar }     { \__unravel_set_cmd_aux_given:n { char_given } }
1451     { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1452   }
1453   {
1454     \__unravel_set_cmd_aux_primitive:nTF {#1}
1455     { }
1456     { \__unravel_set_cmd_aux_unknown: }
1457     \int_add:Nn \l__unravel_head_char_int { 100 000 }
1458   }
1459   \int_add:Nn \l__unravel_head_char_int {#2}
1460 }
1461 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1462 {
1463   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1464   \int_zero:N \l__unravel_head_char_int
1465 }

```

(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 ‘.

```

1466 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:
1467 {
1468   \tl_set:Nx \l__unravel_head_meaning_tl
1469   { \token_to_meaning:N \l__unravel_head_token }
1470   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1471   { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1472   \exp_after:wN \__unravel_set_cmd_aux_char:w
1473   \l__unravel_head_meaning_tl \q_stop
1474   \__unravel_exp_args:NNx \int_set:Nn \l__unravel_head_char_int

```

```

1475     { ' \_unravel_token_to_char:N \l_unravel_head_token }
1476   }
1477   \cs_new_protected:Npn \_unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1478   {
1479     \int_set:Nn \l_unravel_head_cmd_int
1480     { \_unravel_tex_use:n { #1_char } }
1481   }

```

(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.

```

1482 \cs_new:Npn \_unravel_input_to_str:
1483 {
1484   \int_step_function:nnnN \g_unravel_input_int { -1 } { 1 }
1485   \_unravel_input_to_str_aux:n
1486 }
1487 \cs_new:Npn \_unravel_input_to_str_aux:n #1
1488 { \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.

```

1489 \cs_new_protected:Npn \_unravel_input_if_empty:TF
1490 {
1491   \int_compare:nNnTF \g_unravel_input_int = 0
1492   { \use_i:nn }
1493   {
1494     \gtl_if_empty:cTF
1495     { g_unravel_input_ \int_use:N \g_unravel_input_int _gtl }
1496     {
1497       \int_gdecr:N \g_unravel_input_int
1498       \_unravel_input_if_empty:TF
1499     }
1500     {
1501       \_unravel_input_split:
1502       \use_ii:nn
1503     }
1504   }
1505 }

```

(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

```

1506 \cs_new_protected:Npn \_unravel_input_split:
1507 {
1508   \int_compare:nNnT \g_unravel_input_int = 1
1509   {

```

```

1510         \exp_args:Nc \__unravel_input_split_aux:N
1511         { g__unravel_input_1_gtl }
1512     }
1513 }
1514 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1515 {
1516     \gtl_if_tl:NT #1
1517     {
1518         \gtl_if_head_is_N_type:NT #1
1519         {
1520             \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1521             \__unravel_exp_args:NNx \use:nn
1522             \__unravel_input_split_auxii:N
1523             { \tl_head:N \l__unravel_input_tmpa_tl }
1524         }
1525     }
1526 }
1527 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1528 {
1529     \token_if_parameter:NF #1
1530     {
1531         \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1532         { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1533         \group_begin:
1534         \cs_set:Npn \__unravel_input_split_auxiii:w
1535             ##1 \__unravel_input_split_end: { + 1 }
1536         \int_gset:Nn \g__unravel_input_int
1537             { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1538         \group_end:
1539         \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1540         \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1541     }
1542 }
1543 \cs_new:Npn \__unravel_input_split_end: { }
1544 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1545     #1 \__unravel_input_split_end:
1546 {
1547     \gtl_gclear_new:c
1548     { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1549     \gtl_gset:cn
1550     { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1551     \int_gdecr:N \g__unravel_input_tmpa_int
1552 }

```

(End definition for __unravel_input_split:.)

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

```

1553 \cs_new_protected:Npn \__unravel_input_gset:n
1554 {
1555     \int_gzero:N \g__unravel_input_int
1556     \__unravel_back_input:n
1557 }

```

(End definition for __unravel_input_gset:n.)

`_unravel_input_get:N`

```
1558 \cs_new_protected:Npn \_unravel_input_get:N #1
1559 {
1560   \_unravel_input_if_empty:TF
1561     { \gtl_set:Nn #1 { \q_no_value } }
1562     {
1563       \gtl_get_left:cN
1564       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1565     }
1566 }
```

(End definition for _unravel_input_get:N.)

`_unravel_input_get_left:N`

```
\_unravel_input_get_left_aux:nN
\_l_unravel_input_get_left_tl
1567 \tl_new:N \l__unravel_input_get_left_tl
1568 \cs_new_protected:Npn \_unravel_input_get_left:N #1
1569 {
1570   \tl_clear:N #1
1571   \exp_args:NV \_unravel_input_get_left_aux:nN \g__unravel_input_int #1
1572 }
1573 \cs_new_protected:Npn \_unravel_input_get_left_aux:nN #1#2
1574 {
1575   \int_compare:nNnF {#1} = 0
1576   {
1577     \tl_set:Nx \l__unravel_input_get_left_tl
1578     { \gtl_left_tl:c { g__unravel_input_#1_gtl } }
1579     \tl_concat:NNN #2 #2 \l__unravel_input_get_left_tl
1580     \gtl_if_tl:cT { g__unravel_input_#1_gtl }
1581     {
1582       \exp_args:Nf \_unravel_input_get_left_aux:nN
1583       { \int_eval:n { #1 - 1 } } #2
1584     }
1585   }
1586 }
```

(End definition for _unravel_input_get_left:N, _unravel_input_get_left_aux:nN, and \l__unravel_input_get_left_tl.)

`_unravel_input_gpop:N`

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

```
1587 \cs_new_protected:Npn \_unravel_input_gpop:N #1
1588 {
1589   \_unravel_input_if_empty:TF
1590     { \gtl_set:Nn #1 { \q_no_value } }
1591     {
1592       \gtl_gpop_left:cN
1593       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1594     }
1595 }
```

(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.

```

1596 \cs_new_protected:Npn \__unravel_input_merge:
1597 {
1598   \int_gdecr:N \g__unravel_input_int
1599   \gtl_gconcat:ccc
1600   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1601   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1602   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1603   \gtl_gclear:c
1604   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1605 }

```

(End definition for __unravel_input_merge:.)

__unravel_input_gpop_item:NTF 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

```

1606 \prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
1607 {
1608   \int_compare:nNnTF \g__unravel_input_int = 0
1609   { \prg_return_false: }
1610   {
1611     \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1612     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1613   }
1614 }
1615 \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
1616 {
1617   \gtl_gpop_left_item:NNTF #1#2
1618   { \prg_return_true: }
1619   {
1620     \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
1621     { \prg_return_false: }
1622     {
1623       \int_compare:nNnTF \g__unravel_input_int = 1
1624       { \prg_return_false: }
1625       {
1626         \__unravel_input_merge:
1627         \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1628         {
1629           g__unravel_input_
1630           \int_use:N \g__unravel_input_int _gtl
1631         }
1632         #2
1633       }
1634     }
1635   }
1636 }

```

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

__unravel_input_gpop_tl:N

```

1637 \cs_new_protected:Npn \__unravel_input_gpop_tl:N #1

```

```

1638 { \ttl_clear:N #1 \__unravel_input_gpop_tl_aux:N #1 }
1639 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:N #1
1640 {
1641   \int_compare:nNnF \g__unravel_input_int = 0
1642   {
1643     \exp_args:Nc \__unravel_input_gpop_tl_aux:NN
1644     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1645   }
1646 }
1647 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
1648 {
1649   \gtl_if_tl:NTF #1
1650   {
1651     \ttl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1652     \gtl_gclear:N #1
1653     \int_gdecr:N \g__unravel_input_int
1654     \__unravel_input_gpop_tl_aux:N #2
1655   }
1656   {
1657     \int_compare:nNnTF \g__unravel_input_int > 1
1658     { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
1659     { \use_i:nn }
1660     {
1661       \ttl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1662       \gtl_gpop_left_tl:N #1
1663     }
1664     {
1665       \__unravel_input_merge:
1666       \__unravel_input_gpop_tl_aux:N #2
1667     }
1668   }
1669 }

```

(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.

```

1670 \cs_new_protected:Npn \__unravel_back_input:n
1671 {
1672   \int_gincr:N \g__unravel_input_int
1673   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1674   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1675 }
1676 \cs_generate_variant:Nn \__unravel_back_input:n { V , o }
1677 \cs_new_protected:Npn \__unravel_back_input:x
1678 { \__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.

```

1679 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1680 {
1681   \gtl_if_tl:NTF #1

```



```

1682     { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1683     {
1684         \gtl_gconcat:cNc
1685         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1686         #1
1687         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1688     }
1689 }

```

(End definition for __unravel_back_input_gtl:N.)

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

```

1690 \cs_new_protected:Npn \__unravel_back_input:
1691 { \__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.

```

1692 \cs_new_protected:Npn \__unravel_back_input_tl_o:
1693 {
1694     \tl_set:Nx \l__unravel_tmpa_tl
1695     { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1696     \__unravel_back_input:V \l__unravel_tmpa_tl
1697     \__unravel_print_expansion:x
1698     { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1699 }

```

(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.

```

1700 \cs_new_protected:Npn \__unravel_insert_relax:
1701 {
1702     \__unravel_back_input:
1703     \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1704     \__unravel_back_input:
1705     \__unravel_print_action:
1706 }

```

(End definition for __unravel_insert_relax:.)

__unravel_insert_group_begin_error:

```

1707 \cs_new_protected:Npn \__unravel_insert_group_begin_error:
1708 {
1709     \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
1710     \__unravel_back_input:
1711     \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1712     \__unravel_back_input:
1713     \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
1714     \__unravel_print_action:
1715 }

```

(End definition for `_unravel_insert_group_begin_error:`.)

`_unravel_insert_dollar_error:`

```

1716 \cs_new_protected:Npn \_unravel_insert_dollar_error:
1717 {
1718   \_unravel_back_input:
1719   \_unravel_back_input:n { $ } % $
1720   \_unravel_error:nnnnn { missing-dollar } { } { } { } { }
1721   \_unravel_print_action:
1722 }

```

(End definition for `_unravel_insert_dollar_error:`.)

2.5.3 Macro calls

`_unravel_macro_prefix:N`

`_unravel_macro_parameter:N`

`_unravel_macro_replacement:N`

```

1723 \use:x
1724 {
1725   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:NN #1 }
1726   {
1727     \exp_not:n { \exp_after:wN \_unravel_macro_split_do:wN }
1728     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1729     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1730     \exp_not:N \q_stop
1731   }
1732   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:wN }
1733   \exp_not:n { #1 } \tl_to_str:n { : } \exp_not:n { #2 -> }
1734   \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1735   { \exp_not:n { #4 #6 { #1 } { #2 } { #3 } } }
1736 }
1737 \cs_new:Npn \_unravel_macro_prefix:N #1
1738 { \_unravel_macro_split_do:NN #1 \use_i:nnn }
1739 \cs_new:Npn \_unravel_macro_parameter:N #1
1740 { \_unravel_macro_split_do:NN #1 \use_ii:nnn }
1741 \cs_new:Npn \_unravel_macro_replacement:N #1
1742 { \_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 we can be safe by extracting the longest balanced prefix in the input and working with that.

`_unravel_macro_call_safe:`

`_unravel_macro_call_quick:`

`_unravel_macro_call_quick_loop:NNN`

`_unravel_macro_call_quick_runaway:Nw`

```

1743 \cs_new_protected:Npn \_unravel_macro_call:
1744 {
1745   \bool_if:NTF \g_unravel_speedup_macros_bool
1746   {
1747     \tl_set:Nx \l__unravel_tmpa_tl
1748     { ^ \exp_after:wN \_unravel_macro_parameter:N \l__unravel_head_tl }
1749     \_unravel_tl_if_in:ooTF \c_unravel_parameters_tl \l__unravel_tmpa_tl
1750     { \_unravel_macro_call_quick: } { \_unravel_macro_call_safe: }
1751   }
1752   { \_unravel_macro_call_safe: }

```

```

1753     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1754     \__unravel_print_expansion:
1755   }
1756 \cs_new_protected:Npn \__unravel_macro_call_safe:
1757 {
1758   \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1759   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1760 }
1761 \cs_new_protected:Npn \__unravel_macro_call_quick:
1762 {
1763   \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1764   { ? \use_none_delimit_by_q_stop:w } \q_stop
1765 }
1766 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1767 {
1768   \use_none:n #2
1769   \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1770   { \__unravel_macro_call_quick_runaway:Nw #3 }
1771   \tl_put_right:Nx \l__unravel_head_tl
1772   { { \exp_not:V \l__unravel_tmpa_tl } }
1773   \__unravel_macro_call_quick_loop:NNN
1774   #3
1775 }
1776 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1777 {
1778   \__unravel_error:nxxxx { runaway-macro-parameter }
1779   { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1780 }

```

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

```

1781 \cs_new_protected:Npn \__unravel_expand_do:N
1782 {
1783   \__unravel_set_action_text:
1784   \bool_if:NT \g__unravel_internal_debug_bool
1785   {

```

```

1786     \__unravel_set_cmd:
1787     \__unravel_exp_args:Nx \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_in
1788     }
1789     \token_if_macro:NTF \l__unravel_head_token
1790     { \__unravel_macro_call: }
1791     { \__unravel_expand_nonmacro: }
1792     }

```

(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.

```

1793 \cs_new_protected:Npn \__unravel_expand_nonmacro:
1794 {
1795   \__unravel_set_cmd_aux_meaning:
1796   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1797   {
1798     \cs_if_exist_use:cF
1799     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1800     { \__unravel_error:nxxxx { internal } { expandable } { } { } { } }
1801   }
1802   {
1803     \__unravel_error:nxxxx { unknown-primitive }
1804     { \l__unravel_head_meaning_tl } { } { } { }
1805     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1806     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1807     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1808     \__unravel_print_expansion:
1809   }
1810 }

```

(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:.

```

1811 \cs_new_protected:Npn \__unravel_get_x_next:
1812 {
1813   \__unravel_get_next:
1814   \__unravel_token_if_expandable:NT \l__unravel_head_token
1815   { \__unravel_expand_do:N \__unravel_get_x_next: }
1816 }

```

(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.

```

1817 \cs_new_protected:Npn \__unravel_get_x_or_protected:
1818 {

```

```

1819     \__unravel_get_next:
1820     \__unravel_token_if_protected:NF \l__unravel_head_token
1821         { \__unravel_expand_do:N \__unravel_get_x_or_protected: }
1822     }

```

(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).

```

1823 \cs_new_protected:Npn \__unravel_get_token_xdef:
1824 {
1825     \__unravel_get_next:
1826     \__unravel_token_if_protected:NF \l__unravel_head_token
1827         { \__unravel_expand_do:N \__unravel_get_token_xdef: }
1828 }
1829 \cs_new_protected:Npn \__unravel_get_token_x:
1830 {
1831     \__unravel_get_next:
1832     \__unravel_token_if_protected:NF \l__unravel_head_token
1833         { \__unravel_expand_do:N \__unravel_get_token_x: }
1834 }

```

(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.

```

1835 \cs_new_protected:Npn \__unravel_get_x_non_blank:
1836 {
1837     \__unravel_get_x_next:
1838     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1839         { \__unravel_get_x_non_blank: }
1840 }

```

(End definition for __unravel_get_x_non_blank:.)

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

```

1841 \cs_new_protected:Npn \__unravel_get_x_non_relax:
1842 {
1843     \__unravel_get_x_next:
1844     \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
1845         { \__unravel_get_x_non_relax: }
1846     {
1847         \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_special_relax:
1848             { \__unravel_get_x_non_relax: }
1849         {
1850             \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1851                 { \__unravel_get_x_non_relax: }
1852         }
1853     }
1854 }

```

(End definition for __unravel_get_x_non_relax:.)

`__unravel_skip_optional_space:`

```
1855 \cs_new_protected:Npn \__unravel_skip_optional_space:
1856 {
1857   \__unravel_get_x_next:
1858   \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1859   { \__unravel_back_input: }
1860 }
```

(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.

```
1861 \cs_new_protected:Npn \__unravel_scan_optional_equals:
1862 {
1863   \__unravel_get_x_non_blank:
1864   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1865   { \__unravel_prev_input:n { = } }
1866   {
1867     \__unravel_prev_input_silent:n { = }
1868     \__unravel_back_input:
1869   }
1870 }
```

(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:`.

```
1871 \cs_new_protected:Npn \__unravel_scan_left_brace:
1872 {
1873   \__unravel_get_x_non_relax:
1874   \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
1875   {
1876     \__unravel_insert_group_begin_error:
1877     \__unravel_get_next:
1878   }
1879 }
```

(End definition for __unravel_scan_left_brace:.)

`__unravel_scan_keyword:n`
`__unravel_scan_keyword:nTF`
`__unravel_scan_keyword_loop:NNN`
`__unravel_scan_keyword_test:NNTF`
`__unravel_scan_keyword_true:`
`__unravel_scan_keyword_false:w`

The details of how TeX 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 `TEX`'s skipping of leading spaces here must be intertwined with the search for keyword, as is shown by the (plain `TEX`) example

```

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

1880 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1881   { \__unravel_scan_keyword:nTF {#1} { } { } }
1882 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1883   { T , F , TF }
1884   {
1885     \__unravel_prev_input_gpush_gtl:
1886     \__unravel_scan_keyword_loop:NNN \c_true_bool
1887     #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1888   }
1889 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1890   {
1891     \quark_if_recursion_tail_stop_do:nn {#2}
1892     { \__unravel_scan_keyword_true: }
1893     \quark_if_recursion_tail_stop_do:nn {#3}
1894     { \__unravel_error:nxxxx { internal } { odd-keyword-length } { } { } { } }
1895     \__unravel_get_x_next:
1896     \__unravel_scan_keyword_test:NNTF #2#3
1897     {
1898       \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1899       \__unravel_scan_keyword_loop:NNN \c_false_bool
1900     }
1901     {
1902       \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1903       { \__unravel_scan_keyword_false:w }
1904       \bool_if:NF #1
1905       { \__unravel_scan_keyword_false:w }
1906       \__unravel_scan_keyword_loop:NNN #1#2#3
1907     }
1908   }
1909 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1910   { TF }
1911   {
1912     \__unravel_gtl_if_head_is_definable:NNTF \l__unravel_head_gtl
1913     { \prg_return_false: }
1914     {
1915       \str_if_eq:eeTF
1916       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1917       { \prg_return_true: }
1918       {
1919         \str_if_eq:eeTF
1920         { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}

```

```

1921         { \prg_return_true: }
1922         { \prg_return_false: }
1923     }
1924 }
1925 }
1926 \cs_new_protected:Npn \__unravel_scan_keyword_true:
1927 {
1928     \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1929     \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1930     \prg_return_true:
1931 }
1932 \cs_new_protected:Npn \__unravel_scan_keyword_false:w
1933 #1 \q_recursion_stop
1934 {
1935     \__unravel_back_input:
1936     \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1937     \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1938     \prg_return_false:
1939 }

```

(End definition for `__unravel_scan_keyword:n` and others.)

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

```

1940 \cs_new_protected:Npn \__unravel_scan_to:
1941 {
1942     \__unravel_scan_keyword:nF { tTo0 }
1943     {
1944         \__unravel_error:nnnnn { missing-to } { } { } { } { }
1945         \__unravel_prev_input:n { to }
1946     }
1947 }

```

(End definition for `__unravel_scan_to:.`)

`__unravel_scan_font_ident:` Find a font identifier.

```

1948 \cs_new_protected:Npn \__unravel_scan_font_ident:
1949 {
1950     \__unravel_get_x_non_blank:
1951     \__unravel_set_cmd:
1952     \int_case:nnF \l__unravel_head_cmd_int
1953     {
1954         { \__unravel_tex_use:n { def_font } }
1955         { \__unravel_prev_input:V \l__unravel_head_tl }
1956         { \__unravel_tex_use:n { letterspace_font } }
1957         { \__unravel_prev_input:V \l__unravel_head_tl }
1958         { \__unravel_tex_use:n { pdf_copy_font } }
1959         { \__unravel_prev_input:V \l__unravel_head_tl }
1960         { \__unravel_tex_use:n { set_font } }
1961         { \__unravel_prev_input:V \l__unravel_head_tl }
1962         { \__unravel_tex_use:n { def_family } }
1963         {
1964             \__unravel_prev_input:V \l__unravel_head_tl
1965             \__unravel_scan_int:
1966         }
1967     }
}

```



```

1968     {
1969     \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
1970     \__unravel_back_input:
1971     \__unravel_prev_input:n { \__unravel_nullfont: }
1972     }
1973 }

```

(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).

```

1974 \cs_new_protected:Npn \__unravel_scan_font_int:
1975 {
1976   \int_case:nnF \l__unravel_head_char_int
1977   {
1978     { 0 } { \__unravel_scan_font_ident: }
1979     { 1 } { \__unravel_scan_font_ident: }
1980     { 6 } { \__unravel_scan_font_ident: }
1981   }
1982   { \__unravel_scan_font_ident: \__unravel_scan_int: }
1983 }

```

(End definition for __unravel_scan_font_int:.)

__unravel_scan_font_dimen: Find operands for \fontdimen.

```

1984 \cs_new_protected:Npn \__unravel_scan_font_dimen:
1985 {
1986   \__unravel_scan_int:
1987   \__unravel_scan_font_ident:
1988 }

```

(End definition for __unravel_scan_font_dimen:.)

__unravel_scan_something_internal:n Receives an (explicit) “level” argument:

__unravel_scan_something_aux:nwn

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

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

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

If the argument (#3 in the auxiliary) is < 4 but the level that will be produced (#1 in the auxiliary) is ≥ 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).

```

1989 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1990 {
1991   \__unravel_set_cmd:
1992   \__unravel_set_action_text:
1993   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1994   \exp_after:wN \__unravel_scan_something_aux:nwn
1995     \l__unravel_tmpa_tl \q_stop {#1}
1996 }
1997 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1998 {
1999   \int_compare:nT { #3 < 4 <= #1 }
2000   {
2001     \__unravel_back_input:
2002     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2003     \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
2004     \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
2005     \__unravel_break:w
2006   }
2007   \int_compare:nNnT {#1} = { 8 }
2008   {
2009     \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
2010     \__unravel_scan_something_internal_auxii:nn 0 { 0 }
2011     \__unravel_break:w
2012   }
2013   \tl_if_empty:nF {#2}
2014   {
2015     \__unravel_prev_input_gpush:N \l__unravel_head_tl
2016     \__unravel_print_action:
2017     #2
2018     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2019   }
2020   \int_compare:nNnTF {#3} < { 4 }
2021     { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
2022     { \tl_set:Nx \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
2023   \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
2024   \__unravel_break_point:
2025   \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
2026 }
2027 \cs_new_protected:Npn \__unravel_scan_something_internal_auxii:nn #1#2
2028 {

```

```

2029     \_unravel_prev_input_silent:n {#1}
2030     \_unravel_set_action_text:
2031     \_unravel_set_action_text:x
2032     { \g_unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
2033     \int_gset:Nn \g_unravel_val_level_int {#2}
2034   }
2035 \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), followed by some code to fetch necessary operands. In most cases, this can be done by 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.

```

2036 \cs_new:Npn \_unravel_thing_case:
2037   {
2038     \int_case:nnF \l_unravel_head_cmd_int
2039     {
2040       { 68 } { 0 } % char_given
2041       { 69 } { 0 } % math_given
2042       { 70 } { \_unravel_thing_last_item: } % last_item
2043       { 71 } { 5 \_unravel_scan_toks_register: } % toks_register
2044       { 72 } { 5 } % assign_toks
2045       { 73 } { 0 } % assign_int
2046       { 74 } { 1 } % assign_dimen
2047       { 75 } { 2 } % assign_glue
2048       { 76 } { 3 } % assign_mu_glue
2049       { 77 } { 1 \_unravel_scan_font_dimen: } % assign_font_dimen
2050       { 78 } { 0 \_unravel_scan_font_int: } % assign_font_int
2051       { 79 } { \_unravel_thing_set_aux: } % set_aux
2052       { 80 } { 0 } % set_prev_graf
2053       { 81 } { 1 } % set_page_dimen
2054       { 82 } { 0 } % set_page_int
2055       { 83 } { 1 \_unravel_scan_int: } % set_box_dimen
2056       { 84 } { 0 \_unravel_scan_int: } % set_shape
2057       { 85 } { 0 \_unravel_scan_int: } % def_code
2058       { 86 } { 4 \_unravel_scan_int: } % def_family
2059       { 87 } { 4 } % set_font
2060       { 88 } { 4 } % def_font
2061       { 89 } { \_unravel_thing_register: } % register
2062       {101 } { 4 } % letterspace_font
2063       {102 } { 4 } % pdf_copy_font
2064     }
2065     { 8 }
2066   }
2067 \cs_new:Npn \_unravel_thing_set_aux:
2068   { \int_compare:nNnTF \l_unravel_head_char_int = { 1 } { 1 } { 0 } }
2069 \cs_new:Npn \_unravel_thing_last_item:
2070   {
2071     \int_compare:nNnTF \l_unravel_head_char_int < { 26 }
2072     {
2073       \int_case:nnF \l_unravel_head_char_int

```

```

2074     {
2075         { 1 } { 1 } % lastkern
2076         { 2 } { 2 } % lastskip
2077     }
2078     { 0 } % other integer parameters
2079 }
2080 {
2081     \int_case:nnF \l__unravel_head_char_int
2082     {
2083         { 26 } { 0 \__unravel_scan_normal_glue: } % gluestretchorder
2084         { 27 } { 0 \__unravel_scan_normal_glue: } % glueshrinkorder
2085         { 28 } % fontcharwd
2086         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2087         { 29 } % fontcharht
2088         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2089         { 30 } % fontchardp
2090         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2091         { 31 } % fontcharic
2092         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2093         { 32 } { 1 \__unravel_scan_int: } % parshapelength
2094         { 33 } { 1 \__unravel_scan_int: } % parshapeindent
2095         { 34 } { 1 \__unravel_scan_int: } % parshapedimen
2096         { 35 } { 1 \__unravel_scan_normal_glue: } % gluestretch
2097         { 36 } { 1 \__unravel_scan_normal_glue: } % glueshrink
2098         { 37 } { 2 \__unravel_scan_mu_glue: } % mutoglu
2099         { 38 } { 3 \__unravel_scan_normal_glue: } % gluetomu
2100         { 39 } % numexpr
2101         { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
2102         { 40 } % dimexpr
2103         { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
2104         { 41 } % glueexpr
2105         { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
2106         { 42 } % muexpr
2107         { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
2108     }
2109     { }
2110 }
2111 }
2112 \cs_new:Npn \__unravel_thing_register:
2113 {
2114     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
2115     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
2116     { \__unravel_scan_int: }
2117 }

```

(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.

```

2118 \cs_new_protected:Npn \__unravel_scan_toks_register:
2119 {
2120     \int_compare:nNnT \l__unravel_head_char_int = 0
2121     { \__unravel_scan_int: }
2122 }

```

(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.

```

2123 \cs_new_protected:Npn \_unravel_thing_use_get:nnNN #1#2#3#4
2124 {
2125   \int_compare:nNnTF {#2} < { 3 }
2126   {
2127     \int_compare:nNnT {#1} = { 3 }
2128     { \_unravel_tex_error:nV { incompatible-units } #3 }
2129     \tl_set:Nx #4
2130     {
2131       \int_case:nn { \int_min:nn {#1} {#2} }
2132       {
2133         { 0 } \int_eval:n
2134         { 1 } \dim_eval:n
2135         { 2 } \skip_eval:n
2136       }
2137       { \int_compare:nNnT {#1} = { 3 } \tex_mutoglu:D #3 }
2138     }
2139   }
2140   {
2141     \int_case:nnF {#1}
2142     {
2143       { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
2144       { 3 } { \tl_set:Nx #4 { \muskip_eval:n {#3} } }
2145     }
2146     {
2147       \_unravel_tex_error:nV { incompatible-units } #3
2148       \tl_set:Nx #4 { \muskip_eval:n { \tex_gluetomu:D #3 } }
2149     }
2150   }
2151 }

```

(End definition for `_unravel_thing_use_get:nnNN`.)

```

\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
2152 \cs_new_protected:Npn \_unravel_scan_expr:N #1
2153 { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
2154 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
2155 {
2156   \_unravel_get_x_non_blank:
2157   \_unravel_scan_factor:N #1
2158   \_unravel_scan_expr_op:NN #1#2
2159 }
2160 \cs_new_protected:Npn \_unravel_scan_expr_op:NN #1#2
2161 {
2162   \_unravel_get_x_non_blank:
2163   \tl_case:NnF \l__unravel_head_tl
2164   {
2165     \c__unravel_plus_tl
2166     {
2167       \_unravel_prev_input:V \l__unravel_head_tl
2168       \_unravel_scan_expr_aux:NN #1#2

```

```

2169     }
2170     \c__unravel_minus_tl
2171     {
2172         \__unravel_prev_input:V \l__unravel_head_tl
2173         \__unravel_scan_expr_aux:NN #1#2
2174     }
2175     \c__unravel_times_tl
2176     {
2177         \__unravel_prev_input:V \l__unravel_head_tl
2178         \__unravel_get_x_non_blank:
2179         \__unravel_scan_factor:N \__unravel_scan_int:
2180         \__unravel_scan_expr_op:NN #1#2
2181     }
2182     \c__unravel_over_tl
2183     {
2184         \__unravel_prev_input:V \l__unravel_head_tl
2185         \__unravel_get_x_non_blank:
2186         \__unravel_scan_factor:N \__unravel_scan_int:
2187         \__unravel_scan_expr_op:NN #1#2
2188     }
2189     \c__unravel_rp_tl
2190     {
2191         \bool_if:NTF #2
2192         { \__unravel_prev_input:V \l__unravel_head_tl }
2193         { \__unravel_back_input: }
2194     }
2195 }
2196 {
2197     \bool_if:NTF #2
2198     {
2199         \__unravel_error:nnnnn { missing-rparen } { } { } { } { }
2200         \__unravel_back_input:
2201         \__unravel_prev_input:V \c__unravel_rp_tl
2202     }
2203     {
2204         \token_if_eq_meaning:NMF \l__unravel_head_token \scan_stop:
2205         { \__unravel_back_input: }
2206     }
2207 }
2208 }
2209 \cs_new_protected:Npn \__unravel_scan_factor:N #1
2210 {
2211     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2212     {
2213         \__unravel_prev_input:V \l__unravel_head_tl
2214         \__unravel_scan_expr_aux:NN #1 \c_true_bool
2215     }
2216     {
2217         \__unravel_back_input:
2218         #1
2219     }
2220 }

```

(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`.

```

2221 \cs_new_protected:Npn \_unravel_scan_signs:
2222 {
2223   \_unravel_get_x_non_blank:
2224   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
2225     {
2226       \_unravel_prev_input:V \l__unravel_head_tl
2227       \_unravel_scan_signs:
2228     }
2229   {
2230     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
2231     {
2232       \_unravel_prev_input:V \l__unravel_head_tl
2233       \_unravel_scan_signs:
2234     }
2235   }
2236 }

```

(End definition for _unravel_scan_signs:.)

```

\_unravel_scan_int:
\_unravel_scan_int_char:
\_unravel_scan_int_lq:
\_unravel_scan_int_explicit:n
2237 \cs_new_protected:Npn \_unravel_scan_int:
2238 {
2239   \_unravel_scan_signs:
2240   \_unravel_set_cmd:
2241   \_unravel_cmd_if_internal:TF
2242     { \_unravel_scan_something_internal:n { 0 } }
2243     { \_unravel_scan_int_char: }
2244 }
2245 \cs_new_protected:Npn \_unravel_scan_int_char:
2246 {
2247   \tl_case:NnF \l__unravel_head_tl
2248     {
2249       \c__unravel_lq_tl { \_unravel_scan_int_lq: }
2250       \c__unravel_rq_tl
2251       {
2252         \_unravel_prev_input:V \l__unravel_head_tl
2253         \_unravel_get_x_next:
2254         \_unravel_scan_int_explicit:Nn \c_false_bool { ' }
2255       }
2256       \c__unravel_dq_tl
2257       {
2258         \_unravel_prev_input:V \l__unravel_head_tl
2259         \_unravel_get_x_next:
2260         \_unravel_scan_int_explicit:Nn \c_false_bool { " }
2261       }
2262     }
2263     { \_unravel_scan_int_explicit:Nn \c_false_bool { } }
2264 }
2265 \cs_new_protected:Npn \_unravel_scan_int_lq:
2266 {
2267   \_unravel_get_next:
2268   \_unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl

```

```

2269     {
2270     \tl_set:Nx \l__unravel_head_tl
2271     { \__unravel_token_to_char:N \l__unravel_head_token }
2272     }
2273 \tl_set:Nx \l__unravel_tmpa_tl
2274 { \int_eval:n { \exp_after:wN ‘ \l__unravel_head_tl } }
2275 \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2276 \__unravel_print_action:x
2277 { ‘ \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2278 \__unravel_skip_optional_space:
2279 }
2280 \cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
2281 {
2282 \if_int_compare:w 1
2283 < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2284 \exp_after:wN \use_i:nn
2285 \else:
2286 \exp_after:wN \use_ii:nn
2287 \fi:
2288 {
2289 \__unravel_prev_input:V \l__unravel_head_tl
2290 \__unravel_get_x_next:
2291 \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
2292 }
2293 {
2294 \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2295 { \__unravel_back_input: }
2296 \bool_if:NF #1
2297 {
2298 \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2299 \__unravel_prev_input:n { 0 }
2300 }
2301 }
2302 }

```

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

`__unravel_scan_normal_dimen:`

```

2303 \cs_new_protected:Npn \__unravel_scan_normal_dimen:
2304 { \__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`.

```

2305 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2306 {
2307 \__unravel_scan_signs:
2308 \__unravel_prev_input_gpush:
2309 \__unravel_set_cmd:

```



```

2310 \__unravel_cmd_if_internal:TF
2311 {
2312   \int_compare:nNnTF {#1} = { 3 }
2313     { \__unravel_scan_something_internal:n { 3 } }
2314     { \__unravel_scan_something_internal:n { 1 } }
2315   \int_compare:nNnT \g__unravel_val_level_int = { 0 }
2316     { \__unravel_scan_dim_unit:nN {#1} #2 }
2317 }
2318 { \__unravel_scan_dimen_char:nN {#1} #2 }
2319 \__unravel_prev_input_gpop:N \l__unravel_head_tl
2320 \__unravel_prev_input_silent:V \l__unravel_head_tl
2321 }
2322 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
2323 {
2324   \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2325     { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2326   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
2327     {
2328     \__unravel_prev_input:n { . }
2329     \__unravel_scan_decimal_loop:
2330     }
2331   {
2332     \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl
2333     {
2334       \__unravel_back_input:
2335       \__unravel_scan_int:
2336       \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2337         { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2338       \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
2339         {
2340           \__unravel_input_gpop:N \l__unravel_tmpb_gtl
2341           \__unravel_prev_input:n { . }
2342           \__unravel_scan_decimal_loop:
2343         }
2344     }
2345   {
2346     \__unravel_back_input:
2347     \__unravel_scan_int:
2348   }
2349 }
2350 \__unravel_scan_dim_unit:nN {#1} #2
2351 }
2352 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
2353 {
2354   \bool_if:NT #2
2355     {
2356       \__unravel_scan_keyword:nT { fFiIlL }
2357       {
2358         \__unravel_scan_inf_unit_loop:
2359         \__unravel_break:w
2360       }
2361     }
2362   \__unravel_get_x_non_blank:
2363   \__unravel_set_cmd:

```

```

2364 \__unravel_cmd_if_internal:TF
2365 {
2366   \__unravel_prev_input_gpush:
2367   \__unravel_scan_something_internal:n {#1}
2368   \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2369   \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2370   \__unravel_break:w
2371 }
2372 { \__unravel_back_input: }
2373 \int_compare:nNnT {#1} = { 3 }
2374 {
2375   \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2376   \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
2377   \__unravel_prev_input:n { mu }
2378   \__unravel_break:w
2379 }
2380 \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2381 \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2382 \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2383 \__unravel_scan_keyword:nT { tTrRuUeE }
2384 { \__unravel_prepare_mag: }
2385 \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2386 \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2387 \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2388 \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }
2389 \__unravel_scan_keyword:nT { mMmM } { \__unravel_break:w }
2390 \__unravel_scan_keyword:nT { bBpP } { \__unravel_break:w }
2391 \__unravel_scan_keyword:nT { dDdD } { \__unravel_break:w }
2392 \__unravel_scan_keyword:nT { cCcC } { \__unravel_break:w }
2393 \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
2394 \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
2395 \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
2396 \__unravel_tex_error:nV { missing-pt } \l__unravel_head_tl
2397 \__unravel_prev_input:n { pt }
2398 \__unravel_break_point:
2399 }
2400 \cs_new_protected:Npn \__unravel_scan_inf_unit_loop:
2401 { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
2402 \cs_new_protected:Npn \__unravel_scan_decimal_loop:
2403 {
2404   \__unravel_get_x_next:
2405   \tl_if_empty:NTF \l__unravel_head_tl
2406   { \use_ii:nn }
2407   { \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl }
2408   {
2409     \__unravel_prev_input:V \l__unravel_head_tl
2410     \__unravel_scan_decimal_loop:
2411   }
2412   {
2413     \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
2414     { \__unravel_back_input: }
2415     \__unravel_prev_input_silent:n { ~ }
2416   }
2417 }

```

(End definition for _unravel_scan_dimen:nN.)

_unravel_scan_normal_glue:
_unravel_scan_mu_glue:

```
2418 \cs_new_protected:Npn \_unravel_scan_normal_glue:
2419   { \_unravel_scan_glue:n { 2 } }
2420 \cs_new_protected:Npn \_unravel_scan_mu_glue:
2421   { \_unravel_scan_glue:n { 3 } }
```

(End definition for _unravel_scan_normal_glue: and _unravel_scan_mu_glue:.)

_unravel_scan_glue:n

```
2422 \cs_new_protected:Npn \_unravel_scan_glue:n #1
2423   {
2424     \_unravel_prev_input_gpush:
2425     \_unravel_scan_signs:
2426     \_unravel_prev_input_gpush:
2427     \_unravel_set_cmd:
2428     \_unravel_cmd_if_internal:TF
2429     {
2430       \_unravel_scan_something_internal:n {#1}
2431       \int_case:nnF \g__unravel_val_level_int
2432       {
2433         { 0 } { \_unravel_scan_dim_unit:nN {#1} \c_false_bool }
2434         { 1 } { }
2435       }
2436       { \_unravel_break:w }
2437     }
2438     { \_unravel_back_input: \_unravel_scan_dimen:nN {#1} \c_false_bool }
2439     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2440     \_unravel_prev_input_gpush:
2441     \_unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2442     \_unravel_scan_keyword:nT { pPlLuUsS }
2443     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2444     \_unravel_scan_keyword:nT { mMiInNuUsS }
2445     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2446     \_unravel_break_point:
2447     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2448     \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2449   }
```

(End definition for _unravel_scan_glue:n.)

_unravel_scan_file_name:

```
2450 \cs_new_protected:Npn \_unravel_scan_file_name:
2451   {
2452     \bool_gset_true:N \g__unravel_name_in_progress_bool
2453     \_unravel_get_x_non_blank:
2454     \_unravel_scan_file_name_loop:
2455     \bool_gset_false:N \g__unravel_name_in_progress_bool
2456     \_unravel_prev_input_silent:n { ~ }
2457   }
2458 \cs_new_protected:Npn \_unravel_scan_file_name_loop:
2459   {
2460     \_unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
```

```

2461     { \__unravel_back_input: }
2462     {
2463       \tl_set:Nx \l__unravel_tmpa_tl
2464       { \__unravel_token_to_char:N \l__unravel_head_token }
2465       \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2466       {
2467         \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2468         \__unravel_get_x_next:
2469         \__unravel_scan_file_name_loop:
2470       }
2471     }
2472   }

```

(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.

```

2473 \cs_new_protected:Npn \__unravel_scan_r_token:
2474 {
2475   \bool_do_while:nn
2476   { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2477   { \__unravel_get_next: }
2478   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2479   {
2480     \__unravel_error:nnnnn { missing-cs } { } { } { } { }
2481     \__unravel_back_input:
2482     \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2483   }
2484   \__unravel_prev_input_silent:V \l__unravel_head_tl
2485   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2486 }

```

(End definition for __unravel_scan_r_token:.)

__unravel_scan_toks_to_str:

```

2487 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2488 {
2489   \__unravel_prev_input_gpush:
2490   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2491   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2492   \__unravel_prev_input_silent:x
2493   { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2494 }

```

(End definition for __unravel_scan_toks_to_str:.)

__unravel_scan_pdf_ext_toks:

```

2495 \cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
2496 {
2497   \__unravel_prev_input_gpush:
2498   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2499   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2500   \__unravel_prev_input_silent:x
2501   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2502 }

```

(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`).

```
2503 \cs_new_protected:Npn \_unravel_scan_toks:NN #1#2
2504 {
2505   \bool_if:NT #1 { \_unravel_scan_param: }
2506   \_unravel_scan_left_brace:
2507   \bool_if:NTF #2
2508     { \_unravel_scan_group_x:N #1 }
2509     { \_unravel_scan_group_n:N #1 }
2510 }
```

(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`.

`_unravel_scan_param_aux:`

```
2511 \cs_new_protected:Npn \_unravel_scan_param:
2512 {
2513   \tl_clear:N \l__unravel_tmpa_tl
2514   \_unravel_scan_param_aux:
2515   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2516   \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2517 }
2518 \cs_new_protected:Npn \_unravel_scan_param_aux:
2519 {
2520   \_unravel_get_next:
2521   \tl_concat:NNN \l__unravel_tmpa_tl
2522     \l__unravel_tmpa_tl \l__unravel_head_tl
2523   \tl_if_empty:NTF \l__unravel_head_tl
2524     { \_unravel_back_input: } { \_unravel_scan_param_aux: }
2525 }
```

(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.

```
2526 \cs_new_protected:Npn \_unravel_scan_group_n:N #1
2527 {
2528   \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
2529   \_unravel_back_input:
2530   \_unravel_input_gpop_item:NF \l__unravel_head_tl
2531   {
2532     \_unravel_error:nnnnn { runaway-text } { } { } { } { }
2533     \_unravel_exit_hard:w
2534   }
2535   \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2536   \bool_if:NT #1
2537     { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2538   \_unravel_prev_input_silent:V \l__unravel_head_tl
2539 }
```

(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.

```

2540 \cs_new_protected:Npn \_unravel_scan_group_x:N #1
2541   {
2542     \_unravel_input_gpop_tl:N \l__unravel_head_tl
2543     \_unravel_back_input:V \l__unravel_head_tl
2544     \bool_if:NTF #1
2545       {
2546         \_unravel_prev_input_silent:V \c_left_brace_str
2547         \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2548         \_unravel_scan_group_xdef:n { 1 }
2549       }
2550       {
2551         \_unravel_prev_input_gpush_gtl:
2552         \_unravel_prev_input_gtl:N \l__unravel_head_gtl
2553         \_unravel_scan_group_x:n { 1 }
2554         \_unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2555         \_unravel_prev_input_silent:x
2556         { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2557       }
2558   }

```

(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.

```

2559 \cs_new_protected:Npn \_unravel_scan_group_xdef:n #1
2560   {
2561     \_unravel_get_token_xdef:
2562     \tl_if_empty:NTF \l__unravel_head_tl
2563       {
2564         \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2565         {
2566           \_unravel_prev_input_silent:V \c_left_brace_str
2567           \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2568           \_unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2569         }
2570         {
2571           \_unravel_prev_input_silent:V \c_right_brace_str
2572           \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2573           \int_compare:nNnF {#1} = 1
2574           { \_unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2575         }
2576       }
2577       {
2578         \_unravel_prev_input_silent:V \l__unravel_head_tl
2579         \tl_put_right:Nx \l__unravel_defining_tl
2580         { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2581         \_unravel_scan_group_xdef:n {#1}
2582       }
2583   }
2584 \cs_generate_variant:Nn \_unravel_scan_group_xdef:n { f }

```

(End definition for `_unravel_scan_group_xdef:n`.)

`_unravel_scan_group_x:n`

```
2585 \cs_new_protected:Npn \_unravel_scan_group_x:n #1
2586 {
2587   \_unravel_get_token_x:
2588   \_unravel_prev_input_gtl:N \l_unravel_head_gtl
2589   \tl_if_empty:NTF \l_unravel_head_tl
2590   {
2591     \gtl_if_head_is_group_begin:NTF \l_unravel_head_gtl
2592     { \_unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2593     {
2594       \int_compare:nNnF {#1} = 1
2595       { \_unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2596     }
2597   }
2598   { \_unravel_scan_group_x:n {#1} }
2599 }
2600 \cs_generate_variant:Nn \_unravel_scan_group_x:n { f }
```

(End definition for _unravel_scan_group_x:n.)

`_unravel_scan_alt_rule:`

```
2601 \cs_new_protected:Npn \_unravel_scan_alt_rule:
2602 {
2603   \_unravel_scan_keyword:nTF { wWiDdDtThH }
2604   {
2605     \_unravel_scan_normal_dimen:
2606     \_unravel_scan_alt_rule:
2607   }
2608   {
2609     \_unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2610     {
2611       \_unravel_scan_normal_dimen:
2612       \_unravel_scan_alt_rule:
2613     }
2614     {
2615       \_unravel_scan_keyword:nT { dDeEpPtThH }
2616       {
2617         \_unravel_scan_normal_dimen:
2618         \_unravel_scan_alt_rule:
2619       }
2620     }
2621   }
2622 }
```

(End definition for _unravel_scan_alt_rule:.)

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

```
2623 \cs_new_protected:Npn \_unravel_scan_spec:
2624 {
2625   \_unravel_scan_keyword:nTF { tTo0 } { \_unravel_scan_normal_dimen: }
2626   {
2627     \_unravel_scan_keyword:nT { sSpPrReEaAdD }
2628     { \_unravel_scan_normal_dimen: }
2629   }
}
```

```

2630     \__unravel_scan_left_brace:
2631   }

```

(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.

```

2632 \cs_new_protected:Npn \__unravel_do_box:N #1
2633   {
2634     \__unravel_get_x_non_relax:
2635     \__unravel_set_cmd:
2636     \int_compare:nNnTF
2637       \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2638       { \__unravel_do_begin_box:N #1 }
2639       {
2640         \bool_if:NTF #1
2641         {
2642           \int_case:nnTF \l__unravel_head_cmd_int
2643             {
2644               { \__unravel_tex_use:n { hrule } } { }
2645               { \__unravel_tex_use:n { vrule } } { }
2646             }
2647             { \__unravel_do_leaders_rule: }
2648             { \__unravel_do_box_error: }
2649           }
2650           { \__unravel_do_box_error: }
2651         }
2652       }

```

(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.

```

2653 \cs_new_protected:Npn \__unravel_do_box_error:
2654   {
2655     \__unravel_back_input:
2656     \__unravel_error:nnnnn { missing-box } { } { } { } { }
2657     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2658     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2659   }

```


(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.

```
2660 \cs_new_protected:Npn \_unravel\_do\_begin\_box:N #1
2661   {
2662     \_unravel\_prev\_input:V \l\_unravel\_head\_tl
2663     \int\_case:nmTF \l\_unravel\_head\_char\_int
2664     {
2665       { 0 } { \_unravel\_scan\_int: } % box
2666       { 1 } { \_unravel\_scan\_int: } % copy
2667       { 2 } { } % lastbox
2668       { 3 } % vsplit
2669       {
2670         \_unravel\_scan\_int:
2671         \_unravel\_scan\_to:
2672         \_unravel\_scan\_normal\_dimen:
2673       }
2674     }
2675     { \_unravel\_do\_simple\_box:N #1 }
2676     { \_unravel\_do\_box\_explicit:N #1 }
2677   }
```

(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.

```
2678 \cs_new_protected:Npn \_unravel\_do\_simple\_box:N #1
2679   {
2680     \bool\_if:NNTF #1 { \_unravel\_do\_leaders\_fetch\_skip: }
2681     {
2682       \_unravel\_prev\_input\_gpop:N \l\_unravel\_head\_tl
2683       \tl\_if\_head\_eq\_meaning:VNT \l\_unravel\_head\_tl \tex\_shipout:D
2684       { \_unravel\_prepare\_mag: }
2685       \tl\_use:N \l\_unravel\_head\_tl \scan\_stop:
2686       \gtl\_gput\_right:NV \g\_unravel\_output\_gtl \l\_unravel\_head\_tl
2687       \_unravel\_print\_action:x { \tl\_to\_str:N \l\_unravel\_head\_tl }
2688     }
2689   }
```

(End definition for `_unravel_do_simple_box:N`.)

`_unravel_do_leaders_fetch_skip:`

```
2690 \cs_new_protected:Npn \_unravel\_do\_leaders\_fetch\_skip:
2691   {
2692     \_unravel\_get\_x\_non\_relax:
2693     \_unravel\_set\_cmd:
2694     \int\_compare:nNnTF \l\_unravel\_head\_cmd\_int
2695     = { \_unravel\_tex\_use:n { \mode\_if\_vertical:TF { vskip } { hskip } } }
2696     {
2697       \_unravel\_prev\_input\_gpop:N \l\_unravel\_tmpa\_tl
```

```

2698     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2699     \__unravel_do_append_glue:
2700   }
2701   {
2702     \__unravel_back_input:
2703     \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2704     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2705     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2706   }
2707 }

```

(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 TeX 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 TeX 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). TeX 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.

```

2708 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2709 {
2710   \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2711   { \__unravel_box_hook:N \tex_everyhbox:D }
2712   { \__unravel_box_hook:N \tex_everyvbox:D }
2713   % ^^A todo: TeX calls |normal_paragraph| here.
2714   \__unravel_scan_spec:
2715   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2716   \__unravel_set_action_text:x
2717   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2718   \seq_push:Nf \l__unravel_leaders_box_seq
2719   { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2720   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2721   \gtl_gconcat:NNN \g__unravel_output_gtl
2722   \g__unravel_output_gtl \c_group_begin_gtl
2723   \tl_use:N \l__unravel_head_tl
2724   \c_group_begin_token \__unravel_box_hook_end:
2725 }

```

(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`
`__unravel_box_hook:w` too much (just add one token at the start). The various o-expansions remove `\prg_do_`
`__unravel_box_hook_end:` `nothing:`, used to avoid losing braces.

```

2726 \cs_new_protected:Npn \__unravel_box_hook:N #1
2727 {
2728   \tl_set:NV \l__unravel_tmpa_tl #1
2729   \str_if_eq:eeF
2730   { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2731   {
2732     \__unravel_exp_args:Nx #1

```

```

2733     {
2734     \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2735     \exp_not:V #1
2736     }
2737   }
2738 \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2739 {
2740   \exp_args:No #1 {##1}
2741   \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2742   \gtl_clear:N \l__unravel_after_group_gtl
2743   \__unravel_print_action:
2744   \__unravel_back_input:o {##1}
2745   \__unravel_set_action_text:x
2746     { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2747   \tl_if_empty:oF {##1} { \__unravel_print_action: }
2748 }
2749 }
2750 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2751 \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.

```

2752 \cs_new_protected:Npn \__unravel_do_leaders_rule:
2753 {
2754   \__unravel_prev_input:V \l__unravel_head_tl
2755   \__unravel_scan_alt_rule:
2756   \__unravel_do_leaders_fetch_skip:
2757 }

```

(End definition for __unravel_do_leaders_rule:.)

2.9 Paragraphs

__unravel_charcode_if_safe:nTF

```

2758 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2759 {
2760   \bool_if:nTF
2761     {
2762       \int_compare_p:n { #1 = '!' }
2763       || \int_compare_p:n { ' ' <= #1 <= '[' }
2764       || \int_compare_p:n { #1 = ']' }
2765       || \int_compare_p:n { ' ' <= #1 <= 'z' }
2766     }
2767     { \prg_return_true: }
2768     { \prg_return_false: }
2769 }

```

(End definition for __unravel_charcode_if_safe:nTF.)

__unravel_char:n
__unravel_char:V
__unravel_char:x

```

2770 \cs_new_protected:Npn \__unravel_char:n #1

```

```

2771 {
2772   \tex_char:D #1 \scan_stop:
2773   \__unravel_charcode_if_safe:nTF {#1}
2774     { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2775     {
2776       \tl_set:Nx \l__unravel_tmpa_tl
2777         { \exp_not:N \char \int_eval:n {#1} ~ }
2778     }
2779   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2780   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2781 }
2782 \cs_generate_variant:Nn \__unravel_char:n { V }
2783 \cs_new_protected:Npn \__unravel_char:x
2784   { \__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
2785 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2786 {
2787   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2788     { % math active
2789       \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_head_gtl
2790         { \char_generate:nn {#1} { 12 } }
2791       \__unravel_back_input:
2792     }
2793     { \__unravel_char:n {#1} }
2794 }
2795 \cs_generate_variant:Nn \__unravel_char_in_mmode:n { V }
2796 \cs_new_protected:Npn \__unravel_char_in_mmode:x
2797   { \__unravel_exp_args:Nx \__unravel_char_in_mmode:n }

```

(End definition for `__unravel_char_in_mmode:n`.)

```

\__unravel_mathchar:n
\__unravel_mathchar:x
2798 \cs_new_protected:Npn \__unravel_mathchar:n #1
2799 {
2800   \tex_mathchar:D #1 \scan_stop:
2801   \tl_set:Nx \l__unravel_tmpa_tl
2802     { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2803   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2804   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2805 }
2806 \cs_new_protected:Npn \__unravel_mathchar:x
2807   { \__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.

```

2808 \cs_new_protected:Npn \__unravel_new_graf:N #1
2809 {

```

```

2810 \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
2811 \__unravel_everypar:w { }
2812 \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2813 \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl
2814 \__unravel_back_input:V \l__unravel_tmpa_tl
2815 \__unravel_print_action:x
2816 {
2817   \g__unravel_action_text_str \c_space_tl : ~
2818   \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2819 }
2820 }

```

(End definition for __unravel_new_graf:N.)

__unravel_end_graf:

```

2821 \cs_new_protected:Npn \__unravel_end_graf:
2822 { \mode_if_horizontal:T { \__unravel_normal_paragraph: } }

```

(End definition for __unravel_end_graf:.)

__unravel_normal_paragraph:

```

2823 \cs_new_protected:Npn \__unravel_normal_paragraph:
2824 {
2825   \tex_par:D
2826   \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2827   \__unravel_print_action:x { Paragraph~end. }
2828 }

```

(End definition for __unravel_normal_paragraph:.)

__unravel_build_page:

```

2829 \cs_new_protected:Npn \__unravel_build_page:
2830 {
2831 }

```

(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.

```

2832 \cs_new_protected:Npn \__unravel_handle_right_brace:
2833 {
2834   \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
2835   {
2836     \gtl_gconcat:NNN \g__unravel_output_gtl
2837     \g__unravel_output_gtl \c_group_end_gtl
2838     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2839     \int_case:nn \__unravel_currentgrouptype:
2840     {
2841       { 1 } { \__unravel_end_simple_group: } % simple
2842       { 2 } { \__unravel_end_box_group: } % hbox
2843       { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2844       { 4 } { \__unravel_end_graf: \__unravel_end_box_group: } % vbox
2845       { 5 } { \__unravel_end_graf: \__unravel_end_box_group: } % vtop
2846       { 6 } { \__unravel_end_align_group: } % align

```

```

2847         { 7 } { \_unravel_end_no_align_group: } % no_align
2848         { 8 } { \_unravel_end_output_group: } % output
2849         { 9 } { \_unravel_end_simple_group: } % math
2850         { 10 } { \_unravel_end_disc_group: } % disc
2851         { 11 } { \_unravel_end_graf: \_unravel_end_simple_group: } % insert
2852         { 12 } { \_unravel_end_graf: \_unravel_end_simple_group: } % vcenter
2853         { 13 } { \_unravel_end_math_choice_group: } % math_choice
2854     }
2855 }
2856 { % bottom_level, semi_simple, math_shift, math_left
2857     \l_unravel_head_token
2858     \_unravel_print_action:
2859 }
2860 }

```

(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.

```

2861 \cs_new_protected:Npn \_unravel_end_simple_group:
2862 {
2863     \l_unravel_head_token
2864     \_unravel_print_action:
2865 }

```

(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.

```

2866 \cs_new_protected:Npn \_unravel_end_box_group:
2867 {
2868     \seq_pop:NN \l_unravel_leaders_box_seq \l_unravel_tmpa_tl
2869     \exp_args:No \_unravel_end_box_group_aux:n { \l_unravel_tmpa_tl }
2870 }
2871 \cs_new_protected:Npn \_unravel_end_box_group_aux:n #1
2872 {
2873     \str_if_eq:eeTF {#1} { Z }
2874     { \_unravel_end_simple_group: }
2875     {
2876         \_unravel_get_x_non_relax:
2877         \_unravel_set_cmd:
2878         \int_compare:nNnTF \l_unravel_head_cmd_int
2879         = { \_unravel_tex_use:n { #1 skip } }
2880         {
2881             \tl_put_left:Nn \l_unravel_head_tl { \c_group_end_token }
2882             \_unravel_do_append_glue:
2883         }
2884         {
2885             \_unravel_back_input:
2886             \c_group_end_token \group_begin: \group_end:
2887             \_unravel_print_action:
2888         }
2889     }
2890 }

```

(End definition for `_unravel_end_box_group:`.)

`_unravel_off_save:`

```

2891 \cs_new_protected:Npn \_unravel_off_save:
2892 {
2893   \int_compare:nNnTF \_unravel_currentgrouptype: = { 0 }
2894     { % bottom-level
2895       \_unravel_error:nxxxx { extra-close }
2896       { \token_to_meaning:N \l\_unravel_head_token } { } { } { } }
2897     }
2898   {
2899     \_unravel_back_input:
2900     \int_case:nnF \_unravel_currentgrouptype:
2901       {
2902         { 14 } % semi_simple_group
2903         { \gtl_set:Nn \l\_unravel_head_gtl { \group_end: } }
2904         { 15 } % math_shift_group
2905         { \gtl_set:Nn \l\_unravel_head_gtl { $ } } % $
2906         { 16 } % math_left_group
2907         { \gtl_set:Nn \l\_unravel_head_gtl { \tex_right:D . } }
2908       }
2909       { \gtl_set_eq:NN \l\_unravel_head_gtl \c_group_end_gtl }
2910     \_unravel_back_input:
2911     \_unravel_error:nxxxx { off-save }
2912     { \gtl_to_str:N \l\_unravel_head_gtl } { } { } { } { } }
2913   }
2914 }

```

(End definition for `_unravel_off_save:`.)

2.11 Modes

`_unravel_mode_math:n`
`_unravel_mode_non_math:n`
`_unravel_mode_vertical:n`

```

2915 \cs_new_protected:Npn \_unravel_mode_math:n #1
2916 { \mode_if_math:TF {#1} { \_unravel_insert_dollar_error: } }
2917 \cs_new_protected:Npn \_unravel_mode_non_math:n #1
2918 { \mode_if_math:TF { \_unravel_insert_dollar_error: } {#1} }
2919 \cs_new_protected:Npn \_unravel_mode_vertical:n #1
2920 {
2921   \mode_if_math:TF
2922     { \_unravel_insert_dollar_error: }
2923     { \mode_if_horizontal:TF { \_unravel_head_for_vmode: } {#1} }
2924 }
2925 \cs_new_protected:Npn \_unravel_mode_non_vertical:n #1
2926 {
2927   \mode_if_vertical:TF
2928     { \_unravel_back_input: \_unravel_new_graf:N \c_true_bool }
2929     {#1}
2930 }

```

(End definition for `_unravel_mode_math:n`, `_unravel_mode_non_math:n`, and `_unravel_mode_vertical:n`.)

`__unravel_head_for_vmode:` See TeX's `head_for_vmode`.

```
2931 \cs_new_protected:Npn \__unravel_head_for_vmode:
2932 {
2933   \mode_if_inner:TF
2934   {
2935     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
2936     {
2937       \__unravel_error:nmmnn { hrulerule-bad-mode } { } { } { } { }
2938       \__unravel_print_action:
2939     }
2940     { \__unravel_off_save: }
2941   }
2942   {
2943     \__unravel_back_input:
2944     \gtl_set:Nn \l__unravel_head_gtl { \par }
2945     \__unravel_back_input:
2946   }
2947 }
```

(End definition for `__unravel_head_for_vmode:.`)

`__unravel_goto_inner_math:`

```
2948 \cs_new_protected:Npn \__unravel_goto_inner_math:
2949 {
2950   \__unravel_box_hook:N \tex_everymath:D
2951   $ % $
2952   \__unravel_box_hook_end:
2953 }
```

(End definition for `__unravel_goto_inner_math:.`)

`__unravel_goto_display_math:`

```
2954 \cs_new_protected:Npn \__unravel_goto_display_math:
2955 {
2956   \__unravel_box_hook:N \tex_everydisplay:D
2957   $ $
2958   \__unravel_box_hook_end:
2959 }
```

(End definition for `__unravel_goto_display_math:.`)

`__unravel_after_math:`

```
2960 \cs_new_protected:Npn \__unravel_after_math:
2961 {
2962   \mode_if_inner:TF
2963   {
2964     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2965     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2966     $ % $
2967   }
2968   {
2969     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2970     \__unravel_get_x_next:
2971     \token_if_eq_catcode:NMF
```



```

2972         \l__unravel_head_token \c_math_toggle_token
2973     {
2974         \__unravel_back_input:
2975         \tl_set:Nn \l__unravel_head_tl { $ } % $
2976         \__unravel_error:nmmnn { missing-dollar } { } { } { } { }
2977     }
2978     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2979     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2980     $ $
2981 }
2982 \__unravel_print_action:
2983 }

```

(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.

```

2984 \cs_new_protected:Npn \__unravel_do_step:
2985 {
2986     \__unravel_set_action_text:
2987     \bool_if:NT \g__unravel_internal_debug_bool
2988     { \__unravel_exp_args:Nx \iow_term:n { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int
2989     \cs_if_exist_use:cF
2990     { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2991     { \__unravel_error:nxxxx { internal } { unknown-command } { } { } { } }
2992 }

```

(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 `\crr`.
- `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.

```

2993 \__unravel_new_tex_cmd:nn { relax } % 0
2994 {
2995   \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:
2996   {
2997     \exp_after:wN \__unravel_token_if_expandable:NNTF \l__unravel_head_tl
2998     {
2999       \__unravel_set_action_text:x
3000       { \iow_char:N \notexpanded: \g__unravel_action_text_str }
3001     }
3002     { }
3003   }
3004   \__unravel_print_action:
3005 }

```

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.

```

3006 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
3007 {
3008   \gtl_gconcat:NNN \g__unravel_output_gtl
3009   \g__unravel_output_gtl \c_group_begin_gtl
3010   \__unravel_print_action:
3011   \l__unravel_head_token
3012   \gtl_clear:N \l__unravel_after_group_gtl
3013 }
3014 \__unravel_new_tex_cmd:nn { end-group_char } % 2
3015 { \__unravel_handle_right_brace: }

```

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

```

3016 \__unravel_new_tex_cmd:nn { math_char } % 3
3017 {
3018   \__unravel_mode_non_vertical:n
3019   {
3020     \mode_if_math:TF
3021     {
3022       \int_compare:nNnTF
3023       \__unravel_currentgrouptype: = { 15 } % math_shift_group

```

```

3024         { \__unravel_after_math: }
3025         { \__unravel_off_save: }
3026     }
3027     {
3028     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3029     \__unravel_get_next:
3030     \token_if_eq_catcode:NNTF
3031     \l__unravel_head_token \c_math_toggle_token
3032     {
3033         \mode_if_inner:TF
3034         { \__unravel_back_input: \__unravel_goto_inner_math: }
3035         {
3036             \gtl_gput_right:NV
3037             \g__unravel_output_gtl \l__unravel_head_tl
3038             \__unravel_goto_display_math:
3039         }
3040     }
3041     { \__unravel_back_input: \__unravel_goto_inner_math: }
3042 }
3043 }
3044 }

```

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.

```

3045 \__unravel_new_tex_cmd:nn { alignment_char } % 4
3046 { \l__unravel_head_token \__unravel_print_action: }
3047 \__unravel_new_tex_cmd:nn { car_ret } % 5
3048 { \l__unravel_head_token \__unravel_print_action: }
3049 \__unravel_new_tex_cmd:nn { macro_char } % 6
3050 { \l__unravel_head_token \__unravel_print_action: }
3051 \__unravel_new_tex_cmd:nn { superscript_char } % 7
3052 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3053 \__unravel_new_tex_cmd:nn { subscript_char } % 8
3054 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3055 \cs_new_protected:Npn \__unravel_sub_sup:
3056 {
3057     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3058     \__unravel_print_action:
3059     \__unravel_get_x_non_relax:
3060     \__unravel_set_cmd:
3061     \int_case:nnTF \l__unravel_head_cmd_int
3062     {
3063         { \__unravel_tex_use:n { the_char } }
3064         { \__unravel_prev_input:V \l__unravel_head_tl }
3065         { \__unravel_tex_use:n { other_char } }
3066         { \__unravel_prev_input:V \l__unravel_head_tl }
3067         { \__unravel_tex_use:n { char_given } }
3068         { \__unravel_prev_input:V \l__unravel_head_tl }
3069         { \__unravel_tex_use:n { char_num } }
3070         {
3071             \__unravel_prev_input:V \l__unravel_head_tl
3072             \__unravel_scan_int:
3073         }

```

```

3074     { \_unravel_tex_use:n { math_char_num } }
3075     {
3076         \_unravel_prev_input:V \l__unravel_head_tl
3077         \_unravel_scan_int:
3078     }
3079     { \_unravel_tex_use:n { math_given } }
3080     { \_unravel_prev_input:V \l__unravel_head_tl }
3081     { \_unravel_tex_use:n { delim_num } }
3082     { \_unravel_prev_input:V \l__unravel_head_tl \_unravel_scan_int: }
3083 }
3084 {
3085     \_unravel_prev_input_gpop:N \l__unravel_head_tl
3086     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3087     \tl_use:N \l__unravel_head_tl \scan_stop:
3088 }
3089 {
3090     \_unravel_back_input:
3091     \_unravel_scan_left_brace:
3092     \_unravel_prev_input_gpop:N \l__unravel_head_tl
3093     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3094     \gtl_gconcat:NNN \g__unravel_output_gtl
3095     \g__unravel_output_gtl \c_group_begin_gtl
3096     \tl_use:N \l__unravel_head_tl \c_group_begin_token
3097 }
3098 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3099 }
3100 \_unravel_new_tex_cmd:nn { endv } % 9
3101 { \_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.

```

3102 \_unravel_new_tex_cmd:nn { blank_char } % 10
3103 {
3104     \mode_if_horizontal:T
3105     {
3106         \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
3107         \l__unravel_head_token
3108     }
3109     \_unravel_print_action:
3110 }

```

Letters and other characters leave vertical mode.

```

3111 \_unravel_new_tex_cmd:nn { the_char } % 11
3112 {
3113     \_unravel_mode_non_vertical:n
3114     {
3115         \tl_set:Nx \l__unravel_tmpa_tl
3116         { ' \_unravel_token_to_char:N \l__unravel_head_token }
3117         \mode_if_math:TF
3118         { \_unravel_char_in_mmode:V \l__unravel_tmpa_tl }
3119         { \_unravel_char:V \l__unravel_tmpa_tl }
3120     }
3121 }
3122 \_unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12

```

```

3123 \__unravel_new_tex_cmd:nn { par_end } % 13
3124 {
3125   \__unravel_mode_non_math:n
3126   {
3127     \mode_if_vertical:TF
3128     { \__unravel_normal_paragraph: }
3129     {
3130       % if align_state<0 then off_save;
3131       \__unravel_end_graf:
3132       \mode_if_vertical:T
3133       { \mode_if_inner:F { \__unravel_build_page: } }
3134     }
3135   }
3136 }

3137 \__unravel_new_tex_cmd:nn { stop } % 14
3138 {
3139   \__unravel_mode_vertical:n
3140   {
3141     \mode_if_inner:TF
3142     { \__unravel_forbidden_case: }
3143     {
3144       % ^^A todo: unless its_all_over
3145       \int_gdecr:N \g__unravel_ends_int
3146       \int_compare:nNnTF \g__unravel_ends_int > 0
3147       {
3148         \__unravel_back_input:
3149         \__unravel_back_input:n
3150         {
3151           \__unravel_hbox:w to \tex_hsize:D { }
3152           \tex_vfill:D
3153           \tex_penalty:D - '1000000000 ~
3154         }
3155         \__unravel_build_page:
3156         \__unravel_print_action:x { End-everything! }
3157       }
3158       {
3159         \__unravel_print_outcome:
3160         \l__unravel_head_token
3161       }
3162     }
3163   }
3164 }

3165 \__unravel_new_tex_cmd:nn { delim_num } % 15
3166 {
3167   \__unravel_mode_math:n
3168   {
3169     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3170     \__unravel_print_action:
3171     \__unravel_scan_int:
3172     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3173     \tl_use:N \l__unravel_head_tl \scan_stop:
3174     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3175   }

```

3176 }

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`.

```
3177 \__unravel_new_tex_cmd:nn { char_num } % 16
3178 {
3179   \__unravel_mode_non_vertical:n
3180   {
3181     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3182     \__unravel_print_action:
3183     \__unravel_scan_int:
3184     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3185     \mode_if_math:TF
3186     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
3187     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
3188   }
3189 }
```

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.

```

3190 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3191 {
3192   \__unravel_mode_math:n
3193   {
3194     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3195     \__unravel_print_action:
3196     \__unravel_scan_int:
3197     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3198     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
3199   }
3200 }

```

```

3201 \__unravel_new_tex_cmd:nn { mark } % 18
3202 {
3203   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3204   \__unravel_print_action:
3205   \int_compare:nNnF \l__unravel_head_char_int = 0
3206   { \__unravel_scan_int: }
3207   \__unravel_prev_input_gpush:
3208   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3209   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3210   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3211   \__unravel_print_action:x
3212   { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3213   \tl_put_right:Nx \l__unravel_head_tl
3214   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3215   \tl_use:N \l__unravel_head_tl
3216 }

```

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.

```

3217 \__unravel_new_tex_cmd:nn { xray } % 19
3218 {
3219   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3220   \__unravel_print_action:
3221   \int_case:nnF \l__unravel_head_char_int
3222   {
3223     { 0 }
3224     { % show
3225       \__unravel_get_next:
3226       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3227       \token_if_eq_meaning:NNTF
3228       \l__unravel_head_token \__unravel_special_relax:
3229       {
3230         \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
3231         \exp_after:wN \exp_not:N \l__unravel_head_tl
3232       }
3232     }
3232 }

```

```

3233         { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
3234     }
3235     { 2 }
3236     { % showthe
3237         \__unravel_get_x_next:
3238         \__unravel_scan_something_internal:n { 5 }
3239         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3240         \__unravel_exp_args:Nx \use:n
3241         { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3242     }
3243 }
3244 { % no operand for showlists, showgroups, showifs
3245     \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
3246     { \__unravel_scan_int: }
3247     \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
3248     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3249     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3250     \tl_use:N \l__unravel_head_tl \scan_stop:
3251 }
3252 }
    make_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
3253 \__unravel_new_tex_cmd:nn { make_box } % 20
3254 {
3255     \__unravel_prev_input_gpush:
3256     \__unravel_back_input:
3257     \__unravel_do_box:N \c_false_bool
3258 }

\__unravel_do_move: Scan a dimension and a box, and perform the shift, printing the appropriate action.
3259 \cs_new_protected:Npn \__unravel_do_move:
3260 {
3261     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3262     \__unravel_print_action:
3263     \__unravel_scan_normal_dimen:
3264     \__unravel_do_box:N \c_false_bool
3265 }

(End definition for \__unravel_do_move:.)
    hmove=21 for \moveright and \moveleft.
3266 \__unravel_new_tex_cmd:nn { hmove } % 21
3267 {
3268     \mode_if_vertical:TF
3269     { \__unravel_do_move: } { \__unravel_forbidden_case: }
3270 }

    vmove=22 for \lower and \raise.
3271 \__unravel_new_tex_cmd:nn { vmove } % 22
3272 {
3273     \mode_if_vertical:TF
3274     { \__unravel_forbidden_case: } { \__unravel_do_move: }
3275 }

```


`__unravel_do_unpackage:`

```

3276 \cs_new_protected:Npn \__unravel_do_unpackage:
3277 {
3278   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3279   \__unravel_print_action:
3280   \__unravel_scan_int:
3281   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3282   \tl_use:N \l__unravel_head_tl \scan_stop:
3283   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3284 }

```

(End definition for __unravel_do_unpackage:.)

`un_hbox=23` for `\unhbox` and `\unhcopy`.

```

3285 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3286 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

```

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

```

3287 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3288 {
3289   \__unravel_mode_vertical:n
3290   {
3291     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3292     { \l__unravel_head_token \__unravel_print_action: }
3293     { \__unravel_do_unpackage: }
3294   }
3295 }

```

`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.

```

3296 \__unravel_new_tex_cmd:nn { remove_item } % 25
3297 { \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.

```

3298 \cs_new_protected:Npn \__unravel_do_append_glue:
3299 {
3300   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3301   { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3302   {
3303     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3304     \__unravel_print_action:
3305     \exp_args:Nf \__unravel_scan_glue:n
3306     { \int_eval:n { \l__unravel_head_char_int - 2 } }
3307     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3308     \tl_use:N \l__unravel_head_tl \scan_stop:
3309     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3310   }
3311 }

```

(End definition for `_unravel_do_append_glue:`.)

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

3312 `_unravel_new_tex_cmd:nn { hskip }` % 26

3313 `{ _unravel_mode_non_vertical:n { _unravel_do_append_glue: } }`

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

3314 `_unravel_new_tex_cmd:nn { vskip }` % 27

3315 `{ _unravel_mode_vertical:n { _unravel_do_append_glue: } }`

`mskip=28` for `\mskip (5)`.

3316 `_unravel_new_tex_cmd:nn { mskip }` % 28

3317 `{ _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.

3318 `\cs_new_protected:Npn _unravel_do_append_kern:`

3319 `{`

3320 `_unravel_prev_input_gpush:N \l_unravel_head_tl`

3321 `_unravel_print_action:`

3322 `\token_if_eq_meaning:NNTF \l_unravel_head_token \tex_kern:D`

3323 `{ _unravel_scan_dimen:nN { 2 } \c_false_bool }`

3324 `{ _unravel_scan_dimen:nN { 3 } \c_false_bool }`

3325 `_unravel_prev_input_gpop:N \l_unravel_head_tl`

3326 `\tl_use:N \l_unravel_head_tl \scan_stop:`

3327 `_unravel_print_action:x { \tl_to_str:N \l_unravel_head_tl }`

3328 `}`

(End definition for `_unravel_do_append_kern:`.)

`kern=29` for `\kern (1)`.

3329 `_unravel_new_tex_cmd:nn { kern }` % 29

3330 `{ _unravel_do_append_kern: }`

`mkern=30` for `\mkern (99)`.

3331 `_unravel_new_tex_cmd:nn { mkern }` % 30

3332 `{ _unravel_mode_math:n { _unravel_do_append_kern: } }`

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

3333 `_unravel_new_tex_cmd:nn { leader_ship }` % 31

3334 `{`

3335 `_unravel_prev_input_gpush:N \l_unravel_head_tl`

3336 `_unravel_print_action:`

3337 `_unravel_do_box:N \c_true_bool`

3338 `}`

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

```

3339 \_unravel_new_tex_cmd:nn { halign } % 32
3340 { \_unravel_not_implemented:n { halign } }
3341 \_unravel_new_tex_cmd:nn { valign } % 33
3342 { \_unravel_not_implemented:n { valign } }
3343 \_unravel_new_tex_cmd:nn { no_align } % 34
3344 { \_unravel_not_implemented:n { noalign } }

3345 \_unravel_new_tex_cmd:nn { vrule } % 35
3346 { \_unravel_mode_non_vertical:n { \_unravel_do_rule: } }
3347 \_unravel_new_tex_cmd:nn { hrule } % 36
3348 { \_unravel_mode_vertical:n { \_unravel_do_rule: } }
3349 \cs_new_protected:Npn \_unravel_do_rule:
3350 {
3351 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3352 \_unravel_print_action:
3353 \_unravel_scan_alt_rule:
3354 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3355 \tl_use:N \l__unravel_head_tl \scan_stop:
3356 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3357 }

3358 \_unravel_new_tex_cmd:nn { insert } % 37
3359 { \_unravel_begin_insert_or_adjust: }
3360 \_unravel_new_tex_cmd:nn { vadjust } % 38
3361 {
3362 \mode_if_vertical:TF
3363 { \_unravel_forbidden_case: } { \_unravel_begin_insert_or_adjust: }
3364 }

3365 \_unravel_new_tex_cmd:nn { ignore_spaces } % 39
3366 {
3367 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3368 {
3369 \_unravel_print_action:
3370 \_unravel_get_x_non_blank:
3371 \_unravel_set_cmd:

```

```

3372     \__unravel_do_step:
3373     }
3374     { \__unravel_not_implemented:n { pdfprimitive } }
3375 }
3376 \__unravel_new_tex_cmd:nn { after_assignment }           % 40
3377 {
3378   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3379   \__unravel_get_next:
3380   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3381   \__unravel_print_action:x
3382   {
3383     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3384     \gtl_to_str:N \l__unravel_head_gtl
3385   }
3386 }

```

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.

```

3387 \__unravel_new_tex_cmd:nn { after_group }               % 41
3388 {
3389   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3390   \__unravel_get_next:
3391   \int_compare:nNnTF \__unravel_currentgroupstype: = 0
3392   {
3393     \__unravel_print_action:x
3394     {
3395       Aftergroup~(level~0~=>~dropped):~
3396       \tl_to_str:N \l__unravel_tmpa_tl
3397       \gtl_to_str:N \l__unravel_head_gtl
3398     }
3399   }
3400   {
3401     \gtl_concat:NNN \l__unravel_after_group_gtl
3402     \l__unravel_after_group_gtl \l__unravel_head_gtl
3403     \__unravel_print_action:x
3404     {
3405       Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3406       \gtl_to_str:N \l__unravel_head_gtl
3407     }
3408   }
3409 }

```

See `__unravel_do_append_glue:`.

```

3410 \__unravel_new_tex_cmd:nn { break_penalty }             % 42
3411 {
3412   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3413   \__unravel_print_action:
3414   \__unravel_scan_int:
3415   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3416   \tl_use:N \l__unravel_head_tl \scan_stop:
3417   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3418 }
3419 \__unravel_new_tex_cmd:nn { start_par }                 % 43
3420 {

```

```

3421 \mode_if_vertical:TF
3422 {
3423   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3424   { \__unravel_new_graf:N \c_false_bool }
3425   { \__unravel_new_graf:N \c_true_bool }
3426 }
3427 {
3428   \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3429   {
3430     \__unravel_hbox:w width \tex_parindent:D { }
3431     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3432   }
3433   \__unravel_print_action:
3434 }
3435 }
3436 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3437 {
3438   \mode_if_vertical:TF { \__unravel_forbidden_case: }
3439   { \l__unravel_head_token \__unravel_print_action: }
3440 }

```

__unravel_do_accent:

```

3441 \cs_new_protected:Npn \__unravel_do_accent:
3442 {
3443   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3444   \__unravel_print_action:
3445   \__unravel_scan_int:
3446   \__unravel_do_assignments:
3447   \bool_if:nTF
3448   {
3449     \token_if_eq_catcode_p:NN
3450     \l__unravel_head_token \c_catcode_letter_token
3451     ||
3452     \token_if_eq_catcode_p:NN
3453     \l__unravel_head_token \c_catcode_other_token
3454     ||
3455     \int_compare_p:nNn
3456     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3457   }
3458   { \__unravel_prev_input:V \l__unravel_head_tl }
3459   {
3460     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3461     {
3462       \__unravel_prev_input:V \l__unravel_head_tl
3463       \__unravel_scan_int:
3464     }
3465     { \__unravel_break:w }
3466   }
3467   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3468   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3469   \tl_use:N \l__unravel_head_tl \scan_stop:
3470   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3471   \__unravel_break_point:
3472 }

```

(End definition for `_unravel_do_accent:`)

`_unravel_do_math_accent:` T_EX will complain if `\l_unravel_head_tl` happens to start with `\accent` (the user used `\accent` in math mode).

```

3473 \cs_new_protected:Npn \_unravel_do_math_accent:
3474   {
3475     \_unravel_prev_input_gpush:N \l\_unravel_head_tl
3476     \_unravel_print_action:
3477     \_unravel_scan_int:
3478     \_unravel_scan_math:
3479     \_unravel_prev_input_gpop:N \l\_unravel_head_tl
3480     \gtl_gput_right:NV \g\_unravel_output_gtl \l\_unravel_head_tl
3481     \tl_use:N \l\_unravel_head_tl \scan_stop:
3482     \_unravel_print_action:x { \tl_to_str:N \l\_unravel_head_tl }
3483   }

```

(End definition for `_unravel_do_math_accent:`)

```

3484 \_unravel_new_tex_cmd:nn { accent } % 45
3485   {
3486     \_unravel_mode_non_vertical:n
3487     {
3488       \mode_if_math:TF
3489       { \_unravel_do_math_accent: } { \_unravel_do_accent: }
3490     }
3491   }
3492 \_unravel_new_tex_cmd:nn { math_accent } % 46
3493   { \_unravel_mode_math:n { \_unravel_do_math_accent: } }
3494 \_unravel_new_tex_cmd:nn { discretionary } % 47
3495   { \_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

```

3496 \_unravel_new_tex_cmd:nn { eq_no } % 48
3497   { \_unravel_not_implemented:n { eqno } }
3498 \_unravel_new_tex_cmd:nn { left_right } % 49
3499   { \_unravel_not_implemented:n { left/right } }

```

```

3500 \_unravel_new_tex_cmd:nn { math_comp } % 50
3501 { \_unravel_not_implemented:n { math-comp } }

3502 \_unravel_new_tex_cmd:nn { limit_switch } % 51
3503 { \_unravel_not_implemented:n { limits } }

3504 \_unravel_new_tex_cmd:nn { above } % 52
3505 { \_unravel_not_implemented:n { above } }

3506 \_unravel_new_tex_cmd:nn { math_style } % 53
3507 { \_unravel_not_implemented:n { math-style } }

3508 \_unravel_new_tex_cmd:nn { math_choice } % 54
3509 { \_unravel_not_implemented:n { math-choice } }

3510 \_unravel_new_tex_cmd:nn { non_script } % 55
3511 { \_unravel_not_implemented:n { non-script } }

3512 \_unravel_new_tex_cmd:nn { vcenter } % 56
3513 { \_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

```

3514 \_unravel_new_tex_cmd:nn { case_shift } % 57
3515 {
3516   \_unravel_prev_input_gpush:N \l_unravel_head_tl
3517   \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3518   \_unravel_prev_input_gpop:N \l_unravel_tmpa_tl
3519   \exp_after:wN \_unravel_case_shift:Nn \l_unravel_tmpa_tl
3520 }
3521 \cs_new_protected:Npn \_unravel_case_shift:Nn #1#2
3522 {

```

```

3523 #1 { \_unravel_back_input:n {#2} }
3524 \_unravel_print_action:x
3525 { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3526 }
3527 \_unravel_new_tex_cmd:nn { message } % 58
3528 {
3529 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3530 \_unravel_print_action:
3531 \_unravel_scan_toks_to_str:
3532 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3533 \tl_use:N \l__unravel_head_tl
3534 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3535 }
3536
3537 Extensions are implemented in a later section.
3538
3539 \_unravel_new_tex_cmd:nn { extension } % 59
3540 {
3541 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3542 \_unravel_print_action:
3543 \_unravel_scan_extension_operands:
3544 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3545 \tl_use:N \l__unravel_head_tl \scan_stop:
3546 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3547 }
3548
3549 \_unravel_new_tex_cmd:nn { in_stream } % 60
3550 {
3551 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3552 \_unravel_print_action:
3553 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3554 {
3555 \_unravel_scan_int:
3556 \_unravel_scan_optional_equals:
3557 \_unravel_scan_file_name:
3558 }
3559 { \_unravel_scan_int: }
3560 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3561 \tl_use:N \l__unravel_head_tl \scan_stop:
3562 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3563 }
3564
3565 \_unravel_new_tex_cmd:nn { begin_group } % 61
3566 {
3567 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3568 \l__unravel_head_token
3569 \gtl_clear:N \l__unravel_after_group_gtl
3570 \_unravel_print_action:
3571 }
3572
3573 \_unravel_new_tex_cmd:nn { end_group } % 62
3574 {
3575 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3576 \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
3577 \l__unravel_head_token
3578 \_unravel_print_action:
3579 }

```



```

3574 \_unravel_new_tex_cmd:nn { omit } % 63
3575 { \_unravel_not_implemented:n { omit } }

3576 \_unravel_new_tex_cmd:nn { ex_space } % 64
3577 {
3578   \_unravel_mode_non_vertical:n
3579   { \l__unravel_head_token \_unravel_print_action: }
3580 }

3581 \_unravel_new_tex_cmd:nn { no_boundary } % 65
3582 {
3583   \_unravel_mode_non_vertical:n
3584   { \l__unravel_head_token \_unravel_print_action: }
3585 }

3586 \_unravel_new_tex_cmd:nn { radical } % 66
3587 {
3588   \_unravel_mode_math:n
3589   {
3590     \_unravel_prev_input_gpush:N \l__unravel_head_tl
3591     \_unravel_print_action:
3592     \_unravel_scan_int:
3593     \_unravel_scan_math:
3594     \_unravel_prev_input_gpop:N \l__unravel_head_tl
3595     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3596     \tl_use:N \l__unravel_head_tl \scan_stop:
3597     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3598   }
3599 }

3600 \_unravel_new_tex_cmd:nn { end_cs_name } % 67
3601 {
3602   \_unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
3603   \_unravel_print_action:
3604 }

See the_char and other_char.

3605 \_unravel_new_tex_cmd:nn { char_given } % 68
3606 {
3607   \_unravel_mode_non_vertical:n
3608   {
3609     \mode_if_math:TF
3610     { \_unravel_char_in_mmode:V \l__unravel_head_char_int }
3611     { \_unravel_char:V \l__unravel_head_char_int }
3612   }
3613 }

See math_char_num.

3614 \_unravel_new_tex_cmd:nn { math_given } % 69
3615 {
3616   \_unravel_mode_math:n
3617   { \_unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3618 }

3619 \_unravel_new_tex_cmd:nn { last_item } % 70
3620 { \_unravel_forbidden_case: }

```

2.13.6 Extensions

_unravel_scan_extension_operands:

```
3621 \cs_new_protected:Npn \_unravel_scan_extension_operands:
3622 {
3623   \int_case:nnF \l__unravel_head_char_int
3624   {
3625     { 0 } % openout
3626     {
3627       \_unravel_scan_int:
3628       \_unravel_scan_optional_equals:
3629       \_unravel_scan_file_name:
3630     }
3631     { 1 } % write
3632     {
3633       \_unravel_scan_int:
3634       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3635     }
3636     { 2 } % closeout
3637     { \_unravel_scan_int: }
3638     { 3 } % special
3639     { \_unravel_scan_toks_to_str: }
3640     { 4 } % immediate
3641     { \_unravel_scan_immediate_operands: }
3642     { 5 } % setlanguage
3643     {
3644       \mode_if_horizontal:TF
3645       { \_unravel_scan_int: }
3646       { \_unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3647     }
3648     { 6 } % pdfliteral
3649     {
3650       \_unravel_scan_keyword:nF { dDiIrrReEcCtT }
3651       { \_unravel_scan_keyword:n { pPaAgGeE } }
3652       \_unravel_scan_pdf_ext_toks:
3653     }
3654     { 7 } % pdfobj
3655     {
3656       \_unravel_scan_keyword:nTF
3657       { rReEsSeErRvVeEoObBjJnNuUmM }
3658       { \_unravel_skip_optional_space: }
3659       {
3660         \_unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3661         { \_unravel_scan_int: }
3662         \_unravel_scan_keyword:nT { sStTrReEaAmM }
3663         {
3664           \_unravel_scan_keyword:nT { aAtTtTrR }
3665           { \_unravel_scan_pdf_ext_toks: }
3666         }
3667         \_unravel_scan_keyword:n { fFiIlLeE }
3668         \_unravel_scan_pdf_ext_toks:
3669       }
3670     }
3671     { 8 } % pdfrefobj
```

```

3672     { \_unravel_scan_int: }
3673 { 9 } % pdfxform
3674 {
3675     \_unravel_scan_keyword:nT { aAtTtTrR }
3676     { \_unravel_scan_pdf_ext_toks: }
3677     \_unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3678     { \_unravel_scan_pdf_ext_toks: }
3679     \_unravel_scan_int:
3680 }
3681 { 10 } % pdfrefxform
3682 { \_unravel_scan_int: }
3683 { 11 } % pdfximage
3684 { \_unravel_scan_image: }
3685 { 12 } % pdfrefximage
3686 { \_unravel_scan_int: }
3687 { 13 } % pdfannot
3688 {
3689     \_unravel_scan_keyword:nTF
3690     { rReEsSeErRvVeEoObBjJnNuUmM }
3691     { \_unravel_scan_optional_space: }
3692     {
3693         \_unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3694         { \_unravel_scan_int: }
3695         \_unravel_scan_alt_rule:
3696         \_unravel_scan_pdf_ext_toks:
3697     }
3698 }
3699 { 14 } % pdfstartlink
3700 {
3701     \mode_if_vertical:TF
3702     { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3703     {
3704         \_unravel_scan_rule_attr:
3705         \_unravel_scan_action:
3706     }
3707 }
3708 { 15 } % pdfendlink
3709 {
3710     \mode_if_vertical:T
3711     { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3712 }
3713 { 16 } % pdfoutline
3714 {
3715     \_unravel_scan_keyword:nT { aAtTtTrR }
3716     { \_unravel_scan_pdf_ext_toks: }
3717     \_unravel_scan_action:
3718     \_unravel_scan_keyword:nT { cCoOuUnNtT }
3719     { \_unravel_scan_int: }
3720     \_unravel_scan_pdf_ext_toks:
3721 }
3722 { 17 } % pdfdest
3723 { \_unravel_scan_pdfdest_operands: }
3724 { 18 } % pdfthread
3725 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }

```

```

3726 { 19 } % pdfstartthread
3727 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3728 { 20 } % pdfendthread
3729 { }
3730 { 21 } % pdfsavepos
3731 { }
3732 { 22 } % pdfinfo
3733 { \_unravel_scan_pdf_ext_toks: }
3734 { 23 } % pdfcatalog
3735 {
3736     \_unravel_scan_pdf_ext_toks:
3737     \_unravel_scan_keyword:n { o0pPeEnNaAcCtTiIoOnN }
3738     { \_unravel_scan_action: }
3739 }
3740 { 24 } % pdfnames
3741 { \_unravel_scan_pdf_ext_toks: }
3742 { 25 } % pdffontattr
3743 {
3744     \_unravel_scan_font_ident:
3745     \_unravel_scan_pdf_ext_toks:
3746 }
3747 { 26 } % pdfincludechars
3748 {
3749     \_unravel_scan_font_ident:
3750     \_unravel_scan_pdf_ext_toks:
3751 }
3752 { 27 } % pdfmapfile
3753 { \_unravel_scan_pdf_ext_toks: }
3754 { 28 } % pdfmapline
3755 { \_unravel_scan_pdf_ext_toks: }
3756 { 29 } % pdftrailer
3757 { \_unravel_scan_pdf_ext_toks: }
3758 { 30 } % pdfresettimer
3759 { }
3760 { 31 } % pdffontexpand
3761 {
3762     \_unravel_scan_font_ident:
3763     \_unravel_scan_optional_equals:
3764     \_unravel_scan_int:
3765     \_unravel_scan_int:
3766     \_unravel_scan_int:
3767     \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3768     { \_unravel_skip_optional_space: }
3769 }
3770 { 32 } % pdfsetrandomseed
3771 { \_unravel_scan_int: }
3772 { 33 } % pdfsnaprefpoint
3773 { }
3774 { 34 } % pdfsnapy
3775 { \_unravel_scan_normal_glue: }
3776 { 35 } % pdfsnapycomp
3777 { \_unravel_scan_int: }
3778 { 36 } % pdfglyphtounicode
3779 {

```

```

3780         \_unravel_scan_pdf_ext_toks:
3781         \_unravel_scan_pdf_ext_toks:
3782     }
3783     { 37 } % pdfcolorstack
3784     { \_unravel_scan_pdfcolorstack_operands: }
3785     { 38 } % pdfsetmatrix
3786     { \_unravel_scan_pdf_ext_toks: }
3787     { 39 } % pdfsave
3788     { }
3789     { 40 } % pdfrestore
3790     { }
3791     { 41 } % pdfnombuiltintounicode
3792     { \_unravel_scan_font_ident: }
3793 }
3794 { } % no other cases.
3795 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```

3796 \cs_new_protected:Npn \_unravel_scan_pdfcolorstack_operands:
3797 {
3798     \_unravel_scan_int:
3799     \_unravel_scan_keyword:nF { sSeEtT }
3800     {
3801         \_unravel_scan_keyword:nF { pPuUsShH }
3802         {
3803             \_unravel_scan_keyword:nF { pPoOpP }
3804             {
3805                 \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
3806                 {
3807                     \_unravel_error:nnnnn { color-stack-action-missing }
3808                     { } { } { } { }
3809                 }
3810             }
3811         }
3812     }
3813 }

```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```

3814 \cs_new_protected:Npn \_unravel_scan_rule_attr:
3815 {
3816     \_unravel_scan_alt_rule:
3817     \_unravel_scan_keyword:nT { aAtTtTrR }
3818     { \_unravel_scan_pdf_ext_toks: }
3819 }

```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```

3820 \cs_new_protected:Npn \_unravel_scan_action:
3821 {
3822     \_unravel_scan_keyword:nTF { uUsSeErR }

```

```

3823     { \_unravel_scan_pdf_ext_toks: }
3824     {
3825     \_unravel_scan_keyword:nF { gGoOtToO }
3826     {
3827     \_unravel_scan_keyword:nF { tThHrReEaAdD }
3828     { \_unravel_error:nnnn { action-type-missing } { } { } { } { } }
3829     }
3830     }
3831 \_unravel_scan_keyword:nT { fFiIlLeE }
3832 { \_unravel_scan_pdf_ext_toks: }
3833 \_unravel_scan_keyword:nTF { pPaAgGeE }
3834 {
3835 \_unravel_scan_int:
3836 \_unravel_scan_pdf_ext_toks:
3837 }
3838 {
3839 \_unravel_scan_keyword:nTF { nNaAmMeE }
3840 { \_unravel_scan_pdf_ext_toks: }
3841 {
3842 \_unravel_scan_keyword:nTF { nNuUmM }
3843 { \_unravel_scan_int: }
3844 { \_unravel_error:nnnn { identifier-type-missing } { } { } { } { } }
3845 }
3846 }
3847 \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3848 { \_unravel_skip_optional_space: }
3849 {
3850 \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3851 { \_unravel_skip_optional_space: }
3852 }
3853 }

```

(End definition for _unravel_scan_action:.)

_unravel_scan_image: Used by \pdfximage.

```

3854 \cs_new_protected:Npn \_unravel_scan_image:
3855 {
3856 \_unravel_scan_rule_attr:
3857 \_unravel_scan_keyword:nTF { nNaAmMeEdD }
3858 { \_unravel_scan_pdf_ext_toks: }
3859 {
3860 \_unravel_scan_keyword:nT { pPaAgGeE }
3861 { \_unravel_scan_int: }
3862 }
3863 \_unravel_scan_keyword:nT { cCoOlLoOrRsSpPaAcCeE }
3864 { \_unravel_scan_int: }
3865 \_unravel_scan_pdf_ext_toks:
3866 }

```

(End definition for _unravel_scan_image:.)

_unravel_scan_immediate_operands:

```

3867 \cs_new_protected:Npn \_unravel_scan_immediate_operands:
3868 {
3869 \_unravel_get_x_next:

```

```

3870 \__unravel_set_cmd:
3871 \int_compare:nNnTF
3872 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
3873 {
3874   \int_compare:nNnTF
3875   \l__unravel_head_char_int < { 3 } % openout, write, closeout
3876   { \__unravel_scan_immediate_operands_aux: }
3877   {
3878     \int_case:nnF \l__unravel_head_char_int
3879     {
3880       { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
3881       { 9 } { \__unravel_scan_extension_operands_aux: } % pdfxform
3882       { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
3883     }
3884     { \__unravel_scan_immediate_operands_bad: }
3885   }
3886 }
3887 { \__unravel_scan_immediate_operands_bad: }
3888 }
3889 \cs_new_protected:Npn \__unravel_scan_immediate_operands_aux:
3890 {
3891   \__unravel_prev_input:V \l__unravel_head_tl
3892   \__unravel_scan_extension_operands:
3893 }
3894 \cs_new_protected:Npn \__unravel_scan_immediate_operands_bad:
3895 {
3896   \__unravel_back_input:
3897   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3898   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3899   \__unravel_prev_input_gpush:
3900 }
3901

```

(End definition for __unravel_scan_immediate_operands:.)

__unravel_scan_pdfdest_operands:

```

3902 \cs_new_protected:Npn \__unravel_scan_pdfdest_operands:
3903 {
3904   \__unravel_scan_keyword:nTF { nNuUmM }
3905   { \__unravel_scan_int: }
3906   {
3907     \__unravel_scan_keyword:nTF { nNaAmMeE }
3908     { \__unravel_scan_pdf_ext_toks: }
3909     { \__unravel_error:nmnn { identifier-type-missing } { } { } { } { } }
3910   }
3911   \__unravel_scan_keyword:nTF { xXyYzZ }
3912   {
3913     \__unravel_scan_keyword:nT { zZoOoOmM }
3914     { \__unravel_scan_int: }
3915   }
3916   {
3917     \__unravel_scan_keyword:nF { fFiItTbBhH }
3918     {
3919       \__unravel_scan_keyword:nF { fFiItTbBvV }

```

```

3920     {
3921         \_unravel_scan_keyword:nF { fFiItTbB }
3922         {
3923             \_unravel_scan_keyword:nF { fFiItThHhH }
3924             {
3925                 \_unravel_scan_keyword:nF { fFiItTvV }
3926                 {
3927                     \_unravel_scan_keyword:nTF
3928                     { fFiItTrR }
3929                     {
3930                         \_unravel_skip_optional_space:
3931                         \_unravel_scan_alt_rule:
3932                         \use_none:n
3933                     }
3934                     {
3935                         \_unravel_scan_keyword:nF
3936                         { fFiItT }
3937                         {
3938                             \_unravel_error:nnnn { destination-type-missing }
3939                             { } { } { } { }
3940                         }
3941                     }
3942                 }
3943             }
3944         }
3945     }
3946 }
3947 }
3948 \_unravel_skip_optional_space:
3949 }

```

(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`.

```

3950 \cs_set_protected:Npn \_unravel_tmp:w
3951 {
3952     \_unravel_prev_input_gpush:
3953     \_unravel_prefixed_command:
3954 }
3955 \int_step_inline:nnnn
3956 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
3957 { 1 }
3958 { \_unravel_tex_use:n { max_command } }
3959 { \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.

```

3960 \cs_new_protected:Npn \_unravel_prefixed_command:
3961 {

```



```

3962 \int_while_do:nNnn
3963 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
3964 {
3965   \__unravel_prev_input:V \l__unravel_head_tl
3966   \__unravel_get_x_non_relax:
3967   \__unravel_set_cmd:
3968   \int_compare:nNnF \l__unravel_head_cmd_int
3969   > { \__unravel_tex_use:n { max_non_prefixed_command } }
3970   {
3971     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3972     \__unravel_error:nxxxx { erroneous-prefixes }
3973     { \tl_to_str:N \l__unravel_tmpa_tl }
3974     { \tl_to_str:N \l__unravel_head_tl }
3975     { } { }
3976     \__unravel_back_input:
3977     \__unravel_omit_after_assignment:w
3978   }
3979 }
3980 % ^^A todo: Discard non-\global prefixes if they are irrelevant
3981 % ^^A todo: Adjust for the setting of \globaldefs
3982 \cs_if_exist_use:cF
3983 { \__unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3984 {
3985   \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
3986   \__unravel_omit_after_assignment:w
3987 }
3988 \__unravel_after_assignment:
3989 }

```

(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
3990 \cs_new_protected:Npn \__unravel_after_assignment:
3991 {
3992   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3993   \gtl_gclear:N \g__unravel_after_assignment_gtl
3994 }
3995 \cs_new_protected:Npn \__unravel_omit_after_assignment:w
3996 #1 \__unravel_after_assignment: { }

```

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

```
\__unravel_prefixed_new:nn
```

```

3997 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3998 {
3999   \cs_new_protected:cpn
4000   { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
4001 }

```

(End definition for `__unravel_prefixed_new:nn`.)

_unravel_assign_token:n

```
4002 \cs_new_protected:Npn \_unravel_assign_token:n #1
4003 {
4004   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4005   #1
4006   \tl_use:N \l__unravel_head_tl \scan_stop:
4007   \_unravel_print_assigned_token:
4008 }
```

(End definition for _unravel_assign_token:n.)

_unravel_assign_register:

```
4009 \cs_new_protected:Npn \_unravel_assign_register:
4010 {
4011   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4012   \tl_use:N \l__unravel_head_tl \scan_stop:
4013   \_unravel_print_assigned_register:
4014 }
```

(End definition for _unravel_assign_register:.)

_unravel_assign_value:nn

```
4015 \cs_new_protected:Npn \_unravel_assign_value:nn #1#2
4016 {
4017   \tl_if_empty:nF {#1}
4018   {
4019     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4020     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4021     #1
4022     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4023   }
4024   \_unravel_prev_input:V \l__unravel_head_tl
4025   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4026   \_unravel_scan_optional_equals:
4027   #2
4028   \_unravel_assign_register:
4029 }
```

(End definition for _unravel_assign_value:nn.)

_unravel_assign_toks:

```
4030 \_unravel_prefixed_new:nn { toks_register } % 71
4031 {
4032   \int_compare:nNnT \l__unravel_head_char_int = 0
4033   { % \toks
4034     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4035     \_unravel_print_action:
4036     \_unravel_scan_int:
4037     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4038   }
4039   \_unravel_assign_toks:
4040 }
4041 \_unravel_prefixed_new:nn { assign_toks } % 72
4042 { \_unravel_assign_toks: }
```

```

4043 \cs_new_protected:Npn \__unravel_assign_toks:
4044 {
4045   \__unravel_prev_input_silent:V \l__unravel_head_tl
4046   \__unravel_print_action:
4047   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4048   \__unravel_scan_optional_equals:
4049   \__unravel_get_x_non_relax:
4050   \__unravel_set_cmd:
4051   \int_compare:nNnTF
4052     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
4053   {
4054     \__unravel_prev_input:V \l__unravel_head_tl
4055     \int_compare:nNnT \l__unravel_head_char_int = 0
4056     { \__unravel_scan_int: }
4057   }
4058   {
4059     \int_compare:nNnTF
4060       \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
4061       { \__unravel_prev_input:V \l__unravel_head_tl }
4062       {
4063         \__unravel_back_input:
4064         \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4065       }
4066   }
4067   \__unravel_assign_register:
4068 }

```

(End definition for __unravel_assign_toks.)

```

4069 \__unravel_prefixed_new:nn { assign_int } % 73
4070 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4071 \__unravel_prefixed_new:nn { assign_dimen } % 74
4072 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4073 \__unravel_prefixed_new:nn { assign_glue } % 75
4074 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
4075 \__unravel_prefixed_new:nn { assign_mu_glue } % 76
4076 { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
4077 \__unravel_prefixed_new:nn { assign_font_dimen } % 77
4078 {
4079   \__unravel_assign_value:nn
4080   { \__unravel_scan_int: \__unravel_scan_font_ident: }
4081   { \__unravel_scan_normal_dimen: }
4082 }
4083 \__unravel_prefixed_new:nn { assign_font_int } % 78
4084 {
4085   \__unravel_assign_value:nn
4086   { \__unravel_scan_font_int: } { \__unravel_scan_int: }
4087 }
4088 \__unravel_prefixed_new:nn { set_aux } % 79
4089 { % prevdepth = 1, spacefactor = 102
4090   \int_compare:nNnTF \l__unravel_head_char_int = 1
4091     { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4092     { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4093 }
4094 \__unravel_prefixed_new:nn { set_prev_graf } % 80

```

```

4095 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4096 \_unravel_prefixed_new:nn { set_page_dimen } % 81
4097 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4098 \_unravel_prefixed_new:nn { set_page_int } % 82
4099 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4100 \_unravel_prefixed_new:nn { set_box_dimen } % 83
4101 {
4102   \_unravel_assign_value:nn
4103   { \_unravel_scan_int: } { \_unravel_scan_normal_dimen: }
4104 }
4105 \_unravel_prefixed_new:nn { set_shape } % 84
4106 {
4107   \_unravel_assign_value:nn { \_unravel_scan_int: }
4108   {
4109     \prg_replicate:nn
4110     {
4111       \tl_if_head_eq_meaning:VNT
4112       \l_unravel_defined_tl \tex_parshape:D { 2 * }
4113       \tl_tail:N \l_unravel_defined_tl
4114     }
4115     { \_unravel_scan_int: }
4116   }
4117 }
4118 \_unravel_prefixed_new:nn { def_code } % 85
4119 {
4120   \_unravel_assign_value:nn
4121   { \_unravel_scan_int: } { \_unravel_scan_int: }
4122 }
4123 \_unravel_prefixed_new:nn { def_family } % 86
4124 {
4125   \_unravel_assign_value:nn
4126   { \_unravel_scan_int: } { \_unravel_scan_font_ident: }
4127 }
4128 \_unravel_prefixed_new:nn { set_font } % 87
4129 {
4130   \_unravel_prev_input_gpop:N \l_unravel_tmpa_tl
4131   \tl_put_left:NV \l_unravel_head_tl \l_unravel_tmpa_tl
4132   \tl_use:N \l_unravel_head_tl \scan_stop:
4133   \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
4134   \_unravel_print_action:
4135 }
4136 \_unravel_prefixed_new:nn { def_font } % 88
4137 {
4138   \_unravel_prev_input_silent:V \l_unravel_head_tl
4139   \_unravel_set_action_text:x { \tl_to_str:N \l_unravel_head_tl }
4140   \_unravel_scan_r_token:
4141   \_unravel_print_action:x
4142   { \g_unravel_action_text_str \tl_to_str:N \l_unravel_defined_tl }
4143   \_unravel_scan_optional_equals:
4144   \_unravel_scan_file_name:
4145   \bool_gset_true:N \g_unravel_name_in_progress_bool
4146   \_unravel_scan_keyword:nTF { aAtT }
4147   { \_unravel_scan_normal_dimen: }

```

```

4148     {
4149       \__unravel_scan_keyword:nT { sScCaAlLeEdD }
4150       { \__unravel_scan_int: }
4151     }
4152     \bool_gset_false:N \g__unravel_name_in_progress_bool
4153     \__unravel_assign_token:n { }
4154   }

```

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

let, futurelet

```

4155 \__unravel_prefixed_new:nn { let } % 94
4156 {
4157   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4158   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4159   { % |let|
4160     \__unravel_scan_r_token:
4161     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4162     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4163     \__unravel_get_next:
4164     \bool_while_do:nn
4165     { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4166     { \__unravel_get_next: }
4167     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4168     { \__unravel_get_next: }
4169     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4170     { \__unravel_get_next: }
4171   }
4172   { % |futurelet|
4173     \__unravel_scan_r_token:
4174     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4175     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4176     \__unravel_get_next:
4177     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4178     \__unravel_get_next:
4179     \__unravel_back_input:
4180     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4181     \__unravel_back_input:
4182   }
4183   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4184   \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4185   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4186   \__unravel_exp_args:Nx \use:n
4187   {
4188     \exp_not:V \l__unravel_head_tl
4189     \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4190   }
4191   \__unravel_print_assigned_token:
4192 }
4193 \__unravel_prefixed_new:nn { shorthand_def } % 95
4194 {
4195   \__unravel_prev_input_silent:V \l__unravel_head_tl
4196   \tl_set:Nx \l__unravel_prev_action_tl
4197   { \tl_to_str:N \l__unravel_head_tl }

```

```

4198   \_unravel_scan_r_token:
4199   \_unravel_print_action:x
4200   { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4201   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4202   \_unravel_scan_optional_equals:
4203   \_unravel_scan_int:
4204   \_unravel_assign_token:n { }
4205 }

```

`_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.

```

4206 \_unravel_prefixed_new:nn { read_to_cs } % 96
4207 {
4208   \_unravel_prev_input_silent:V \l__unravel_head_tl
4209   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4210   \_unravel_scan_int:
4211   \_unravel_scan_to:
4212   \_unravel_scan_r_token:
4213   \_unravel_prev_input_get:N \l__unravel_tmpa_tl
4214   \_unravel_read_to_cs_safe:fTF
4215   { \_unravel_tl_first_int:N \l__unravel_tmpa_tl }
4216   { \_unravel_assign_token:n { } }
4217   {
4218     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4219     \_unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4220   }
4221 }
4222 \prg_new_conditional:Npnn \_unravel_read_to_cs_safe:n #1 { TF }
4223 {
4224   \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
4225   { \prg_return_true: }
4226   {
4227     \int_compare:nNnTF {#1} < { 0 }
4228     { \prg_return_false: }
4229     {
4230       \int_compare:nNnTF {#1} > { 15 }
4231       { \prg_return_false: }
4232       {
4233         \tex_ifeof:D #1 \exp_stop_f:
4234         \prg_return_false:
4235         \else:
4236         \prg_return_true:
4237         \fi:
4238       }
4239     }
4240   }
4241 }
4242 \cs_generate_variant:Nn \_unravel_read_to_cs_safe:nTF { f }

```

(End definition for `_unravel_read_to_cs_safe:nTF`.)

```

4243 \__unravel_prefixed_new:nn { def } % 97
4244 {
4245   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4246   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4247   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4248   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4249   \int_compare:nNnTF \l__unravel_head_char_int < 2
4250     { % def/gdef
4251       \__unravel_scan_r_token:
4252       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4253       \__unravel_scan_toks:NN \c_true_bool \c_false_bool
4254     }
4255     { % edef/xdef
4256       \__unravel_scan_r_token:
4257       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4258       \__unravel_scan_toks:NN \c_true_bool \c_true_bool
4259     }
4260   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4261   \__unravel_prev_input:V \l__unravel_head_tl
4262   \__unravel_assign_token:n
4263   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4264 }

```

\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.

```

4265 \__unravel_prefixed_new:nn { set_box } % 98
4266 {
4267   \__unravel_prev_input:V \l__unravel_head_tl
4268   \__unravel_scan_int:
4269   \__unravel_scan_optional_equals:
4270   \bool_if:NTF \g__unravel_set_box_allowed_bool
4271     { \__unravel_do_box:N \c_false_bool }
4272     {
4273       \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
4274       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4275       \__unravel_omit_after_assignment:w
4276     }
4277 }

```

\hyphenation and \patterns

```

4278 \__unravel_prefixed_new:nn { hyph_data } % 99
4279 {
4280   \__unravel_prev_input:V \l__unravel_head_tl
4281   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4282   \__unravel_assign_token:n { }
4283 }

```

```

4284 \__unravel_prefixed_new:nn { set_interaction } % 100
4285 {
4286   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4287   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4288   \tl_use:N \l__unravel_head_tl \scan_stop:
4289   \__unravel_print_assignment:x { \tl_to_str:N \l__unravel_head_tl }
4290 }

```

```

4291 \__unravel_prefixed_new:nn { letterspace_font } % 101
4292 {
4293   \__unravel_prev_input_silent:V \l__unravel_head_tl
4294   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4295   \__unravel_scan_r_token:
4296   \__unravel_print_action:x
4297   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4298   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4299   \__unravel_scan_optional_equals:
4300   \__unravel_scan_font_ident:
4301   \__unravel_scan_int:
4302   \__unravel_assign_token:n { }
4303 }
4304 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4305 {
4306   \__unravel_prev_input_silent:V \l__unravel_head_tl
4307   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4308   \__unravel_scan_r_token:
4309   \__unravel_print_action:x
4310   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4311   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4312   \__unravel_scan_optional_equals:
4313   \__unravel_scan_font_ident:
4314   \__unravel_assign_token:n { }
4315 }

```

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

```

4316 \__unravel_prefixed_new:nn { register } % 89
4317 { \__unravel_do_register:N 0 }
4318 \__unravel_prefixed_new:nn { advance } % 90
4319 { \__unravel_do_operation:N 1 }
4320 \__unravel_prefixed_new:nn { multiply } % 91
4321 { \__unravel_do_operation:N 2 }
4322 \__unravel_prefixed_new:nn { divide } % 92
4323 { \__unravel_do_operation:N 3 }

```

`__unravel_do_operation:N`

`__unravel_do_operation_fail:w`

```

4324 \cs_new_protected:Npn \__unravel_do_operation:N #1
4325 {
4326   \__unravel_prev_input_silent:V \l__unravel_head_tl
4327   \__unravel_print_action:
4328   \__unravel_get_x_next:
4329   \__unravel_set_cmd:
4330   \int_compare:nNnTF
4331     \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4332     {
4333       \int_compare:nNnTF
4334         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4335         { \__unravel_do_register:N #1 }
4336         { \__unravel_do_operation_fail:w }
4337     }
4338     {
4339       \int_compare:nNnTF

```



```

4340         \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
4341     { \__unravel_do_operation_fail:w }
4342     {
4343         \__unravel_prev_input:V \l__unravel_head_tl
4344         \exp_args:NNf \__unravel_do_register_set:Nn #1
4345         {
4346             \int_eval:n
4347             {
4348                 \l__unravel_head_cmd_int
4349                 - \__unravel_tex_use:n { assign_toks }
4350             }
4351         }
4352     }
4353 }
4354 }
4355 \cs_new_protected:Npn \__unravel_do_operation_fail:w
4356 {
4357     \__unravel_error:nnnnn { after-advance } { } { } { } { }
4358     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4359     \__unravel_omit_after_assignment:w
4360 }

```

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

__unravel_do_register:N

__unravel_do_register_aux:Nn

```

4361 \cs_new_protected:Npn \__unravel_do_register:N #1
4362 {
4363     \exp_args:NNV \__unravel_do_register_aux:Nn #1
4364     \l__unravel_head_char_int
4365 }
4366 \cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
4367 {
4368     \int_compare:nNnTF { \tl_tail:n {#2} } = 0
4369     {
4370         \__unravel_prev_input_gpush:N \l__unravel_head_tl
4371         \__unravel_print_assignment:
4372         \__unravel_scan_int:
4373         \__unravel_prev_input_gpop:N \l__unravel_head_tl
4374         \__unravel_prev_input_silent:V \l__unravel_head_tl
4375     }
4376     {
4377         \__unravel_prev_input_silent:V \l__unravel_head_tl
4378         \__unravel_print_assignment:
4379     }
4380     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4381     \exp_args:NNf \__unravel_do_register_set:Nn #1
4382     { \int_eval:n { #2 / 1 000 000 } }
4383 }

```

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

__unravel_do_register_set:Nn

```

4384 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
4385 {
4386     \int_compare:nNnTF {#1} = 0

```

```

4387 { % truly register command
4388   \__unravel_scan_optional_equals:
4389 }
4390 { % \advance, \multiply, \divide
4391   \__unravel_scan_keyword:nF { bByY }
4392   { \__unravel_prev_input_silent:n { by } }
4393 }
4394 \int_compare:nNnTF {#1} < 2
4395 {
4396   \int_case:nnF {#2}
4397   {
4398     { 1 } { \__unravel_scan_int:          } % count
4399     { 2 } { \__unravel_scan_normal_dimen: } % dim
4400     { 3 } { \__unravel_scan_normal_glue:  } % glue
4401     { 4 } { \__unravel_scan_mu_glue:      } % muglue
4402   }
4403   { \__unravel_error:nxxxx { internal } { do-reg=#2 } { } { } { } }
4404 }
4405 { \__unravel_scan_int: }
4406 \__unravel_assign_register:
4407 }

```

(End definition for `__unravel_do_register_set:Nn`.)

The following is used for instance when making accents.

```

4408 \cs_new_protected:Npn \__unravel_do_assignments:
4409 {
4410   \__unravel_get_x_non_relax:
4411   \__unravel_set_cmd:
4412   \int_compare:nNnT
4413     \l__unravel_head_cmd_int
4414     > { \__unravel_tex_use:n { max_non_prefixed_command } }
4415   {
4416     \bool_gset_false:N \g__unravel_set_box_allowed_bool
4417     \__unravel_prev_input_gpush:
4418     \__unravel_prefixed_command:
4419     \bool_gset_true:N \g__unravel_set_box_allowed_bool
4420     \__unravel_do_assignments:
4421   }
4422 }

```

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` `\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.

```

4423 \__unravel_new_tex_expandable:nm { undefined_cs } % 103
4424 { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }

\__unravel_expandafter:
  \__unravel_unless: 4425 \__unravel_new_tex_expandable:nm { expand_after } % 104
\__unravel_unless_bad: 4426 {
4427   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4428   { \__unravel_expandafter: } { \__unravel_unless: }
4429 }
4430 \cs_new_protected:Npn \__unravel_expandafter:
4431 {
4432   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4433   \__unravel_get_next:
4434   \gtl_concat:NNN \l__unravel_head_gtl
4435   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4436   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4437   \__unravel_print_expansion:x { \gtl_to_str:N \l__unravel_head_gtl }
4438   \__unravel_get_next:
4439   \__unravel_token_if_expandable:NTF \l__unravel_head_token
4440   { \__unravel_expand_do:N \prg_do_nothing: }
4441   { \__unravel_back_input: }
4442   \__unravel_prev_input_gpop:N \l__unravel_head_gtl
4443   \__unravel_set_action_text:x
4444   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4445   \gtl_pop_left:N \l__unravel_head_gtl
4446   \__unravel_back_input:
4447   \__unravel_print_expansion:
4448 }
4449 \cs_new_protected:Npn \__unravel_unless:
4450 {
4451   \__unravel_get_token:
4452   \int_compare:nNnTF

```

```

4453 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4454 {
4455   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4456   { \__unravel_unless_bad: }
4457   {
4458     \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4459     % \int_add:Nn \l__unravel_head_char_int { 32 }
4460     \__unravel_expand_nonmacro:
4461   }
4462 }
4463 { \__unravel_unless_bad: }
4464 }
4465 \cs_new_protected:Npn \__unravel_unless_bad:
4466 {
4467   \__unravel_error:nmnnn { bad-unless } { } { } { } { }
4468   \__unravel_back_input:
4469 }

```

(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`.

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.

```

4470 \__unravel_new_tex_expandable:mn { no_expand } % 105
4471 {
4472   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4473   { \__unravel_noexpand:N }
4474   { \__unravel_pdfprimitive: }
4475 }
4476 \cs_new_protected:Npn \__unravel_noexpand:N #1
4477 {
4478   \__unravel_get_token:
4479   \cs_if_eq:NNTF #1 \prg_do_nothing:
4480   {
4481     \tl_if_empty:NTF \l__unravel_head_tl
4482     { \__unravel_back_input: }
4483     {
4484       \exp_after:wN \__unravel_token_if_definable:NNTF \l__unravel_head_tl
4485       { \__unravel_noexpand_after: }
4486       { \__unravel_back_input: }

```

```

4487     }
4488   }
4489   {
4490     \__unravel_back_input:
4491     \__unravel_get_next:
4492     \__unravel_token_if_expandable:NT \l__unravel_head_token
4493     { \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax: }
4494   }
4495 }
4496 \cs_new_protected:Npn \__unravel_noexpand_after:
4497 {
4498   \group_begin:
4499   \__unravel_set_escapechar:n { 92 }
4500   \exp_args:NNc
4501   \group_end:
4502   \__unravel_noexpand_after:N
4503   { notexpanded: \exp_after:wN \token_to_str:N \l__unravel_head_tl }
4504 }
4505 \cs_new_protected:Npn \__unravel_noexpand_after:N #1
4506 {
4507   \cs_gset_eq:NN #1 \__unravel_special_relax:
4508   \__unravel_back_input:n {#1}
4509 }
4510 \cs_new_protected:Npn \__unravel_pdfprimitive:
4511 { \__unravel_not_implemented:n { pdfprimitive } }

```

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

```

\__unravel_endinput:
\__unravel_scantokens: 4512 \__unravel_new_tex_expandable:nn { input } % 106
\__unravel_input: 4513 {
4514   \int_case:nnF \l__unravel_head_char_int
4515   {
4516     { 1 } { \__unravel_endinput: } % \endinput
4517     { 2 } { \__unravel_scantokens: } % \scantokens
4518   }
4519   { % 0=\input
4520     \bool_if:NTF \g__unravel_name_in_progress_bool
4521     { \__unravel_insert_relax: } { \__unravel_input: }
4522   }
4523 }
4524 \cs_new_protected:Npn \__unravel_endinput:
4525 {
4526   \group_begin:
4527   \msg_warning:nn { unravel } { endinput-ignored }
4528   \group_end:
4529   \__unravel_print_expansion:
4530 }
4531 \cs_new_protected:Npn \__unravel_scantokens:
4532 {
4533   \__unravel_prev_input_gpush:
4534   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4535   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4536   \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl

```

```

4537     \__unravel_back_input:V \l__unravel_head_tl
4538     \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_tmpa_tl }
4539   }
4540 \cs_new_protected:Npn \__unravel_input:
4541 {
4542   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4543   \__unravel_scan_file_name:
4544   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4545   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4546   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4547   \__unravel_back_input:V \l__unravel_tmpa_tl
4548   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
4549 }

```

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

__unravel_csname_loop:

```

4550 \__unravel_new_tex_expandable:nn { cs_name } % 109
4551 {
4552   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4553   \__unravel_print_expansion:
4554   \__unravel_csname_loop:
4555   \__unravel_prev_input_silent:V \l__unravel_head_tl
4556   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4557   \__unravel_back_input_tl_o:
4558 }
4559 \cs_new_protected:Npn \__unravel_csname_loop:
4560 {
4561   \__unravel_get_x_next:
4562   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4563   {
4564     \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4565     {
4566       \__unravel_back_input:
4567       \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4568       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4569     }
4570   }
4571   {
4572     \__unravel_prev_input_silent:x
4573     { \__unravel_token_to_char:N \l__unravel_head_token }
4574     \__unravel_csname_loop:
4575   }
4576 }

```

(End definition for __unravel_csname_loop:.)

```

4577 \__unravel_new_tex_expandable:nn { convert } % 110
4578 {
4579   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4580   \__unravel_print_expansion:
4581   \int_case:nn \l__unravel_head_char_int
4582   {
4583     0     \__unravel_scan_int:
4584     1     \__unravel_scan_int:

```

```

4585     2     \__unravel_convert_string:
4586     3     \__unravel_convert_meaning:
4587     4     \__unravel_scan_font_ident:
4588     8     \__unravel_scan_font_ident:
4589     9     \__unravel_scan_font_ident:
4590     { 10 } \__unravel_scan_font_ident:
4591     { 11 } \__unravel_scan_int:
4592     { 12 } \__unravel_scan_int:
4593     { 13 } \__unravel_scan_pdf_ext_toks:
4594     { 14 } \__unravel_scan_pdf_ext_toks:
4595     { 15 } \__unravel_scan_int:
4596     { 16 } \__unravel_scan_int:
4597     { 17 } \__unravel_scan_pdfstrcmp:
4598     { 18 } \__unravel_scan_pdfcolorstackinit:
4599     { 19 } \__unravel_scan_pdf_ext_toks:
4600     { 20 } \__unravel_scan_pdf_ext_toks:
4601     { 22 } \__unravel_scan_pdf_ext_toks:
4602     { 23 } \__unravel_scan_pdf_ext_toks:
4603     { 24 }
4604     {
4605         \__unravel_scan_keyword:n { fFillLeE }
4606         \__unravel_scan_pdf_ext_toks:
4607     }
4608     { 25 } \__unravel_scan_pdffiledump:
4609     { 26 } \__unravel_scan_pdfmatch:
4610     { 27 } \__unravel_scan_int:
4611     { 28 } \__unravel_scan_int:
4612     { 30 } \__unravel_scan_int:
4613     { 31 } \__unravel_scan_pdfximagebbox:
4614     { 33 } \__unravel_scan_directlua:
4615     { 34 } \__unravel_scan_pdf_ext_toks:
4616     { 35 } \__unravel_scan_pdf_ext_toks:
4617     { 40 }
4618     {
4619         \__unravel_scan_int:
4620         \__unravel_prev_input_silent:n { ~ }
4621         \__unravel_scan_int:
4622     }
4623 }
4624 \__unravel_prev_input_gpop:N \l__unravel_head_tl
4625 \__unravel_back_input_tl_o:
4626 }
4627 \cs_new_protected:Npn \__unravel_convert_string:
4628 {
4629     \__unravel_get_next:
4630     \tl_if_empty:NTF \l__unravel_head_tl
4631     { \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_head_gtl } }
4632     { \__unravel_prev_input:V \l__unravel_head_tl }
4633 }
4634 \cs_new_protected:Npn \__unravel_convert_meaning:
4635 {
4636     \__unravel_get_next:
4637     \tl_if_empty:NTF \l__unravel_head_tl
4638     { \__unravel_prev_input:n { \l__unravel_head_token } }

```

```

4639     { \__unravel_prev_input:V \l__unravel_head_tl }
4640   }
4641 \cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
4642   {
4643     \__unravel_scan_toks_to_str:
4644     \__unravel_scan_toks_to_str:
4645   }
4646 \cs_new_protected:Npn \__unravel_scan_pdfximagebbox:
4647   { \__unravel_scan_int: \__unravel_scan_int: }
4648 \cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:
4649   {
4650     \__unravel_scan_keyword:nTF { pPaAgGeE }
4651     { \bool_set_true:N \l__unravel_tmpa_bool }
4652     { \bool_set_false:N \l__unravel_tmpb_bool }
4653     \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4654     { \__unravel_scan_keyword:n { pPaAgGeE } }
4655     \__unravel_scan_toks_to_str:
4656   }
4657 \cs_new_protected:Npn \__unravel_scan_pdffiledump:
4658   {
4659     \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
4660     \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
4661     \__unravel_scan_pdf_ext_toks:
4662   }
4663 \cs_new_protected:Npn \__unravel_scan_pdfmatch:
4664   {
4665     \__unravel_scan_keyword:n { iIcCaAsSeE }
4666     \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
4667     { \__unravel_scan_int: }
4668     \__unravel_scan_pdf_ext_toks:
4669     \__unravel_scan_pdf_ext_toks:
4670   }
4671 \sys_if_engine luatex:T
4672   {
4673     \cs_new_protected:Npn \__unravel_scan_directlua:
4674     {
4675       \__unravel_get_x_non_relax:
4676       \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
4677       { \__unravel_back_input: }
4678       {
4679         \__unravel_scan_int:
4680         \__unravel_get_x_non_relax:
4681       }
4682       \__unravel_scan_pdf_ext_toks:
4683     }
4684   }

```

__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.

```

4685 \__unravel_new_tex_expandable:mn { the } % 111
4686   { \__unravel_get_the:N }
4687 \cs_new_protected:Npn \__unravel_get_the:N #1
4688   {
4689     \__unravel_prev_input_gpush:N \l__unravel_head_tl

```



```

4690 \__unravel_print_expansion:
4691 \int_if_odd:nTF \l__unravel_head_char_int
4692 { % \unexpanded, \detokenize
4693   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4694   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4695   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4696 }
4697 { % \the
4698   \__unravel_get_x_next:
4699   \__unravel_scan_something_internal:n { 5 }
4700   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4701   \__unravel_set_action_text:x
4702   {
4703     \tl_head:N \l__unravel_head_tl
4704     => \tl_tail:N \l__unravel_head_tl
4705   }
4706   \tl_set:Nx \l__unravel_head_tl
4707   { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4708 }
4709 \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
4710 {
4711   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4712   \__unravel_prev_input:V \l__unravel_head_tl
4713 }
4714 {
4715   \cs_if_eq:NNTF #1 \__unravel_get_token_x:
4716   {
4717     \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
4718     \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
4719   }
4720   {
4721     \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4722     \__unravel_back_input:V \l__unravel_tmpa_tl
4723   }
4724   \__unravel_print_expansion:
4725 }
4726 #1
4727 }

```

(End definition for __unravel_get_the:N.)

```

4728 \__unravel_new_tex_expandable:nn { top_bot_mark } % 112
4729 { \__unravel_back_input_tl_o: }
4730 \__unravel_new_tex_expandable:nn { end_template } % 117
4731 {
4732   \__unravel_not_implemented:n { end-template } { } { } { }
4733   \__unravel_back_input_tl_o:
4734 }

```

2.14.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w
4735 \cs_new_protected:Npn \__unravel_pass_text:
4736 {

```

```

4737 \_unravel_input_if_empty:TF
4738 { \_unravel_pass_text_empty: }
4739 {
4740 \_unravel_input_get:N \l__unravel_tmpb_gtl
4741 \if_true:
4742 \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
4743 \exp_after:wN \_unravel_pass_text_done:w
4744 \fi:
4745 \_unravel_input_gpop:N \l__unravel_tmpb_gtl
4746 \exp_after:wN \_unravel_pass_text:
4747 \else:
4748 \use:c { fi: }
4749 \int_set:Nn \l__unravel_if_nesting_int { 1 }
4750 \_unravel_input_gpop:N \l__unravel_tmpb_gtl
4751 \exp_after:wN \_unravel_pass_text_nested:
4752 \fi:
4753 }
4754 }
4755 \cs_new_protected:Npn \_unravel_pass_text_done:w
4756 {
4757 \_unravel_get_next:
4758 \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4759 \else:
4760 }

```

(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:`.

```

4761 \cs_new_protected:Npn \_unravel_pass_text_nested:
4762 {
4763 \_unravel_input_if_empty:TF
4764 { \_unravel_pass_text_empty: }
4765 {
4766 \_unravel_input_get:N \l__unravel_tmpb_gtl
4767 \if_true:
4768 \if_true:
4769 \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4770 \int_decr:N \l__unravel_if_nesting_int
4771 \use_none:nnnnn
4772 \fi:
4773 \use_none:nnn
4774 \fi:

```

```

4775     \int_incr:N \l__unravel_if_nesting_int
4776     \fi:
4777     \__unravel_input_gpop:N \l__unravel_unused_gt1
4778     \int_compare:nNnTF \l__unravel_if_nesting_int = 0
4779       { \__unravel_pass_text: }
4780       { \__unravel_pass_text_nested: }
4781   }
4782 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

4783 \cs_new_protected:Npn \__unravel_pass_text_empty:
4784 {
4785   \__unravel_error:nnnnn { runaway-if } { } { } { } { }
4786   \__unravel_exit_hard:w
4787 }

```

(End definition for __unravel_pass_text_empty:.)

__unravel_cond_push:

__unravel_cond_pop:

```

4788 \cs_new_protected:Npn \__unravel_cond_push:
4789 {
4790   \tl_gput_left:Nx \g__unravel_if_limit_tl
4791   { { \int_use:N \g__unravel_if_limit_int } }
4792   \int_gincr:N \g__unravel_if_depth_int
4793   \int_gzero:N \g__unravel_if_limit_int
4794 }
4795 \cs_new_protected:Npn \__unravel_cond_pop:
4796 {
4797   \int_gset:Nn \g__unravel_if_limit_int
4798   { \tl_head:N \g__unravel_if_limit_tl }
4799   \tl_gset:Nx \g__unravel_if_limit_tl
4800   { \tl_tail:N \g__unravel_if_limit_tl }
4801   \int_gdecr:N \g__unravel_if_depth_int
4802 }

```

(End definition for __unravel_cond_push: and __unravel_cond_pop:.)

__unravel_change_if_limit:nn

```

4803 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
4804 {
4805   \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4806   { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4807   {
4808     \tl_clear:N \l__unravel_tmpa_tl
4809     \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
4810     {
4811       \tl_put_right:Nx \l__unravel_tmpa_tl
4812       { { \tl_head:N \g__unravel_if_limit_tl } }
4813       \tl_gset:Nx \g__unravel_if_limit_tl
4814       { \tl_tail:N \g__unravel_if_limit_tl }
4815     }
4816     \tl_gset:Nx \g__unravel_if_limit_tl
4817     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }

```

```

4818     }
4819 }

```

(End definition for _unravel_change_if_limit:nn.)

```

4820 \_unravel_new_tex_expandable:nn { if_test } % 107
4821 {
4822   \_unravel_cond_push:
4823   \exp_args:NV \_unravel_cond_aux:n \g\_unravel_if_depth_int
4824 }

```

_unravel_cond_aux:nn

```

4825 \cs_new_protected:Npn \_unravel_cond_aux:n #1
4826 {
4827   \int_case:nnF \l\_unravel_head_char_int
4828   {
4829     { 0 } { \_unravel_test_two_chars:nn { 0 } {#1} } % if
4830     { 1 } { \_unravel_test_two_chars:nn { 1 } {#1} } % ifcat
4831     { 12 } { \_unravel_test_ifx:n {#1} }
4832     { 16 } { \_unravel_test_case:n {#1} }
4833     { 21 } { \_unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4834   }
4835   {
4836     \_unravel_prev_input_gpush:N \l\_unravel_head_tl
4837     \_unravel_print_expansion:
4838     \int_case:nn \l\_unravel_head_char_int
4839     {
4840       { 2 } % ifnum
4841       { \_unravel_test_two_vals:N \_unravel_scan_int: }
4842       { 3 } % ifdim
4843       { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4844       { 4 } { \_unravel_scan_int: } % ifodd
4845       % { 5 } { } % ifvmode
4846       % { 6 } { } % ifhmode
4847       % { 7 } { } % ifmmode
4848       % { 8 } { } % ifinner
4849       { 9 } { \_unravel_scan_int: } % ifvoid
4850       { 10 } { \_unravel_scan_int: } % ifhbox
4851       { 11 } { \_unravel_scan_int: } % ifvbox
4852       { 13 } { \_unravel_scan_int: } % ifeof
4853       % { 14 } { } % iftrue
4854       % { 15 } { } % iffalse
4855       { 17 } { \_unravel_test_ifdefined: } % ifdefined
4856       { 18 } { \_unravel_test_ifcsname: } % ifcsname
4857       { 19 } % iffontchar
4858       { \_unravel_scan_font_ident: \_unravel_scan_int: }
4859       % { 20 } { } % ifincsname % ^^A todo: something?
4860       { 22 } % ifpdfabsnum
4861       { \_unravel_test_two_vals:N \_unravel_scan_int: }
4862       { 23 } % ifpdfabsdim
4863       { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4864     }
4865     \_unravel_prev_input_gpop:N \l\_unravel_head_tl
4866     \_unravel_set_action_text:x { \tl_to_str:N \l\_unravel_head_tl }
4867     \l\_unravel_head_tl \scan_stop:

```

```

4868         \exp_after:wN \_unravel_cond_true:n
4869     \else:
4870         \exp_after:wN \_unravel_cond_false:n
4871     \fi:
4872     {#1}
4873 }
4874 }

```

(End definition for _unravel_cond_aux:nn.)

_unravel_cond_true:n

```

4875 \cs_new_protected:Npn \_unravel_cond_true:n #1
4876 {
4877     \_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4878     \_unravel_print_expansion:x { \g_unravel_action_text_str = true }
4879 }

```

(End definition for _unravel_cond_true:n.)

_unravel_cond_false:n

_unravel_cond_false_loop:n
 _unravel_cond_false_common:

```

4880 \cs_new_protected:Npn \_unravel_cond_false:n #1
4881 {
4882     \_unravel_cond_false_loop:n {#1}
4883     \_unravel_cond_false_common:
4884     \_unravel_print_expansion:x
4885     {
4886         \g_unravel_action_text_str = false ~
4887         => ~ skipped ~ to ~ \iow_char:N\fi
4888     }
4889 }
4890 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4891 {
4892     \_unravel_pass_text:
4893     \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
4894     {
4895         \token_if_eq_meaning:NNT \l_unravel_head_token \or:
4896         {
4897             \_unravel_error:nmnn { extra-or } { } { } { } { }
4898             \_unravel_cond_false_loop:n {#1}
4899         }
4900     }
4901     {
4902         \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
4903         { \_unravel_cond_pop: }
4904         \_unravel_cond_false_loop:n {#1}
4905     }
4906 }
4907 \cs_new_protected:Npn \_unravel_cond_false_common:
4908 {
4909     \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
4910     { \_unravel_cond_pop: }
4911     { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
4912 }

```

(End definition for _unravel_cond_false:n, _unravel_cond_false_loop:n, and _unravel_cond_false_common:.)

_unravel_test_two_vals:N

```

4913 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
4914 {
4915   #1
4916   \_unravel_get_x_non_blank:
4917   \_unravel_tl_if_in:ooTF { < = > } \l__unravel_head_tl { }
4918   {
4919     \_unravel_error:nmnnn { missing-equals } { } { } { } { }
4920     \_unravel_back_input:
4921     \tl_set:Nn \l__unravel_head_tl { = }
4922   }
4923   \_unravel_prev_input:V \l__unravel_head_tl
4924   #1
4925 }

```

(End definition for _unravel_test_two_vals:N.)

_unravel_test_two_chars:nn

_unravel_test_two_chars_get:n
_unravel_test_two_chars_gtl:N

```

4926 \cs_new_protected:Npn \_unravel_test_two_chars:nn #1
4927 {
4928   \_unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4929   \_unravel_print_expansion:
4930   \_unravel_test_two_chars_get:n {#1}
4931   \_unravel_test_two_chars_get:n {#1}
4932   \_unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
4933   \_unravel_set_action_text:x { \gtl_to_str:N \l__unravel_head_gtl }
4934   \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
4935   \gtl_pop_left:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4936   \_unravel_test_two_chars_gtl:N \l__unravel_tmpb_gtl
4937   \_unravel_test_two_chars_gtl:N \l__unravel_head_gtl
4938   \l__unravel_head_tl \scan_stop:
4939   \exp_after:wN \_unravel_cond_true:n
4940   \else:
4941   \exp_after:wN \_unravel_cond_false:n
4942   \fi:
4943 }
4944 \cs_new_protected:Npn \_unravel_test_two_chars_get:n #1
4945 {
4946   \_unravel_get_x_next:
4947   \int_compare:nNnT {#1} = 0
4948   {
4949     \gtl_if_head_is_N_type:NF \l__unravel_head_gtl
4950     { \gtl_set:Nx \l__unravel_head_gtl { \gtl_to_str:N \l__unravel_head_gtl } }
4951   }
4952   \_unravel_prev_input_gtl:N \l__unravel_head_gtl
4953   \_unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
4954 }
4955 \cs_new_protected:Npn \_unravel_test_two_chars_gtl:N #1
4956 {
4957   \tl_put_right:Nx \l__unravel_head_tl
4958   {

```

```

4959     \gtl_if_head_is_group_begin:NTF #1 { \c_group_begin_token }
4960     {
4961         \gtl_if_head_is_group_end:NTF #1 { \c_group_end_token }
4962         { \exp_not:N \exp_not:N \gtl_head_do:NN #1 \exp_not:N }
4963     }
4964 }
4965 }

```

(End definition for `__unravel_test_two_chars:nn`, `__unravel_test_two_chars_get:n`, and `__unravel_test_two_chars_gtl:N`.)

```

\__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`.

```

4966 \cs_new_protected:Npn \__unravel_test_ifx:n #1
4967 {
4968     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4969     \__unravel_print_expansion:
4970     \__unravel_get_next:
4971     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4972     \cs_set_eq:NN \l__unravel_tmpb_token \l__unravel_head_token
4973     \__unravel_get_next:
4974     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4975     \__unravel_set_action_text:x
4976     {
4977         Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4978         \__unravel_test_ifx_str:NN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
4979         \__unravel_test_ifx_str:NN \l__unravel_head_token \l__unravel_head_gtl
4980     }
4981     \__unravel_test_ifx_aux:NNN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
4982     \__unravel_test_ifx_aux:w
4983     \exp_after:wN \__unravel_cond_true:n
4984     \else:
4985     \exp_after:wN \__unravel_cond_false:n
4986     \fi:
4987     {#1}
4988 }
4989 \cs_new:Npn \__unravel_test_ifx_str:NN #1#2
4990 {
4991     \token_if_eq_meaning:NNT #1 \__unravel_special_relax:
4992     { \iow_char:N \notexpanded: }
4993     \gtl_to_str:N #2
4994 }
4995 \cs_new_protected:Npn \__unravel_test_ifx_aux:NNN #1#2#3
4996 {
4997     \token_if_eq_meaning:NNTF #1 \__unravel_special_relax:
4998     {
4999         \gtl_head_do:NN #2 \__unravel_token_if_expandable:NTF

```

```

5000         { #3 #1 } { \gtl_head_do:NN #2 #3 }
5001     }
5002     { \gtl_head_do:NN #2 #3 }
5003 }
5004 \cs_new:Npn \__unravel_test_ifx_aux:w
5005 {
5006     \__unravel_test_ifx_aux:NNN \l__unravel_head_token \l__unravel_head_gtl
5007     \l__unravel_tmpa_tl
5008 }

```

(End definition for __unravel_test_ifx:n and others.)

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn
5009 \cs_new_protected:Npn \__unravel_test_case:n #1
5010 {
5011     \__unravel_prev_input_gpush:N \l__unravel_head_tl
5012     \__unravel_print_expansion:
5013     \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
5014     \__unravel_scan_int:
5015     \__unravel_prev_input_get:N \l__unravel_head_tl
5016     \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
5017     % ^^A does text_case_aux use prev_input_seq?
5018     \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
5019     \__unravel_prev_input_gpop:N \l__unravel_head_tl
5020     \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
5021 }
5022 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
5023 {
5024     \int_compare:nNnTF {#1} = 0
5025     { \__unravel_change_if_limit:nn { 4 } {#2} }
5026     {
5027         \__unravel_pass_text:
5028         \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
5029         {
5030             \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
5031             {
5032                 \exp_args:Nf \__unravel_test_case_aux:nn
5033                 { \int_eval:n { #1 - 1 } } {#2}
5034             }
5035             { \__unravel_cond_false_common: }
5036         }
5037         {
5038             \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
5039             { \__unravel_cond_pop: }
5040             \__unravel_test_case_aux:nn {#1} {#2}
5041         }
5042     }
5043 }

```

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

```

\__unravel_test_ifdefined:
5044 \cs_new_protected:Npn \__unravel_test_ifdefined:
5045 {
5046     \__unravel_input_if_empty:TF

```



```

5047 { \__unravel_pass_text_empty: }
5048 {
5049   \__unravel_input_gpop:N \l__unravel_tmpb_gtl
5050   \__unravel_set_action_text:x
5051   {
5052     Conditional:~ \tl_to_str:N \l__unravel_head_tl
5053     \gtl_to_str:N \l__unravel_tmpb_gtl
5054   }
5055   \__unravel_prev_input:x
5056   {
5057     \gtl_if_tl:NTF \l__unravel_tmpb_gtl
5058     { \gtl_head:N \l__unravel_tmpb_gtl }
5059     { \gtl_to_str:N \l__unravel_tmpb_gtl }
5060   }
5061 }
5062 }

```

(End definition for __unravel_test_ifdefined:.)

__unravel_test_ifcsname:

```

5063 \cs_new_protected:Npn \__unravel_test_ifcsname:
5064 {
5065   \__unravel_csname_loop:
5066   \__unravel_prev_input:V \l__unravel_head_tl
5067 }

```

(End definition for __unravel_test_ifcsname:.)

```

5068 \__unravel_new_tex_expandable:nm { fi_or_else } % 108
5069 {
5070   \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
5071   {
5072     \int_compare:nNnTF \g__unravel_if_limit_int = 0
5073     {
5074       \int_compare:nNnTF \g__unravel_if_depth_int = 0
5075       { \__unravel_error:nmnnn { extra-fi-or-else } { } { } { } { } }
5076       { \__unravel_insert_relax: }
5077     }
5078     { \__unravel_error:nmnnn { extra-fi-or-else } { } { } { } { } }
5079   }
5080   {
5081     \__unravel_set_action_text:
5082     \int_compare:nNnF \l__unravel_head_char_int = 2
5083     {
5084       \__unravel_if_or_else_loop:
5085       \__unravel_set_action_text:x
5086       {
5087         \g__unravel_action_text_str \c_space_tl
5088         => ~ skipped ~ to ~ \tl_to_str:N \l__unravel_head_tl
5089       }
5090     }
5091     % ^^A todo: in the terminal output the token itself is missing.
5092     \__unravel_print_expansion:
5093     \__unravel_cond_pop:
5094   }

```

```

5095 }
5096 \cs_new_protected:Npn \__unravel_fi_or_else_loop:
5097 {
5098   \int_compare:nNnF \l__unravel_head_char_int = 2
5099   {
5100     \__unravel_pass_text:
5101     \__unravel_set_cmd:
5102     \__unravel_fi_or_else_loop:
5103   }
5104 }

```

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.

`\l__unravel_print_tl`

```

5105 \tl_new:N \l__unravel_print_tl
5106 \sys_if_engine luatex:TF
5107 {
5108   \cs_new_protected:Npx \__unravel_print_normalize_null:
5109   {
5110     \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
5111     { \char_generate:nn { 0 } { 12 } }
5112     { \tl_to_str:n { ^^@ } }
5113   }
5114 }
5115 { \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`

```

5116 \cs_new_protected:Npn \__unravel_print:n #1
5117 {
5118   \tl_set:Nn \l__unravel_print_tl {#1}
5119   \__unravel_print_normalize_null:
5120   \__unravel_exp_args:Nx \iow_term:n { \l__unravel_print_tl }
5121 }
5122 \cs_new_protected:Npn \__unravel_print:x
5123 { \__unravel_exp_args:Nx \__unravel_print:n }
5124 \cs_new_protected:Npn \__unravel_log:n #1
5125 {
5126   \tl_set:Nn \l__unravel_print_tl {#1}
5127   \__unravel_print_normalize_null:
5128   \__unravel_exp_args:Nx \iow_log:n { \l__unravel_print_tl }
5129 }

```

(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`.

```

5130 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
5131 {
5132   \int_compare:nNnF \g__unravel_online_int < 0
5133   {
5134     \int_compare:nNnTF \g__unravel_online_int = 0
5135     { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_log:n }
5136     { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_print:n }
5137   }
5138 }

```

(End definition for `__unravel_print_message:nn`.)

`__unravel_set_action_text:x`

```

5139 \cs_new_protected:Npn \__unravel_set_action_text:x #1
5140 {
5141   \group_begin:
5142   \__unravel_set_escapechar:n { 92 }
5143   \str_gset:Nx \g__unravel_action_text_str {#1}
5144   \group_end:
5145 }

```

(End definition for `__unravel_set_action_text:x`.)

`__unravel_set_action_text:`

```

5146 \cs_new_protected:Npn \__unravel_set_action_text:
5147 {
5148   \__unravel_set_action_text:x
5149   {
5150     \tl_to_str:N \l__unravel_head_tl
5151     \tl_if_single_token:VT \l__unravel_head_tl
5152     { = ~ \token_to_meaning:N \l__unravel_head_token }
5153   }
5154 }

```

(End definition for `__unravel_set_action_text:.`)

`__unravel_print_state:`

```

5155 \cs_new_protected:Npn \__unravel_print_state:
5156 {
5157   \group_begin:
5158   \__unravel_set_escapechar:n { 92 }
5159   \tl_use:N \g__unravel_before_print_state_tl
5160   \int_compare:nNnT \g__unravel_online_int > 0
5161   {
5162     \__unravel_print_state_output:
5163     \__unravel_print_state_prev:
5164     \__unravel_print_state_input:
5165   }
5166   \group_end:
5167 }

```

(End definition for `__unravel_print_state:.`)

```

\__unravel_print_state_output: Unless empty, print #1 with each line starting with <|~. The \__unravel_str_-
\__unravel_print_state_output:n truncate_left:nn function trims #1 if needed, to fit in a maximum of \g__unravel_-
max_output_int characters.
5168 \cs_new_protected:Npn \__unravel_print_state_output:
5169 {
5170   \__unravel_exp_args:Nx \__unravel_print_state_output:n
5171   { \gtl_to_str:N \g__unravel_output_gtl }
5172 }
5173 \cs_new_protected:Npn \__unravel_print_state_output:n #1
5174 {
5175   \tl_if_empty:nF {#1}
5176   {
5177     \__unravel_print_message:nn { <| ~ }
5178     { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
5179   }
5180 }

```

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

```

\__unravel_print_state_prev: Never trim ##1.
5181 \cs_new_protected:Npn \__unravel_print_state_prev:
5182 {
5183   \seq_set_map:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
5184   { \__unravel_to_str:n {##1} }
5185   \seq_remove_all:Nn \l__unravel_tmpa_seq { }
5186   \seq_if_empty:NTF \l__unravel_tmpa_seq
5187   { \__unravel_print_message:nn { || ~ } { } }
5188   {
5189     \seq_map_inline:Nn \l__unravel_tmpa_seq
5190     {
5191       \__unravel_print_message:nn { || ~ } {##1}
5192     }
5193   }
5194 }

```

(End definition for __unravel_print_state_prev:.)

```

\__unravel_print_state_input: Print #1 with each line starting with |>~. The \__unravel_str_truncate_right:nn
\__unravel_print_state_input:n function trims #1 if needed, to fit in a maximum of \g__unravel_max_input_int characters.
5195 \cs_new_protected:Npn \__unravel_print_state_input:
5196 {
5197   \__unravel_exp_args:Nx \__unravel_print_state_input:n
5198   { \__unravel_input_to_str: }
5199 }
5200 \cs_new_protected:Npn \__unravel_print_state_input:n #1
5201 {
5202   \__unravel_print_message:nn { |> ~ }
5203   { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
5204 }

```

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

__unravel_print_meaning:

```
5205 \cs_new_protected:Npn \__unravel_print_meaning:
5206 {
5207   \__unravel_input_if_empty:TF
5208     { \__unravel_print_message:nn { } { Empty-input! } }
5209     {
5210       \__unravel_input_get:N \l__unravel_tmpb_gtl
5211       \__unravel_print_message:nn { }
5212       {
5213         \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
5214         = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
5215       }
5216     }
5217 }
```

(End definition for __unravel_print_meaning:.)

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

```
\__unravel_print_assignment: 5218 \cs_new_protected:Npn \__unravel_print_action:
  \__unravel_print_assignment:x 5219 { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
  \__unravel_print_expansion: 5220 \cs_new_protected:Npn \__unravel_print_action:x #1
  \__unravel_print_expansion:x 5221 {
  \__unravel_print_action_aux:N 5222   \__unravel_set_action_text:x {#1}
  5223   \__unravel_print_action:
  5224 }
  \cs_new_protected:Npn \__unravel_print_assignment: 5225 \cs_new_protected:Npn \__unravel_print_assignment:
  5226 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
  \cs_new_protected:Npn \__unravel_print_assignment:x #1 5227 \cs_new_protected:Npn \__unravel_print_assignment:x #1
  5228 {
  5229   \__unravel_set_action_text:x {#1}
  5230   \__unravel_print_assignment:
  5231 }
  \cs_new_protected:Npn \__unravel_print_expansion: 5232 \cs_new_protected:Npn \__unravel_print_expansion:
  5233 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
  \cs_new_protected:Npn \__unravel_print_expansion:x #1 5234 \cs_new_protected:Npn \__unravel_print_expansion:x #1
  5235 {
  5236   \__unravel_set_action_text:x {#1}
  5237   \__unravel_print_expansion:
  5238 }
  \cs_new_protected:Npn \__unravel_print_action_aux:N #1 5239 \cs_new_protected:Npn \__unravel_print_action_aux:N #1
  5240 {
  5241   \int_gdecr:N \g__unravel_nonstop_int
  5242   \int_gincr:N \g__unravel_step_int
  5243   \bool_if:NT #1
  5244   {
  5245     \__unravel_print:x
  5246     {
  5247       [====
  5248       \bool_if:NT \g__unravel_number_steps_bool
  5249       { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
  5250       =====]~
  5251       \int_compare:nNnTF
  5252       { \str_count:N \g__unravel_action_text_str }
```

```

5253         > { \g__unravel_max_action_int }
5254         {
5255             \str_range:Nnn \g__unravel_action_text_str
5256                 { 1 } { \g__unravel_max_action_int - 3 } ...
5257         }
5258         { \g__unravel_action_text_str }
5259     }
5260     \__unravel_print_state:
5261     \__unravel_prompt:
5262 }
5263 }

```

(End definition for __unravel_print_action: and others.)

__unravel_print_assigned_token:
__unravel_print_assigned_register:

```

5264 \cs_new_protected:Npn \__unravel_print_assigned_token:
5265 {
5266     \__unravel_after_assignment: % ^^A todo: simplify
5267     \__unravel_print_assignment:x
5268     {
5269         Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5270             = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
5271     }
5272     \__unravel_omit_after_assignment:w
5273 }
5274 \cs_new_protected:Npn \__unravel_print_assigned_register:
5275 {
5276     \__unravel_after_assignment: % ^^A todo: simplify
5277     \__unravel_exp_args:Nx \__unravel_print_assignment:x
5278     {
5279         \exp_not:n
5280         {
5281             Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5282                 \tl_if_single:NT \l__unravel_defined_tl
5283                 { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
5284         }
5285         = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
5286     }
5287     \__unravel_omit_after_assignment:w
5288 }

```

(End definition for __unravel_print_assigned_token: and __unravel_print_assigned_register:.)

__unravel_print_welcome: Welcome message.

```

5289 \cs_new_protected:Npn \__unravel_print_welcome:
5290 {
5291     \__unravel_print_message:nn { }
5292     {
5293         \bool_if:NTF \g__unravel_welcome_message_bool
5294         {
5295             \\\
5296             =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5297             \iow_indent:n
5298             {
5299                 "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\

```

```

5300             "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\
5301             "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
5302             Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
5303         }
5304     }
5305     { [====~Start~=====] }
5306 }
5307 \__unravel_print_state:
5308 \__unravel_prompt:
5309 }

```

(End definition for __unravel_print_welcome:.)

__unravel_print_outcome: Final message.

```

5310 \cs_new_protected:Npn \__unravel_print_outcome:
5311 { \__unravel_print_message:nn { } { [====~End~=====] } }

```

(End definition for __unravel_print_outcome:.)

2.15.2 Prompt

__unravel_ior_str_get:NN
__unravel_ior_str_get:Nc

```

5312 \cs_new_protected:Npn \__unravel_ior_str_get:NN #1#2
5313 { \tex_readline:D #1 to #2 }
5314 \cs_generate_variant:Nn \__unravel_ior_str_get:NN { Nc }

```

(End definition for __unravel_ior_str_get:NN.)

__unravel_prompt:

```

5315 \cs_new_protected:Npn \__unravel_prompt:
5316 {
5317     \int_compare:nNnF \g__unravel_nonstop_int > 0
5318     {
5319         \group_begin:
5320         \__unravel_set_escapechar:n { -1 }
5321         \int_set:Nn \tex_endlinechar:D { -1 }
5322         \tl_use:N \g__unravel_before_prompt_tl
5323         \__unravel_prompt_aux:
5324         \group_end:
5325     }
5326 }
5327 \cs_new_protected:Npn \__unravel_prompt_aux:
5328 {
5329     \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
5330     {
5331         \bool_if:NTF \g__unravel_explicit_prompt_bool
5332         { \__unravel_ior_str_get:Nc \c__unravel_prompt_ior }
5333         { \__unravel_ior_str_get:Nc \c__unravel_noprompt_ior }
5334         { Your~input }
5335         \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5336     }
5337 }
5338 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5339 {
5340     \tl_if_empty:nF {#1}

```

```

5341 {
5342   \__unravel_exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
5343   {
5344     { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
5345     { q }
5346     {
5347       \int_gset:Nn \g__unravel_online_int { -1 }
5348       \int_gzero:N \g__unravel_nonstop_int
5349     }
5350     { x }
5351     {
5352       \group_end:
5353       \__unravel_exit_hard:w
5354     }
5355     { X } { \tex_batchmode:D \tex_end:D }
5356     { s } { \__unravel_prompt_scan_int:nn {#1}
5357       \__unravel_prompt_silent_steps:n }
5358     { o } { \__unravel_prompt_scan_int:nn {#1}
5359       { \int_gset:Nn \g__unravel_online_int } }
5360     { C }
5361     {
5362       \__unravel_exp_args:Nx \use:n
5363       {
5364         \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5365         { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5366       }
5367       \tl_gput_left:Nn \g__unravel_tmpc_tl
5368       { \tl_gclear:N \g__unravel_tmpc_tl }
5369       \group_insert_after:N \g__unravel_tmpc_tl
5370       \group_insert_after:N \__unravel_prompt:
5371     }
5372     { | } { \__unravel_prompt_scan_int:nn {#1}
5373       \__unravel_prompt_vert:n }
5374     { u } { \__unravel_prompt_until:n {#1} }
5375     { a } { \__unravel_prompt_all: }
5376   }
5377   { \__unravel_prompt_help: }
5378 }
5379 }
5380 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
5381 {
5382   \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
5383   \l__unravel_prompt_tmpa_int =
5384   \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5385   \use_ii:nn #1 \scan_stop:
5386 }
5387 \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5388 {
5389   #2 \l__unravel_prompt_tmpa_int
5390   \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
5391 }
5392 \cs_new_protected:Npn \__unravel_prompt_help:
5393 {
5394   \__unravel_print:n { "m":~meaning-of~first~token }

```



```

5395 \__unravel_print:n { "a":~print~state~again,~without~truncating }
5396 \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
5397 \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
5398 \__unravel_print:n { "u<text>":~silent~steps~until~the~input~starts~with~<text> }
5399 \__unravel_print:n
5400 { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
5401 \__unravel_print:n { "q":~semi-quiet~(same~as~"o-1") }
5402 \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
5403 \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
5404 \__unravel_prompt_aux:
5405 }
5406 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
5407 {
5408   \int_compare:nNnF {#1} < 0
5409   {
5410     \int_gset:Nn \g__unravel_online_int { -1 }
5411     \tl_gset:Nn \g__unravel_before_prompt_tl
5412     {
5413       \int_gset:Nn \g__unravel_online_int { 1 }
5414       \tl_gclear:N \g__unravel_before_prompt_tl
5415     }
5416     \int_gset:Nn \g__unravel_nonstop_int {#1}
5417   }
5418 }
5419 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5420 {
5421   \int_compare:nNnTF {#1} < { 0 }
5422   { \__unravel_prompt_vert:Nn > {#1} }
5423   { \__unravel_prompt_vert:Nn < {#1} }
5424 }
5425 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5426 {
5427   \int_gset:Nn \g__unravel_online_int { -1 }
5428   \tl_gset:Nf \g__unravel_before_print_state_tl
5429   {
5430     \exp_args:Nnf \exp_stop_f: \int_compare:nNnTF
5431     { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5432     #1 { \__unravel_prev_input_count: }
5433     {
5434       \int_gset:Nn \g__unravel_nonstop_int
5435       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5436     }
5437     {
5438       \int_gset:Nn \g__unravel_online_int { 1 }
5439       \tl_gclear:N \g__unravel_before_print_state_tl
5440     }
5441   }
5442 }
5443 \cs_new_protected:Npn \__unravel_prompt_all:
5444 {
5445   \tl_gset:Nx \g__unravel_tmpc_tl
5446   {
5447     \exp_not:n
5448     {

```

```

5449         \tl_gclear:N \g__unravel_tmpc_tl
5450         \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5451         \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5452         \__unravel_print_state:
5453         \int_gdecr:N \g__unravel_nonstop_int
5454         \__unravel_prompt:
5455     }
5456     \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5457     \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5458 }
5459 \group_insert_after:N \g__unravel_tmpc_tl
5460 }
5461 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5462 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for __unravel_prompt:.)

```

\__unravel_prompt_until:n
  \g__unravel_until_tl
5463 \tl_new:N \g__unravel_until_tl
5464 \cs_new_protected:Npn \__unravel_prompt_until:n #1
5465 {
5466   \tl_gset:Nx \g__unravel_until_tl { \tl_tail:n {#1} }
5467   \int_gset:Nn \g__unravel_online_int { -1 }
5468   \tl_gset:Nn \g__unravel_before_print_state_tl
5469   {
5470     \__unravel_input_get_left:N \l__unravel_tmpa_tl
5471     \__unravel_exp_args:Nx \use:n
5472     {
5473       \exp_not:N \tl_if_in:nnTF
5474       { \exp_not:N \__unravel:nn \tl_to_str:N \l__unravel_tmpa_tl }
5475       { \exp_not:N \__unravel:nn \tl_to_str:N \g__unravel_until_tl }
5476     }
5477     {
5478       \int_gset:Nn \g__unravel_online_int { 1 }
5479       \tl_gclear:N \g__unravel_before_print_state_tl
5480     }
5481     {
5482       \int_gset:Nn \g__unravel_nonstop_int
5483       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5484     }
5485   }
5486 }

```

(End definition for __unravel_prompt_until:n and \g__unravel_until_tl.)

2.15.3 Errors

```

\__unravel_not_implemented:n
5487 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5488 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End definition for __unravel_not_implemented:n.)

`__unravel_error:nnnnn` 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.

```

5489 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5490 {
5491   \group_begin:
5492   \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5493   \group_end:
5494 }
5495 \cs_new_protected:Npn \__unravel_error:nxxxx #1#2#3#4#5
5496 {
5497   \group_begin:
5498   \msg_error:nxxxxx { unravel } {#1} {#2} {#3} {#4} {#5}
5499   \group_end:
5500 }

```

(End definition for __unravel_error:nnnnn.)

`__unravel_tex_msg_new:nnn` This stores a `TeX` error message.

```

5501 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5502 {
5503   \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
5504   \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
5505 }

```

(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.

```

5506 \cs_new_protected:Npn \__unravel_tex_error:nn #1#2
5507 {
5508   \group_begin:
5509   \msg_error:nxxxx { unravel } { tex-error }
5510   { \tl_to_str:n {#2} }
5511   { \use:c { __unravel_tex_msg_error_#1: } }
5512   { \use:c { __unravel_tex_msg_help_#1: } }
5513   \group_end:
5514 }
5515 \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.

```

5516 \cs_new_protected:Npn \__unravel_tex_fatal_error:nn #1#2
5517 {
5518   \__unravel_error:nxxxxx { tex-fatal }
5519   { \tl_to_str:n {#2} }
5520   { \use:c { __unravel_tex_msg_error_#1: } }
5521   { \use:c { __unravel_tex_msg_help_#1: } }
5522   { }
5523 }
5524 \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.

```
5525 \keys_define:nn { unravel/defaults }
5526 {
5527   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5528   internal-debug  .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5529   max-action      .int_gset:N  = \g__unravel_default_max_action_int ,
5530   max-output      .int_gset:N  = \g__unravel_default_max_output_int ,
5531   max-input       .int_gset:N  = \g__unravel_default_max_input_int ,
5532   number-steps    .bool_gset:N = \g__unravel_default_number_steps_bool ,
5533   online          .int_gset:N  = \g__unravel_default_online_int ,
5534   trace-assigns   .bool_gset:N = \g__unravel_default_trace_assign_bool ,
5535   trace-expansion .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
5536   trace-other     .bool_gset:N = \g__unravel_default_trace_other_bool ,
5537   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5538 }
5539 \keys_define:nn { unravel }
5540 {
5541   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5542   internal-debug  .bool_gset:N = \g__unravel_internal_debug_bool ,
5543   max-action      .int_gset:N  = \g__unravel_max_action_int ,
5544   max-output      .int_gset:N  = \g__unravel_max_output_int ,
5545   max-input       .int_gset:N  = \g__unravel_max_input_int ,
5546   number-steps    .bool_gset:N = \g__unravel_number_steps_bool ,
5547   online          .int_gset:N  = \g__unravel_online_int ,
5548   trace-assigns   .bool_gset:N = \g__unravel_trace_assigns_bool ,
5549   trace-expansion .bool_gset:N = \g__unravel_trace_expansion_bool ,
5550   trace-other     .bool_gset:N = \g__unravel_trace_other_bool ,
5551   welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,
5552 }
```

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.

```
5553 \tl_map_inline:nn { { /defaults } { } }
5554 {
5555   \keys_define:nn { unravel #1 }
5556   {
5557     machine .meta:nn =
5558     { unravel #1 }
5559     {
5560       explicit-prompt = false ,
5561       internal-debug  = false ,
5562       max-action      = \c_max_int ,
5563       max-output      = \c_max_int ,
5564       max-input       = \c_max_int ,
5565       number-steps    = false ,
5566       welcome-message = false ,
5567     } ,
5568     mute .meta:nn =
```

```

5569     { unravel #1 }
5570     {
5571         trace-assigns = false ,
5572         trace-expansion = false ,
5573         trace-other = false ,
5574         welcome-message = false ,
5575         online = -1 ,
5576     }
5577 }
5578 }

```

2.17 Main command

\unravel Simply call an underlying code-level command.

```
5579 \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.

```
5580 \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.

```

5581 \cs_new_protected:Npn \unravel_setup:n #1
5582 {
5583     \keys_set:nn { unravel/defaults } {#1}
5584     \keys_set:nn { unravel } {#1}
5585 }

```

(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.

```

5586 \cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
5587 \cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
5588 \cs_new_protected:Npn \__unravel:nn #1#2
5589 {
5590     \__unravel_init_key_vars:
5591     \keys_set:nn { unravel } {#1}
5592     \__unravel_init_vars:
5593     \__unravel_input_gset:n {#2}
5594     \__unravel_print_welcome:
5595     \__unravel_main_loop:
5596     \__unravel_exit_point:
5597     \__unravel_print_outcome:
5598     \__unravel_final_test:
5599     \__unravel_exit_point:
5600 }

```

```

5601 \cs_new_protected:Npn \unravel_get:nnN #1#2#3
5602 {
5603   \unravel:nn {#1} {#2}
5604   \tl_set:Nx #3 { \gtl_left_tl:N \g__unravel_output_gtl }
5605 }

```

(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).

```

5606 \cs_new_protected:Npn \__unravel_init_key_vars:
5607 {
5608   \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bo
5609   \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5610   \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5611   \int_gset_eq:NN \g__unravel_online_int \g__unravel_default_online_int
5612   \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
5613   \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
5614   \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
5615   \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
5616   \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5617   \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5618   \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
5619   \int_gzero:N \g__unravel_nonstop_int
5620 }

```

(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.

```

5621 \cs_new_protected:Npn \__unravel_init_vars:
5622 {
5623   \seq_gclear:N \g__unravel_prev_input_seq
5624   \gtl_gclear:N \g__unravel_output_gtl
5625   \int_gzero:N \g__unravel_step_int
5626   \tl_gclear:N \g__unravel_if_limit_tl
5627   \int_gzero:N \g__unravel_if_limit_int
5628   \int_gzero:N \g__unravel_if_depth_int
5629   \gtl_gclear:N \g__unravel_after_assignment_gtl
5630   \bool_gset_true:N \g__unravel_set_box_allowed_bool
5631   \bool_gset_false:N \g__unravel_name_in_progress_bool
5632   \gtl_clear:N \l__unravel_after_group_gtl
5633 }

```

(End definition for `__unravel_init_vars:`)

`__unravel_main_loop:` Loop forever, getting the next token (with expansion) and performing the corresponding command. We use `__unravel_get_x_next_or_done:`, which is basically `__unravel_get_x_next:` but with a different behaviour when there are no more tokens: running out of tokens here is a successful exit of `\unravel`. Note that we cannot put the logic into `__unravel_main_loop:` because `__unravel_expand_do:N` suppresses the loop when a token is marked with `\notexpanded:`, and we don't want that to suppress the main loop, only the expansion loop.

```

5634 \cs_new_protected:Npn \__unravel_get_x_next_or_done:

```

```

5635 {
5636   \__unravel_input_if_empty:TF { \__unravel_exit:w } { }
5637   \__unravel_get_next:
5638   \__unravel_token_if_expandable:NT \l__unravel_head_token
5639   { \__unravel_expand_do:N \__unravel_get_x_next_or_done: }
5640 }
5641 \cs_new_protected:Npn \__unravel_main_loop:
5642 {
5643   \__unravel_get_x_next_or_done:
5644   \__unravel_set_cmd:
5645   \__unravel_do_step:
5646   \__unravel_main_loop:
5647 }

```

(End definition for __unravel_main_loop: and __unravel_get_x_next_or_done:.)

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

```

\__unravel_final_bad:
5648 \cs_new_protected:Npn \__unravel_final_test:
5649 {
5650   \bool_if:nTF
5651   {
5652     \tl_if_empty_p:N \g__unravel_if_limit_tl
5653     && \int_compare_p:nNn \g__unravel_if_limit_int = 0
5654     && \int_compare_p:nNn \g__unravel_if_depth_int = 0
5655     && \seq_if_empty_p:N \g__unravel_prev_input_seq
5656   }
5657   { \__unravel_input_if_empty:TF { } { \__unravel_final_bad: } }
5658   { \__unravel_final_bad: }
5659 }
5660 \cs_new_protected:Npn \__unravel_final_bad:
5661 {
5662   \__unravel_error:nnnnn { internal }
5663   { the-last-unravel-finished-badly } { } { } { }
5664 }

```

(End definition for __unravel_final_test: and __unravel_final_bad:.)

2.18 Messages

```

5665 \msg_new:nnn { unravel } { unknown-primitive }
5666   { Internal-error:~the-primitive-~'#1'-is-not-known. }
5667 \msg_new:nnn { unravel } { extra-fi-or-else }
5668   { Extra-fi,~or,~or-else. }
5669 \msg_new:nnn { unravel } { missing-dollar }
5670   { Missing-dollar-inserted. }
5671 \msg_new:nnn { unravel } { unknown-expandable }
5672   { Internal-error:~the-expandable-command-~'#1'-is-not-known. }
5673 \msg_new:nnn { unravel } { missing-font-id }
5674   { Missing-font-identifier.~\iow_char:N\~\nullfont~inserted. }
5675 \msg_new:nnn { unravel } { missing-rparen }
5676   { Missing-right-parenthesis~inserted~for-expression. }
5677 \msg_new:nnn { unravel } { missing-cs }
5678   { Missing-control-sequence.~\iow_char:N\~inaccessible~inserted. }
5679 \msg_new:nnn { unravel } { missing-box }
5680   { Missing-box-inserted. }

```

```

5681 \msg_new:nnn { unravel } { missing-to }
5682   { Missing-keyword-'to'-inserted. }
5683 \msg_new:nnn { unravel } { improper-leaders }
5684   { Leaders-not-followed-by-proper-glue. }
5685 \msg_new:nnn { unravel } { extra-close }
5686   { Extra-right-brace-or-\iow_char:N\endgroup. }
5687 \msg_new:nnn { unravel } { off-save }
5688   { Something-is-wrong-with-groups. }
5689 \msg_new:nnn { unravel } { hrule-bad-mode }
5690   { \iow_char\hrule-used-in-wrong-mode. }
5691 \msg_new:nnn { unravel } { invalid-mode }
5692   { Invalid-mode-for-this-command. }
5693 \msg_new:nnn { unravel } { color-stack-action-missing }
5694   { Missing-color-stack-action. }
5695 \msg_new:nnn { unravel } { action-type-missing }
5696   { Missing-action-type. }
5697 \msg_new:nnn { unravel } { identifier-type-missing }
5698   { Missing-identifier-type. }
5699 \msg_new:nnn { unravel } { destination-type-missing }
5700   { Missing-destination-type. }
5701 \msg_new:nnn { unravel } { erroneous-prefixes }
5702   { Prefixes-applied-to-non-assignment-command. }
5703 \msg_new:nnn { unravel } { improper-setbox }
5704   { \iow_char:N\setbox-while-fetching-base-of-an-accent. }
5705 \msg_new:nnn { unravel } { after-advance }
5706   {
5707     Missing-register-after-\iow_char:N\advance,~
5708     \iow_char:N\multiply,~or~\iow_char:N\divide.
5709   }
5710 \msg_new:nnn { unravel } { bad-unless }
5711   { \iow_char:N\unless-not-followed-by-conditional. }
5712 \msg_new:nnn { unravel } { runaway-if }
5713   { Runaway-\iow_char:N\if... }
5714 \msg_new:nnn { unravel } { runaway-macro-parameter }
5715   {
5716     Runaway-macro-parameter-\# #2~after \\\
5717     \iow_indent:n {#1}
5718   }
5719 \msg_new:nnn { unravel } { extra-or }
5720   { Extra-\iow_char:N\or. }
5721 \msg_new:nnn { unravel } { missing-equals }
5722   { Missing-equals-for-\iow_char:N\ifnum-or-\iow_char:N\ifdim. }
5723 \msg_new:nnn { unravel } { internal }
5724   { Internal-error:~'#1'.~\ Please-report. }
5725 \msg_new:nnn { unravel } { not-implemented }
5726   { The-following-feature-is-not-implemented:~'#1'. }
5727 \msg_new:nnn { unravel } { endinput-ignored }
5728   { The-primitive-\iow_char:N\endinput-was-ignored. }
5729 \msg_new:nnn { unravel } { missing-something }
5730   { Something-is-missing,~sorry! }
5731 \msg_new:nnn { unravel } { nested-unravel }
5732   { The-\iow_char:N\unravel-command-may-not-be-nested. }
5733 \msg_new:nnnn { unravel } { tex-error }
5734   { TeX-sees-~'#1'-and-throws-an-error:\\\ \iow_indent:n {#2} }

```



```

5735 {
5736   \tl_if_empty:nTF {#3}
5737     { TeX~provides~no~further~help~for~this~error. }
5738     { TeX's~advice~is:~\ \ \ \ \iow_indent:n {#3} }
5739 }
5740 \msg_new:nnnn { unravel } { tex-fatal }
5741 { TeX~sees~"#1"~and~throws~a~fatal~error:\ \ \ \ \iow_indent:n {#2} }
5742 {
5743   \tl_if_empty:nTF {#3}
5744     { TeX~provides~no~further~help~for~this~error. }
5745     { TeX's~advice~is:~\ \ \ \ \iow_indent:n {#3} }
5746 }
5747 \msg_new:nnn { unravel } { runaway-unravel }
5748 { Runaway~\iow_char:N\unravel }

Some error messages from TEX itself.
5749 \__unravel_tex_msg_new:nnn { incompatible-mag }
5750 {
5751   Incompatible~magnification~
5752   ( \int_to_arabic:n { \__unravel_mag: } );~
5753   the~previous~value~will~be~retained
5754 }
5755 {
5756   I~can~handle~only~one~magnification~ratio~per~job.~So~I've
5757   reverted~to~the~magnification~you~used~earlier~on~this~run.
5758 }
5759 \__unravel_tex_msg_new:nnn { illegal-mag }
5760 {
5761   Illegal~magnification~has~been~changed~to~1000~
5762   ( \int_to_arabic:n { \__unravel_mag: } )
5763 }
5764 { The~magnification~ratio~must~be~between~1~and~32768. }
5765 \__unravel_tex_msg_new:nnn { missing-number }
5766 { Missing~number,~treated~as~zero }
5767 {
5768   A~number~should~have~been~here;~I~inserted~'0'.~
5769   If~you~can't~figure~out~why~I~needed~to~see~a~number,~
5770   look~up~'weird~error'~in~the~index~to~The~TeXbook.
5771 }
5772 \__unravel_tex_msg_new:nnn { the-cannot }
5773 { You~can't~use~'\tl_to_str:N\l__unravel_head_tl'~after~\iow_char:N\the }
5774 { I'm~forgetting~what~you~said~and~using~zero~instead. }
5775 \__unravel_tex_msg_new:nnn { incompatible-units }
5776 { Incompatible~glue~units }
5777 { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }
5778 \__unravel_tex_msg_new:nnn { missing-mu }
5779 { Illegal~unit~of~measure~(mu~inserted) }
5780 {
5781   The~unit~of~measurement~in~math~glue~must~be~mu.~
5782   To~recover~gracefully~from~this~error,~it's~best~to~
5783   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5784   two~letters.~(See~Chapter~27~of~The~TeXbook.)
5785 }
5786 \__unravel_tex_msg_new:nnn { missing-pt }
5787 { Illegal~unit~of~measure~(pt~inserted) }

```

```

5788 {
5789   Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
5790   cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
5791   I'll~assume~that~you~meant~to~say~pt,~for~printer's~points.~
5792   To~recover~gracefully~from~this~error,~it's~best~to~
5793   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5794   two~letters.~(See~Chapter~27~of~The~TeXbook.)
5795 }
5796 \_unravel_tex_msg_new:nnn { missing-lbrace }
5797 { Missing~\iow_char:N\{~inserted }
5798 {
5799   A~left~brace~was~mandatory~here,~so~I've~put~one~in.~
5800   You~might~want~to~delete~and/or~insert~some~corrections~
5801   so~that~I~will~find~a~matching~right~brace~soon.~
5802   (If~you're~confused~by~all~this,~try~typing~'\iow_char:N\}'~now.)
5803 }
5804 \_unravel_tex_msg_new:nnn { extra-endcsname }
5805 { Extra~\token_to_str:c{endcsname} }
5806 { I'm~ignoring~this,~since~I~wasn't~doing~a~\token_to_str:c{csname}. }
5807 \_unravel_tex_msg_new:nnn { missing-endcsname }
5808 { Missing~\token_to_str:c{endcsname}~inserted }
5809 {
5810   The~control~sequence~marked~<to~be~read~again>~should~
5811   not~appear~between~\token_to_str:c{csname}~and~
5812   \token_to_str:c{endcsname}.
5813 }

```

Fatal T_EX error messages.

```

5814 \_unravel_tex_msg_new:nnn { cannot-read }
5815 { ***~(cannot~\iow_char:N\read~from~terminal~in~nonstop~modes) }
5816 { }
5817 \_unravel_tex_msg_new:nnn { file-error }
5818 { ***~(job~aborted,~file~error~in~nonstop~mode) }
5819 { }
5820 \_unravel_tex_msg_new:nnn { interwoven-preambles }
5821 { (interwoven~alignment~preambles~are~not~allowed) }
5822 { }

```

Restore catcodes to their original values.

```

5823 \_unravel_setup_restore:
5824 </package>

```