

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2019/11/15

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	11
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 0.2h, last revised 2019/11/15.

2.9	Paragraphs	76
2.10	Groups	78
2.11	Modes	79
2.12	Commands	81
2.12.1	Characters: from 0 to 15	81
2.12.2	Boxes: from 16 to 31	86
2.12.3	From 32 to 47	90
2.12.4	Maths: from 48 to 56	94
2.12.5	From 57 to 70	95
2.12.6	Extensions	98
2.12.7	Assignments	104
2.13	Expandable primitives	114
2.13.1	Conditionals	121
2.14	User interaction	130
2.14.1	Print	130
2.14.2	Prompt	135
2.14.3	Errors	139
2.15	Keys	140
2.16	Main command	141
2.17	Messages	144

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`.

`\unravel_get:nnN` {*options*} {*code*} *tl var*

Performs `\unravel:nn` with the *options* and *code* then saves the output into the *tl var*. The option `mute` is useful in this case.

`\unravel_setup:n` `\unravel_setup:n {options}`

See `\unravelsetup`.

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 `pdfTeX`, 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.

online

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.

`prompt-input`

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

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

`trace-assigns`
`trace-expansion`
`trace-other`

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

`welcome-message`

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

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

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

Differences.

- Kerning between letters of a word is omitted, which can lead to incorrect widths.
- Some primitives are not implemented yet: alignments (`\halign`, `\valign`, `\noalign`, `\omit`, `\span`, `\cr`, `\crrc`, `&`), many math mode primitives, and `\pdfprimitive`, `\discretionary`, as well as all primitives specific to engines other than pdf`TEX`. This list may sadly be incomplete!
- `\aftergroup` is only partially implemented.
- `\everyhbox`, `\everyvbox`, `\everymath`, `\everydisplay`, `\lastkern`, `\lastnodetype`, `\lastpenalty`, `\lastskip`, `\currentiflevel` and `\currentiftyp` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgrouptype` too.
- Setting `\globaldefs` to a non-zero value may cause problems.
- Tokens passed to `\aftergroup` are lost when `unravel` is done.
- For `unravel`, category codes are fixed when a file is read using `\input`, while `TEX` only fixes category codes when the corresponding characters are converted to tokens. Similarly, the argument of `\scantokens` is converted to the new category code regime in one go, and the result must be balanced.
- Explicit begin-group and end-group characters other than the usual left and right braces may make `unravel` choke, or may be silently replaced by the usual left and right braces.

- `\endinput` is ignored with a warning, as it is very difficult to implement it in a way similar to T_EX'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 to replay steps that have already been run.
- 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 `unrave1003` testfile for instance)

2 unravel implementation

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

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

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

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

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

```
8 \c84 11\c88 11\c35 6\c123 1\c125 2\c62 12\c61 12\c43 12 %
```

If ϵ -TeX's `\numexpr` or `\protected` are not available, bail out with an error.

```
9 \expandafter\ifx\csname numexpr\endcsname\relax
10 \errmessage{unravel requires \numexpr from eTeX}
11 \endinput\expandafter\endgroup\fi
12 \expandafter\ifx\csname protected\endcsname\relax
13 \errmessage{unravel requires \protected from eTeX}
14 \endinput\expandafter\endgroup\fi
```

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

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

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

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

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

(End definition for `_unravel_setup_catcodes:nmn.`)

`_unravel_setup_latexe:` This saves the catcodes and related parameters, then sets them to the value they normally have in a $\text{\LaTeX} 2_{\epsilon}$ package (in particular, @ is a letter).

```
53 \protected \gdef \_unravel_setup_latexe:
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_latexe:.`)

`__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} {2019/11/15} {0.2h} {Watching TeX digest tokens}%
136   \csname __unravel_setup_unravel:\endcsname
137 }%
```

2.1 Primitives, variants, and helpers

2.1.1 Renamed primitives

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

```

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` and `\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     }
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 257 \cs_new:Npn \_unravel_to_str:n #1
\_unravel_to_str_auxii:w 258 {
\_unravel_gtl_to_str:n 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 \tl \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 } { \tl_to_str:n } }
273 }
274 \exp_args:No \_unravel_tmp:w { \tl_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.

```

\_unravel_str_truncate_left_aux:nnn 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   { \tl_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.

```

\_unravel_str_truncate_right_aux:nnn 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     { \tl_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.
\__unravel_exit_hard:w 304 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
\__unravel_exit_point: 305 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }
                       306 \cs_new:Npn \__unravel_exit_hard:w #1 \__unravel_exit_point: #2 \__unravel_exit_point: { }

```

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

```

\__unravel_break:w    Useful to jump out of complicated conditionals.
\__unravel_break_point: 307 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
                       308 \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).

```

```

309 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
310   {
311     \int_compare:nNnTF
312       \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
313       { \prg_return_false: }
314     {
315       \int_compare:nNnTF
316         \l__unravel_head_cmd_int
317         > { \__unravel_tex_use:n { max_internal } }
318         { \prg_return_false: }
319         { \prg_return_true: }
320     }
321   }

```

(End definition for `__unravel_cmd_if_internal:TF`.)

2.1.6 Helpers concerning tokens

```

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

```

```

\__unravel_meaning_to_char_auxi:w 322 \cs_new:Npn \__unravel_meaning_to_char:n #1
\__unravel_meaning_to_char_auxii:w 323   { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
324 \cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
325   { \__unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
326 \cs_new:Npn \__unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
327   { \tl_if_empty:nTF {#2} { ~ } {#2} }
328 \cs_generate_variant:Nn \__unravel_meaning_to_char:n { o }
329 \cs_new:Npn \__unravel_token_to_char:N #1
330   { \__unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

(End definition for `_unravel_token_to_char:N` and others.)

`_unravel_token_if_expandable_p:N` We need to cook up our own version of `\token_if_expandable:N` because the `expl3`
`_unravel_token_if_expandable:N` one does not think that `undefined` is expandable.

```
331 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
332 { p , T , F , TF }
333 {
334   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
335   \prg_return_false:
336   \else:
337   \prg_return_true:
338   \fi:
339 }
```

(End definition for `_unravel_token_if_expandable:N`.)

`_unravel_token_if_protected_p:N` Returns `true` if the token is either not expandable or is a protected macro.

```
340 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
341 { p , T , F , TF }
342 {
343   \_unravel_token_if_expandable:N #1
344   {
345     \token_if_protected_macro:N #1
346     { \prg_return_true: }
347     {
348       \token_if_protected_long_macro:N #1
349       { \prg_return_true: }
350       { \prg_return_false: }
351     }
352   }
353   { \prg_return_true: }
354 }
```

(End definition for `_unravel_token_if_protected:N`.)

`_unravel_token_if_active_char:N` 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.

```
355 \group_begin:
356 \char_set_catcode_active:n { 'Z }
357 \prg_new_protected_conditional:Npnn \_unravel_token_if_active_char:N #1
358 { TF }
359 {
360   \group_begin:
361   \_unravel_exp_args:Nx \char_set_lccode:nn
362   { ' \exp_args:No \str_head:n { \token_to_str:N #1 } }
363   { ' Z }
364   \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
365   { \group_end: \prg_return_true: }
366   { \group_end: \prg_return_false: }
367 }
368 \group_end:
```

(End definition for `_unravel_token_if_active_char:N`.)

`_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.

```

369 \group_begin:
370   \char_set_catcode_active:n { 'Z }
371   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
372     { TF }
373     {
374       \group_begin:
375         \_unravel_set_escapechar:n { 92 }
376         \tl_set:Nx \l__unravel_tmpa_tl
377           { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
378         \tl_if_empty:NTF \l__unravel_tmpa_tl
379           {
380             \_unravel_token_if_active_char:NTF #1
381             { \group_end: \prg_return_true: }
382             { \group_end: \prg_return_false: }
383           }
384           { \group_end: \prg_return_true: }
385       }
386   \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.

```

387 \prg_new_protected_conditional:Npnn \_unravel_gtl_if_head_is_definable:N #1
388   { TF , F }
389   {
390     \gtl_if_single_token:NTF #1
391     {
392       \gtl_if_head_is_N_type:NTF #1
393       {
394         \gtl_head_do:NN #1 \_unravel_token_if_definable:NTF
395         { \prg_return_true: }
396         { \prg_return_false: }
397       }
398       { \prg_return_false: }
399     }
400     { \prg_return_false: }
401   }

```

(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
402 \cs_new:Npn \__unravel_prev_input_count:
403   {
404     \int_eval:n
405     {
406       0
407       \seq_map_function:NN \g__unravel_prev_input_seq
408         \__unravel_prev_input_count_aux:n
409     }
410   }
411 \cs_new:Npn \__unravel_prev_input_count_aux:n #1
412   { \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
413 \cs_new_protected:Npn \__unravel_prev_input_get:N
414   { \seq_get_right:NN \g__unravel_prev_input_seq }
415 \cs_new_protected:Npn \__unravel_prev_input_gpush:
416   { \seq_gput_right:Nn \g__unravel_prev_input_seq { } }
417 \cs_new_protected:Npn \__unravel_prev_input_gpush:N
418   { \seq_gput_right:NV \g__unravel_prev_input_seq }
419 \cs_new_protected:Npn \__unravel_prev_input_gpop:N
420   { \seq_gpop_right:NN \g__unravel_prev_input_seq }
421 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
422   { \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl }
423 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
424   { \seq_gput_right:NV \g__unravel_prev_input_seq }
425 \cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
426   { \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
427 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
428   {
429     \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
430     \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
431     \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
432   }
433 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V }
434 \cs_new_protected:Npn \__unravel_prev_input_silent:x
435   { \__unravel_exp_args:Nx \__unravel_prev_input_silent:n }
436 \cs_new_protected:Npn \__unravel_prev_input:n #1
437   {
438     \__unravel_prev_input_silent:n {#1}
439     \__unravel_print_action:x { \tl_to_str:n {#1} }
440   }
441 \cs_generate_variant:Nn \__unravel_prev_input:n { V }
442 \cs_new_protected:Npn \__unravel_prev_input:x
443   { \__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`

```
444 \cs_new_protected:Npn \_unravel_prev_input_gtl:N #1
445 {
446   \_unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
447   \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
448   \_unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
449 }
```

(End definition for _unravel_prev_input_gtl:N.)

`_unravel_prev_input_join_get:nnN`

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

`_unravel_join_get_aux:NNnN`

```
450 \cs_new_protected:Npn \_unravel_prev_input_join_get:nnN #1
451 {
452   \int_case:nnF {#1}
453   {
454     { 2 } { \_unravel_join_get_aux:NNnN \skip_eval:n \tex_glueexpr:D }
455     { 3 } { \_unravel_join_get_aux:NNnN \muskip_eval:n \tex_muexpr:D }
456   }
457   {
458     \_unravel_error:nnnnn { internal } { join-factor } { } { } { }
459     \_unravel_join_get_aux:NNnN \use:n \prg_do_nothing:
460   }
461 }
462 \cs_new_protected:Npn \_unravel_join_get_aux:NNnN #1#2#3#4
463 {
464   \_unravel_prev_input_gpop:N \l__unravel_head_tl
465   \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
466   \tl_set:Nx #4 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl #3 } }
467 }
```

(End definition for _unravel_prev_input_join_get:nnN and _unravel_join_get_aux:NNnN.)

2.2 Variables

2.2.1 User interaction

`\g_unravel_before_print_state_tl`

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

`\g_unravel_before_prompt_tl`

```
468 \tl_new:N \g_unravel_before_print_state_tl
469 \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`

```
470 \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.

```
471 \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 472 \bool_new:N \g__unravel_default_explicit_prompt_bool
\g__unravel_default_internal_debug_bool 473 \bool_new:N \g__unravel_default_internal_debug_bool
\g__unravel_default_number_steps_bool 474 \bool_new:N \g__unravel_default_number_steps_bool
\g__unravel_default_online_int 475 \int_new:N \g__unravel_default_online_int
\g__unravel_online_int 476 \clist_new:N \g__unravel_default_prompt_input_clist
\g__unravel_default_prompt_input_clist 477 \bool_new:N \g__unravel_default_trace_assigns_bool
\g__unravel_prompt_input_clist 478 \bool_new:N \g__unravel_default_trace_expansion_bool
\g__unravel_default_trace_assigns_bool 479 \bool_new:N \g__unravel_default_trace_other_bool
\g__unravel_trace_assigns_bool 480 \bool_new:N \g__unravel_default_welcome_message_bool
\g__unravel_default_trace_expansion_bool 481 \bool_gset_true:N \g__unravel_default_number_steps_bool
\g__unravel_trace_expansion_bool 482 \int_gset:Nn \g__unravel_default_online_int { 1 }
\g__unravel_default_trace_other_bool 483 \bool_gset_true:N \g__unravel_default_trace_assigns_bool
\g__unravel_trace_other_bool 484 \bool_gset_true:N \g__unravel_default_trace_expansion_bool
\g__unravel_trace_other_bool 485 \bool_gset_true:N \g__unravel_default_trace_other_bool
\g__unravel_default_welcome_message_bool 486 \bool_gset_true:N \g__unravel_default_welcome_message_bool
\g__unravel_welcome_message_bool 487 \bool_new:N \g__unravel_explicit_prompt_bool
488 \bool_new:N \g__unravel_internal_debug_bool
489 \bool_new:N \g__unravel_number_steps_bool
490 \int_new:N \g__unravel_online_int
491 \clist_new:N \g__unravel_prompt_input_clist
492 \bool_new:N \g__unravel_trace_assigns_bool
493 \bool_new:N \g__unravel_trace_expansion_bool
494 \bool_new:N \g__unravel_trace_other_bool
495 \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.

```
496 \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.

```
497 \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_action_int 498 \int_new:N \g__unravel_default_max_action_int
\g__unravel_default_max_output_int 499 \int_new:N \g__unravel_default_max_output_int
\g__unravel_default_max_input_int 500 \int_new:N \g__unravel_default_max_input_int
\g__unravel_max_action_int 501 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
\g__unravel_max_output_int 502 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
\g__unravel_max_input_int 503 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
504 \int_new:N \g__unravel_max_action_int
505 \int_new:N \g__unravel_max_output_int
506 \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.

```
507 \bool_new:N \g_unravel_speedup_macros_bool
508 \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.

```
509 \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.

```
510 \int_new:N \g__unravel_input_int
```

(End definition for \g__unravel_input_int.)

`\g__unravel_input_tmpa_int`

`\l__unravel_input_tmpa_tl`

```
511 \int_new:N \g__unravel_input_tmpa_int
```

```
512 \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.

```
513 \seq_new:N \g__unravel_prev_input_seq
```

```
514 \tl_new:N \l__unravel_prev_input_tl
```

```
515 \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.

```
516 \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.

```
517 \gtl_new:N \l__unravel_head_gtl
```

```
518 \tl_new:N \l__unravel_head_tl
```

```
519 \cs_new_eq:NN \l__unravel_head_token ?
```

```
520 \int_new:N \l__unravel_head_cmd_int
```

```
521 \int_new:N \l__unravel_head_char_int
```

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

`\l__unravel_head_meaning_tl`

522 `\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` 523 `\tl_new:N \l__unravel_argi_tl`

524 `\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` 525 `\tl_new:N \l__unravel_tmpa_tl`

`\l__unravel_tmpa_seq` 526 `\gtl_new:N \l__unravel_unused_gtl`

`\l__unravel_unused_gtl` 527 `\gtl_new:N \l__unravel_tmpb_gtl`

`\l__unravel_tmpb_token` 528 `\tl_new:N \g__unravel_tmpc_tl`

529 `\seq_new:N \l__unravel_tmpa_seq`

530 `\cs_new_eq:NN \l__unravel_tmpb_token ?`

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

`\l__unravel_defined_tl` The token that is defined by the prefixed command (such as `\chardef` or `\futurelet`),
`\l__unravel_defining_tl` 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.

531 `\tl_new:N \l__unravel_defined_tl`

532 `\tl_new:N \l__unravel_defining_tl`

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

`__unravel_inaccessible:w`

533 `\cs_new_eq:NN __unravel_inaccessible:w ?`

(End definition for `__unravel_inaccessible:w`.)

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

534 `\gtl_new:N \g__unravel_after_assignment_gtl`

535 `\bool_new:N \g__unravel_set_box_allowed_bool`

536 `\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.

537 `\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.

```
538 \group_begin:
539   \cs_set:Npx \__unravel_tmp:w #1 { \c_hash_str #1 }
540   \tl_const:Nx \c__unravel_parameters_tl
541     { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
542 \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.

```
543 \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 544 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 545 \int_new:N \g__unravel_if_limit_int
546 \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`

```
547 \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.

```
548 \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.

```
549 \int_new:N \g__unravel_ends_int
550 \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 551 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 552 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 553 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 554 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 555 \tl_const:Nn \c__unravel_lq_tl { ‘ }
\c__unravel_dq_tl 556 \tl_const:Nn \c__unravel_rq_tl { ’ }
\c__unravel_lp_tl 557 \tl_const:Nn \c__unravel_dq_tl { " }
\c__unravel_rp_tl 558 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 559 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 560 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 561 \tl_const:Nn \c__unravel_comma_tl { , }
562 \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:.
563 \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.

```

```

564 \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 565 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
566 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
567 \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
568 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
569 {
570   \tl_const:cx { c__unravel_tex_#1_tl }
571   { { \__unravel_tex_use:n {#2} } {#3} }
572 }

```

(End definition for `__unravel_tex_primitive:nnn`.)


```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
573 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
574 {
575   \cs_new_protected:cpn
576     { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
577 }
578 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
579 {
580   \cs_new_eq:cc
581     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
582     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
583 }

```

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

```

\__unravel_new_tex_expandable:nn
584 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
585 {
586   \cs_new_protected:cpn
587     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
588 }

```

(End definition for `__unravel_new_tex_expandable:nn`.)

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

```

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

```

```

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

```

```

673 \_unravel_tex_const:nn { set_prev_graf           } { 80 }
674 \_unravel_tex_const:nn { set_page_dimen         } { 81 }
675 \_unravel_tex_const:nn { set_page_int          } { 82 }
676 \_unravel_tex_const:nn { set_box_dimen         } { 83 }
677 \_unravel_tex_const:nn { set_shape             } { 84 }
678 \_unravel_tex_const:nn { def_code              } { 85 }
679 \_unravel_tex_const:nn { def_family            } { 86 }
680 \_unravel_tex_const:nn { set_font              } { 87 }
681 \_unravel_tex_const:nn { def_font             } { 88 }
682 \_unravel_tex_const:nn { register             } { 89 }
683 \_unravel_tex_const:nn { max_internal          } { 89 }
684 \_unravel_tex_const:nn { advance              } { 90 }
685 \_unravel_tex_const:nn { multiply             } { 91 }
686 \_unravel_tex_const:nn { divide              } { 92 }
687 \_unravel_tex_const:nn { prefix              } { 93 }
688 \_unravel_tex_const:nn { let                 } { 94 }
689 \_unravel_tex_const:nn { shorthand_def        } { 95 }
690 \_unravel_tex_const:nn { read_to_cs          } { 96 }
691 \_unravel_tex_const:nn { def                 } { 97 }
692 \_unravel_tex_const:nn { set_box             } { 98 }
693 \_unravel_tex_const:nn { hyph_data           } { 99 }
694 \_unravel_tex_const:nn { set_interaction      } { 100 }
695 \_unravel_tex_const:nn { letterspace_font    } { 101 }
696 \_unravel_tex_const:nn { pdf_copy_font       } { 102 }
697 \_unravel_tex_const:nn { max_command         } { 102 }
698 \_unravel_tex_const:nn { undefined_cs        } { 103 }
699 \_unravel_tex_const:nn { expand_after        } { 104 }
700 \_unravel_tex_const:nn { no_expand           } { 105 }
701 \_unravel_tex_const:nn { input              } { 106 }
702 \_unravel_tex_const:nn { if_test            } { 107 }
703 \_unravel_tex_const:nn { fi_or_else         } { 108 }
704 \_unravel_tex_const:nn { cs_name            } { 109 }
705 \_unravel_tex_const:nn { convert           } { 110 }
706 \_unravel_tex_const:nn { the                } { 111 }
707 \_unravel_tex_const:nn { top_bot_mark       } { 112 }
708 \_unravel_tex_const:nn { call              } { 113 }
709 \_unravel_tex_const:nn { end_template       } { 117 }

```

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

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

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

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

1031 \_unravel_tex_primitive:nnn { tracingnesting } { assign_int } { 86 }
1032 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
1033 \_unravel_tex_primitive:nnn { lastlinefit } { assign_int } { 88 }
1034 \_unravel_tex_primitive:nnn { savingdiscards } { assign_int } { 89 }
1035 \_unravel_tex_primitive:nnn { savinghyphcodes } { assign_int } { 90 }
1036 \_unravel_tex_primitive:nnn { TeXTeXtstate } { assign_int } { 91 }
1037 \_unravel_tex_primitive:nnn { parindent } { assign_dimen } { 0 }
1038 \_unravel_tex_primitive:nnn { mathsurround } { assign_dimen } { 1 }
1039 \_unravel_tex_primitive:nnn { lineskiplimit } { assign_dimen } { 2 }
1040 \_unravel_tex_primitive:nnn { hsize } { assign_dimen } { 3 }
1041 \_unravel_tex_primitive:nnn { vsize } { assign_dimen } { 4 }
1042 \_unravel_tex_primitive:nnn { maxdepth } { assign_dimen } { 5 }
1043 \_unravel_tex_primitive:nnn { splitmaxdepth } { assign_dimen } { 6 }
1044 \_unravel_tex_primitive:nnn { boxmaxdepth } { assign_dimen } { 7 }
1045 \_unravel_tex_primitive:nnn { hfuzz } { assign_dimen } { 8 }
1046 \_unravel_tex_primitive:nnn { vfuzz } { assign_dimen } { 9 }
1047 \_unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
1048 \_unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
1049 \_unravel_tex_primitive:nnn { scriptspace } { assign_dimen } { 12 }
1050 \_unravel_tex_primitive:nnn { predisplaysize } { assign_dimen } { 13 }
1051 \_unravel_tex_primitive:nnn { displaywidth } { assign_dimen } { 14 }
1052 \_unravel_tex_primitive:nnn { displayindent } { assign_dimen } { 15 }
1053 \_unravel_tex_primitive:nnn { overfullrule } { assign_dimen } { 16 }
1054 \_unravel_tex_primitive:nnn { hangindent } { assign_dimen } { 17 }
1055 \_unravel_tex_primitive:nnn { hoffset } { assign_dimen } { 18 }
1056 \_unravel_tex_primitive:nnn { voffset } { assign_dimen } { 19 }
1057 \_unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
1058 \_unravel_tex_primitive:nnn { pdfhorigin } { assign_dimen } { 21 }
1059 \_unravel_tex_primitive:nnn { pdfvorigin } { assign_dimen } { 22 }
1060 \_unravel_tex_primitive:nnn { pdfpagewidth } { assign_dimen } { 23 }
1061 \_unravel_tex_primitive:nnn { pdfpageheight } { assign_dimen } { 24 }
1062 \_unravel_tex_primitive:nnn { pdflinkmargin } { assign_dimen } { 25 }
1063 \_unravel_tex_primitive:nnn { pdfdestmargin } { assign_dimen } { 26 }
1064 \_unravel_tex_primitive:nnn { pdfthreadmargin } { assign_dimen } { 27 }
1065 \_unravel_tex_primitive:nnn { pdffirstlineheight } { assign_dimen } { 28 }
1066 \_unravel_tex_primitive:nnn { pdflastlinedepth } { assign_dimen } { 29 }
1067 \_unravel_tex_primitive:nnn { pdfeachlineheight } { assign_dimen } { 30 }
1068 \_unravel_tex_primitive:nnn { pdfeachlinedepth } { assign_dimen } { 31 }
1069 \_unravel_tex_primitive:nnn { pdfignoreddimen } { assign_dimen } { 32 }
1070 \_unravel_tex_primitive:nnn { pdfpxdimen } { assign_dimen } { 33 }
1071 \_unravel_tex_primitive:nnn { lineskip } { assign_glue } { 0 }
1072 \_unravel_tex_primitive:nnn { baselineskip } { assign_glue } { 1 }
1073 \_unravel_tex_primitive:nnn { parskip } { assign_glue } { 2 }
1074 \_unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue } { 3 }
1075 \_unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue } { 4 }
1076 \_unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
1077 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1078 \_unravel_tex_primitive:nnn { leftskip } { assign_glue } { 7 }
1079 \_unravel_tex_primitive:nnn { rightskip } { assign_glue } { 8 }
1080 \_unravel_tex_primitive:nnn { topskip } { assign_glue } { 9 }
1081 \_unravel_tex_primitive:nnn { splittopskip } { assign_glue } { 10 }
1082 \_unravel_tex_primitive:nnn { tabskip } { assign_glue } { 11 }
1083 \_unravel_tex_primitive:nnn { spaceskip } { assign_glue } { 12 }
1084 \_unravel_tex_primitive:nnn { xspaceskip } { assign_glue } { 13 }

```

```

1085 \_unravel_tex_primitive:nnn { parfillskip           } { assign_glue } { 14 }
1086 \_unravel_tex_primitive:nnn { thinmuskip         } { assign_mu_glue } { 15 }
1087 \_unravel_tex_primitive:nnn { medmuskip         } { assign_mu_glue } { 16 }
1088 \_unravel_tex_primitive:nnn { thickmuskip       } { assign_mu_glue } { 17 }
1089 \_unravel_tex_primitive:nnn { fontdimen        } { assign_font_dimen } { 0 }
1090 \_unravel_tex_primitive:nnn { hyphenchar       } { assign_font_int } { 0 }
1091 \_unravel_tex_primitive:nnn { skewchar         } { assign_font_int } { 1 }
1092 \_unravel_tex_primitive:nnn { lpcode           } { assign_font_int } { 2 }
1093 \_unravel_tex_primitive:nnn { rpcode           } { assign_font_int } { 3 }
1094 \_unravel_tex_primitive:nnn { efcodes          } { assign_font_int } { 4 }
1095 \_unravel_tex_primitive:nnn { tagcode          } { assign_font_int } { 5 }
1096 \_unravel_tex_primitive:nnn { pdfnoligatures  } { assign_font_int } { 6 }
1097 \_unravel_tex_primitive:nnn { knbscode        } { assign_font_int } { 7 }
1098 \_unravel_tex_primitive:nnn { stbscode        } { assign_font_int } { 8 }
1099 \_unravel_tex_primitive:nnn { shbscode        } { assign_font_int } { 9 }
1100 \_unravel_tex_primitive:nnn { knbccode        } { assign_font_int } { 10 }
1101 \_unravel_tex_primitive:nnn { knaccode        } { assign_font_int } { 11 }
1102 \_unravel_tex_primitive:nnn { spacefactor     } { set_aux } { 102 }
1103 \_unravel_tex_primitive:nnn { prevdepth       } { set_aux } { 1 }
1104 \_unravel_tex_primitive:nnn { prevgraf        } { set_prev_graf } { 0 }
1105 \_unravel_tex_primitive:nnn { pagegoal        } { set_page_dimen } { 0 }
1106 \_unravel_tex_primitive:nnn { pagetotal       } { set_page_dimen } { 1 }
1107 \_unravel_tex_primitive:nnn { pagestretch     } { set_page_dimen } { 2 }
1108 \_unravel_tex_primitive:nnn { pagefilstretch  } { set_page_dimen } { 3 }
1109 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 4 }
1110 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 5 }
1111 \_unravel_tex_primitive:nnn { pageshrink      } { set_page_dimen } { 6 }
1112 \_unravel_tex_primitive:nnn { pagedepth      } { set_page_dimen } { 7 }
1113 \_unravel_tex_primitive:nnn { deadcycles      } { set_page_int } { 0 }
1114 \_unravel_tex_primitive:nnn { insertpenalties } { set_page_int } { 1 }
1115 \_unravel_tex_primitive:nnn { interactionmode } { set_page_int } { 2 }
1116 \_unravel_tex_primitive:nnn { wd              } { set_box_dimen } { 1 }
1117 \_unravel_tex_primitive:nnn { dp              } { set_box_dimen } { 2 }
1118 \_unravel_tex_primitive:nnn { ht              } { set_box_dimen } { 3 }
1119 \_unravel_tex_primitive:nnn { parshape        } { set_shape } { 0 }
1120 \_unravel_tex_primitive:nnn { interlinepenalties } { set_shape } { 1 }
1121 \_unravel_tex_primitive:nnn { clubpenalties  } { set_shape } { 2 }
1122 \_unravel_tex_primitive:nnn { widowpenalties } { set_shape } { 3 }
1123 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
1124 \_unravel_tex_primitive:nnn { catcode         } { def_code } { 0 }
1125 \_unravel_tex_primitive:nnn { lccode          } { def_code } { 256 }
1126 \_unravel_tex_primitive:nnn { uccode          } { def_code } { 512 }
1127 \_unravel_tex_primitive:nnn { sfcodes         } { def_code } { 768 }
1128 \_unravel_tex_primitive:nnn { mathcodes       } { def_code } { 1024 }
1129 \_unravel_tex_primitive:nnn { delcodes        } { def_code } { 1591 }
1130 \_unravel_tex_primitive:nnn { textfont        } { def_family } { -48 }
1131 \_unravel_tex_primitive:nnn { scriptfont      } { def_family } { -32 }
1132 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family } { -16 }
1133 \_unravel_tex_primitive:nnn { nullfont        } { set_font } { 0 }
1134 \_unravel_tex_primitive:nnn { font            } { def_font } { 0 }
1135 \_unravel_tex_primitive:nnn { count           } { register } { 1 000 000 }
1136 \_unravel_tex_primitive:nnn { dimen           } { register } { 2 000 000 }
1137 \_unravel_tex_primitive:nnn { skip            } { register } { 3 000 000 }
1138 \_unravel_tex_primitive:nnn { muskip          } { register } { 4 000 000 }

```

```

1139 \_unravel_tex_primitive:nnn { advance } { advance } { 0 }
1140 \_unravel_tex_primitive:nnn { multiply } { multiply } { 0 }
1141 \_unravel_tex_primitive:nnn { divide } { divide } { 0 }
1142 \_unravel_tex_primitive:nnn { long } { prefix } { 1 }
1143 \_unravel_tex_primitive:nnn { outer } { prefix } { 2 }
1144 \_unravel_tex_primitive:nnn { global } { prefix } { 4 }
1145 \_unravel_tex_primitive:nnn { protected } { prefix } { 8 }
1146 \_unravel_tex_primitive:nnn { let } { let } { 0 }
1147 \_unravel_tex_primitive:nnn { futurelet } { let } { 1 }
1148 \_unravel_tex_primitive:nnn { chardef } { shorthand_def } { 0 }
1149 \_unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
1150 \_unravel_tex_primitive:nnn { countdef } { shorthand_def } { 2 }
1151 \_unravel_tex_primitive:nnn { dimendef } { shorthand_def } { 3 }
1152 \_unravel_tex_primitive:nnn { skipdef } { shorthand_def } { 4 }
1153 \_unravel_tex_primitive:nnn { muskipdef } { shorthand_def } { 5 }
1154 \_unravel_tex_primitive:nnn { toksdef } { shorthand_def } { 6 }
1155 \_unravel_tex_primitive:nnn { read } { read_to_cs } { 0 }
1156 \_unravel_tex_primitive:nnn { readline } { read_to_cs } { 1 }
1157 \_unravel_tex_primitive:nnn { def } { def } { 0 }
1158 \_unravel_tex_primitive:nnn { gdef } { def } { 1 }
1159 \_unravel_tex_primitive:nnn { edef } { def } { 2 }
1160 \_unravel_tex_primitive:nnn { xdef } { def } { 3 }
1161 \_unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1162 \_unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1163 \_unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1164 \_unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1165 \_unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1166 \_unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1167 \_unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1168 \_unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1169 \_unravel_tex_primitive:nnn { pdfcopyfont } { pdf_copy_font } { 0 }
1170 \_unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1171 \_unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1172 \_unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1173 \_unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1174 \_unravel_tex_primitive:nnn { pdfprimitive } { no_expand } { 1 }
1175 \_unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1176 \_unravel_tex_primitive:nnn { input } { input } { 0 }
1177 \_unravel_tex_primitive:nnn { endinput } { input } { 1 }
1178 \_unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1179 \_unravel_tex_primitive:nnn { if } { if_test } { 0 }
1180 \_unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }
1181 \_unravel_tex_primitive:nnn { ifnum } { if_test } { 2 }
1182 \_unravel_tex_primitive:nnn { ifdim } { if_test } { 3 }
1183 \_unravel_tex_primitive:nnn { ifodd } { if_test } { 4 }
1184 \_unravel_tex_primitive:nnn { ifvmode } { if_test } { 5 }
1185 \_unravel_tex_primitive:nnn { ifhmode } { if_test } { 6 }
1186 \_unravel_tex_primitive:nnn { ifmmode } { if_test } { 7 }
1187 \_unravel_tex_primitive:nnn { ifinner } { if_test } { 8 }
1188 \_unravel_tex_primitive:nnn { ifvoid } { if_test } { 9 }
1189 \_unravel_tex_primitive:nnn { ifhbox } { if_test } { 10 }
1190 \_unravel_tex_primitive:nnn { ifvbox } { if_test } { 11 }
1191 \_unravel_tex_primitive:nnn { ifx } { if_test } { 12 }
1192 \_unravel_tex_primitive:nnn { ifeof } { if_test } { 13 }

```

```

1193 \_unravel_tex_primitive:nnn { iftrue } { if_test } { 14 }
1194 \_unravel_tex_primitive:nnn { iffalse } { if_test } { 15 }
1195 \_unravel_tex_primitive:nnn { ifcase } { if_test } { 16 }
1196 \_unravel_tex_primitive:nnn { ifdefined } { if_test } { 17 }
1197 \_unravel_tex_primitive:nnn { ifcsname } { if_test } { 18 }
1198 \_unravel_tex_primitive:nnn { iffontchar } { if_test } { 19 }
1199 \_unravel_tex_primitive:nnn { ifincsname } { if_test } { 20 }
1200 \_unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1201 \_unravel_tex_primitive:nnn { ifpdfabsnum } { if_test } { 22 }
1202 \_unravel_tex_primitive:nnn { ifpdfabsdim } { if_test } { 23 }
1203 \_unravel_tex_primitive:nnn { fi } { fi_or_else } { 2 }
1204 \_unravel_tex_primitive:nnn { else } { fi_or_else } { 3 }
1205 \_unravel_tex_primitive:nnn { or } { fi_or_else } { 4 }
1206 \_unravel_tex_primitive:nnn { csname } { cs_name } { 0 }
1207 \_unravel_tex_primitive:nnn { number } { convert } { 0 }
1208 \_unravel_tex_primitive:nnn { romannumeral } { convert } { 1 }
1209 \_unravel_tex_primitive:nnn { string } { convert } { 2 }
1210 \_unravel_tex_primitive:nnn { meaning } { convert } { 3 }
1211 \_unravel_tex_primitive:nnn { fontname } { convert } { 4 }
1212 \_unravel_tex_primitive:nnn { eTeXrevision } { convert } { 5 }
1213 \_unravel_tex_primitive:nnn { pdftexrevision } { convert } { 6 }
1214 \_unravel_tex_primitive:nnn { pdftexbanner } { convert } { 7 }
1215 \_unravel_tex_primitive:nnn { pdffontname } { convert } { 8 }
1216 \_unravel_tex_primitive:nnn { pdffontobjnum } { convert } { 9 }
1217 \_unravel_tex_primitive:nnn { pdffontsize } { convert } { 10 }
1218 \_unravel_tex_primitive:nnn { pdfpageref } { convert } { 11 }
1219 \_unravel_tex_primitive:nnn { pdfxformname } { convert } { 12 }
1220 \_unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
1221 \_unravel_tex_primitive:nnn { pdfescapename } { convert } { 14 }
1222 \_unravel_tex_primitive:nnn { leftmarginkern } { convert } { 15 }
1223 \_unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
1224 \_unravel_tex_primitive:nnn
1225 { \sys_if_engine_xetex:F { pdf } strcmp } { convert } { 17 }
1226 \_unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
1227 \_unravel_tex_primitive:nnn { pdfescapehex } { convert } { 19 }
1228 \_unravel_tex_primitive:nnn { pdfunescapehex } { convert } { 20 }
1229 \_unravel_tex_primitive:nnn { pdfcreationdate } { convert } { 21 }
1230 \_unravel_tex_primitive:nnn { pdffilemoddate } { convert } { 22 }
1231 \_unravel_tex_primitive:nnn { pdffilesize } { convert } { 23 }
1232 \_unravel_tex_primitive:nnn { pdfmdfivesum } { convert } { 24 }
1233 \_unravel_tex_primitive:nnn { pdffiledump } { convert } { 25 }
1234 \_unravel_tex_primitive:nnn { pdfmatch } { convert } { 26 }
1235 \_unravel_tex_primitive:nnn { pdflastmatch } { convert } { 27 }
1236 \_unravel_tex_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
1237 \_unravel_tex_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
1238 \_unravel_tex_primitive:nnn { pdfinsertht } { convert } { 30 }
1239 \_unravel_tex_primitive:nnn { pdfximagebbox } { convert } { 31 }
1240 \_unravel_tex_primitive:nnn { jobname } { convert } { 32 }
1241 \sys_if_engine luatex:T
1242 { \_unravel_tex_primitive:nnn { directlua } { convert } { 33 } }
1243 \_unravel_tex_primitive:nnn { expanded } { convert } { 34 }
1244 \sys_if_engine luatex:T
1245 { \_unravel_tex_primitive:nnn { luaescapestring } { convert } { 35 } }
1246 \sys_if_engine_xetex:T

```

```

1247 {
1248   \_unravel_tex_primitive:nmn { Ucharcat      } { convert } { 40 }
1249 }
1250 \_unravel_tex_primitive:nmn { the              } { the } { 0 }
1251 \_unravel_tex_primitive:nmn { unexpanded      } { the } { 1 }
1252 \_unravel_tex_primitive:nmn { detokenize      } { the } { 5 }
1253 \_unravel_tex_primitive:nmn { topmark         } { top_bot_mark } { 0 }
1254 \_unravel_tex_primitive:nmn { firstmark       } { top_bot_mark } { 1 }
1255 \_unravel_tex_primitive:nmn { botmark         } { top_bot_mark } { 2 }
1256 \_unravel_tex_primitive:nmn { splitfirstmark  } { top_bot_mark } { 3 }
1257 \_unravel_tex_primitive:nmn { splitbotmark    } { top_bot_mark } { 4 }
1258 \_unravel_tex_primitive:nmn { topmarks        } { top_bot_mark } { 5 }
1259 \_unravel_tex_primitive:nmn { firstmarks      } { top_bot_mark } { 6 }
1260 \_unravel_tex_primitive:nmn { botmarks        } { top_bot_mark } { 7 }
1261 \_unravel_tex_primitive:nmn { splitfirstmarks  } { top_bot_mark } { 8 }
1262 \_unravel_tex_primitive:nmn { 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, insert a frozen \relax (the alternative would be either to grab a
\_unravel_get_next_aux:w token in the input stream after \unravel, which is tough, or simply produce an error
and exit; perhaps this should be configurable). Then remove the first token in the input,
and store it in \l__unravel_head_gtl. Set \l__unravel_head_token equal in meaning
to that first token. Then set \l__unravel_head_tl to contain the token, unless it is a
begin-group or end-group character, in which case this token list is emptied.

```

```

1263 \cs_new_protected:Npn \_unravel_get_next:
1264 {
1265   \_unravel_input_if_empty:TF
1266   {
1267     \_unravel_error:nmnnn { runaway-unravel } { } { } { } { }
1268     \_unravel_back_input_gtl:N \c__unravel_frozen_relax_gtl
1269   }
1270   { }
1271   \_unravel_input_gpop:N \l__unravel_head_gtl
1272   \gtl_head_do:NN \l__unravel_head_gtl \_unravel_get_next_aux:w
1273   \gtl_if_tl:NNTF \l__unravel_head_gtl
1274   {
1275     \tl_set:Nx \l__unravel_head_tl
1276     { \gtl_head:N \l__unravel_head_gtl }
1277     \token_if_eq_meaning:NNT
1278     \l__unravel_head_token \_unravel_special_relax:
1279     \_unravel_get_next_notexpanded:
1280   }

```

```

1281     { \tl_clear:N \l__unravel_head_tl }
1282   }
1283   \cs_new_protected:Npn \__unravel_get_next_aux:w
1284     { \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 of that check the control sequence has the form `\notexpanded:...`. Since we don't know the escape character we must use `\cs_to_str:N`, but that function is not meant for active characters and has a runaway argument if its argument is a space (active since we know its meaning is the special `\relax`). To avoid the runaway we include an arbitrary delimiter Z. If the token in `\l__unravel_head_tl` is not `\notexpanded:...` we do nothing. Otherwise `__unravel_notexpanded_expand:n` reconstructs the token that was hit with `\noexpand` (an active character if the argument is a single character) and do the job of `__unravel_get_next:`, setting `\l__unravel_head_token` to the special `\relax` marker for expandable commands, as `\noexpand` would.

```

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

```

```

1320         \__unravel_notexpanded_expand:N { \use_none:n #1 }
1321     }
1322 }
1323 \group_end:
1324 \cs_new_protected:Npn \__unravel_notexpanded_expand:N #1
1325 {
1326     \gtl_set:Nn \l__unravel_head_gtl {#1}
1327     \tl_set:Nn \l__unravel_head_tl {#1}
1328     \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
1329 }

```

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

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

```

1330 \cs_new_protected:Npn \__unravel_get_next_marker:
1331 {
1332     \__unravel_get_next:
1333     \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
1334     { \__unravel_error:nxxxx { nested-unravel } { } { } { } { } }
1335     { \__unravel_error:nxxxx { internal } { marker-unknown } { } { } { } }
1336     \__unravel_input_gpop_item:NF \l__unravel_argi_tl
1337     { \__unravel_error:nxxxx { internal } { marker-1 } { } { } { } }
1338     \__unravel_input_gpop_item:NF \l__unravel_argii_tl
1339     { \__unravel_error:nxxxx { internal } { marker-2 } { } { } { } }
1340     \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
1341     \__unravel_exp_args:Nx \__unravel_back_input:n
1342     { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
1343     \__unravel_get_next:
1344 }

```

(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`.

```

1345 \cs_new_protected:Npn \__unravel_get_token:
1346 {
1347     \__unravel_get_next:
1348     \__unravel_set_cmd:
1349 }

```

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

```

1350 \cs_new_protected:Npn \__unravel_set_cmd:
1351 {
1352     \__unravel_set_cmd_aux_meaning:
1353     \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }

```



```

1354     { }
1355     {
1356     \__unravel_token_if_expandable:NTF \l__unravel_head_token
1357     {
1358     \token_if_macro:NTF \l__unravel_head_token
1359     { \__unravel_set_cmd_aux_macro: }
1360     { \__unravel_set_cmd_aux_unknown: }
1361     }
1362     {
1363     \token_if_cs:NTF \l__unravel_head_token
1364     { \__unravel_set_cmd_aux_cs: }
1365     { \__unravel_set_cmd_aux_char: }
1366     }
1367     }
1368 }

```

(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.

```

1369 \cs_new_protected:Npn \__unravel_set_cmd_aux_meaning:
1370 {
1371   \tl_set:Nx \l__unravel_head_meaning_tl
1372   {
1373     \exp_after:wN \__unravel_strip_escape:w
1374     \token_to_meaning:N \l__unravel_head_token
1375     \tl_to_str:n { : }
1376   }
1377   \tl_set:Nx \l__unravel_head_meaning_tl
1378   {
1379     \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1380     \l__unravel_head_meaning_tl \q_stop
1381   }
1382 }
1383 \use:x
1384 {
1385   \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1386   ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1387 }

```

(End definition for __unravel_set_cmd_aux_meaning: and __unravel_set_cmd_aux_meaning:w.)

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

```

\__unravel_set_cmd_aux_primitive:oTF
\__unravel_set_cmd_aux_primitive:nn
1388 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
1389 {
1390   \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1391   {
1392     \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1393     { c__unravel_tex_#1_tl }
1394     #2
1395   }

```

```

1396 }
1397 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
1398 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
1399 {
1400   \int_set:Nn \l__unravel_head_cmd_int {#1}
1401   \int_set:Nn \l__unravel_head_char_int {#2}
1402 }

```

(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.

```

1403 \cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
1404 {
1405   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1406   \int_zero:N \l__unravel_head_char_int
1407 }

```

(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.

```

1408 \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1409 {
1410   \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1411   \c__unravel_tex_relax_tl
1412   \__unravel_error:nxxxx { unknown-primitive }
1413   { \l__unravel_head_meaning_tl } { } { } { }
1414 }

```

(End definition for __unravel_set_cmd_aux_unknown:.)

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

```

1415 \cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
1416 {
1417   \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1418   { \tl_to_str:n { elect-font } }
1419   {
1420     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1421     \c__unravel_tex_nullfont_tl
1422   }
1423   { \__unravel_set_cmd_aux_numeric: }
1424 }

```

(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` or `omathchar` in (u)pTeX), or one of `count`, `dimen`, `skip`, `muskip`, or `toks`. In the first two (three) cases, the command is `char_given` or `math_given`. It is otherwise identical to the corresponding primitive

(\count *etc.*). We then keep track of the associated number (part after \q_mark) in \l__unravel_head_char_int. For unknown non-expandable primitives, assuming that their meaning consists solely of letters, the \q_mark is inserted at their end, and is followed by +0, so nothing breaks.

```

1425 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:
1426 {
1427   \tl_set:Nx \l__unravel_tmpa_tl
1428     {
1429       \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1430       \l__unravel_head_meaning_tl + 0
1431     }
1432   \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1433   \l__unravel_tmpa_tl \q_stop
1434 }
1435 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1436 {
1437   \if_int_compare:w '#1 < 'A \exp_stop_f:
1438     \exp_not:N \q_mark
1439     \exp_after:wN \use_i:nn
1440   \fi:
1441   #1 \__unravel_set_cmd_aux_numeric:N
1442 }
1443 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1444 {
1445   \str_case:nnF {#1}
1446     {
1447       { char }      { \__unravel_set_cmd_aux_given:n { char_given } }
1448       { kchar }    { \__unravel_set_cmd_aux_given:n { char_given } }
1449       { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1450       { omathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1451     }
1452     {
1453       \__unravel_set_cmd_aux_primitive:nTF {#1}
1454         { }
1455         { \__unravel_set_cmd_aux_unknown: }
1456       \int_add:Nn \l__unravel_head_char_int { 100 000 }
1457     }
1458   \int_add:Nn \l__unravel_head_char_int {#2}
1459 }
1460 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1461 {
1462   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1463   \int_zero:N \l__unravel_head_char_int
1464 }

```

(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 ‘.

```

1465 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:

```

```

1466 {
1467   \tl_set:Nx \l__unravel_head_meaning_tl
1468     { \token_to_meaning:N \l__unravel_head_token }
1469   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1470     { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1471   \exp_after:wN \__unravel_set_cmd_aux_char:w
1472     \l__unravel_head_meaning_tl \q_stop
1473   \__unravel_exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1474     { ' \__unravel_token_to_char:N \l__unravel_head_token }
1475 }
1476 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1477 {
1478   \int_set:Nn \l__unravel_head_cmd_int
1479     { \__unravel_tex_use:n { #1_char } }
1480 }

```

(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.

```

1481 \cs_new:Npn \__unravel_input_to_str:
1482 {
1483   \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1484     \__unravel_input_to_str_aux:n
1485 }
1486 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1487 { \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.

```

1488 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1489 {
1490   \int_compare:nNnTF \g__unravel_input_int = 0
1491     { \use_i:nn }
1492     {
1493       \gtl_if_empty:cTF
1494         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1495         {
1496           \int_gdecr:N \g__unravel_input_int
1497           \__unravel_input_if_empty:TF
1498         }
1499         {
1500           \__unravel_input_split:
1501           \use_ii:nn
1502         }
1503     }
1504 }

```

(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

```

1505 \cs_new_protected:Npn \_unravel_input_split:
1506 {
1507   \int_compare:nNnT \g__unravel_input_int = 1
1508     {
1509       \exp_args:Nc \_unravel_input_split_aux:N
1510         { g__unravel_input_1_gtl }
1511     }
1512 }
1513 \cs_new_protected:Npn \_unravel_input_split_aux:N #1
1514 {
1515   \gtl_if_tl:NT #1
1516   {
1517     \gtl_if_head_is_N_type:NT #1
1518     {
1519       \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1520       \_unravel_exp_args:NNx \use:nn
1521         \_unravel_input_split_auxii:N
1522         { \tl_head:N \l__unravel_input_tmpa_tl }
1523     }
1524   }
1525 }
1526 \cs_new_protected:Npn \_unravel_input_split_auxii:N #1
1527 {
1528   \token_if_parameter:NF #1
1529   {
1530     \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1531     { \_unravel_input_split_end: \_unravel_input_split_auxiii:w #1 }
1532     \group_begin:
1533       \cs_set:Npn \_unravel_input_split_auxiii:w
1534         ##1 \_unravel_input_split_end: { + 1 }
1535       \int_gset:Nn \g__unravel_input_int
1536         { 0 \l__unravel_input_tmpa_tl \_unravel_input_split_end: }
1537     \group_end:
1538     \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1539     \l__unravel_input_tmpa_tl \_unravel_input_split_end:
1540   }
1541 }
1542 \cs_new:Npn \_unravel_input_split_end: { }
1543 \cs_new_protected:Npn \_unravel_input_split_auxiii:w
1544   #1 \_unravel_input_split_end:
1545 {
1546   \gtl_gclear_new:c
1547   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1548   \gtl_gset:cn
1549   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1550   \int_gdecr:N \g__unravel_input_tmpa_int
1551 }

```

(End definition for _unravel_input_split:.)

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

```

1552 \cs_new_protected:Npn \_unravel_input_gset:n

```

```

1553 {
1554   \int_gzero:N \g__unravel_input_int
1555   \__unravel_back_input:n
1556 }

```

(End definition for __unravel_input_gset:n.)

__unravel_input_get:N

```

1557 \cs_new_protected:Npn \__unravel_input_get:N #1
1558 {
1559   \__unravel_input_if_empty:TF
1560   { \gtl_set:Nn #1 { \q_no_value } }
1561   {
1562     \gtl_get_left:cN
1563     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1564   }
1565 }

```

(End definition for __unravel_input_get:N.)

__unravel_input_get_left:N

```

\__unravel_input_get_left_aux:nN
\l__unravel_input_get_left_tl
1566 \tl_new:N \l__unravel_input_get_left_tl
1567 \cs_new_protected:Npn \__unravel_input_get_left:N #1
1568 {
1569   \tl_clear:N #1
1570   \exp_args:NV \__unravel_input_get_left_aux:nN \g__unravel_input_int #1
1571 }
1572 \cs_new_protected:Npn \__unravel_input_get_left_aux:nN #1#2
1573 {
1574   \int_compare:nNnF {#1} = 0
1575   {
1576     \tl_set:Nx \l__unravel_input_get_left_tl
1577     { \gtl_left_tl:c { g__unravel_input_#1_gtl } }
1578     \tl_concat:NNN #2 #2 \l__unravel_input_get_left_tl
1579     \gtl_if_tl:cT { g__unravel_input_#1_gtl }
1580     {
1581       \exp_args:Nf \__unravel_input_get_left_aux:nN
1582       { \int_eval:n { #1 - 1 } } #2
1583     }
1584   }
1585 }

```

(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.

```

1586 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1587 {
1588   \__unravel_input_if_empty:TF
1589   { \gtl_set:Nn #1 { \q_no_value } }
1590   {
1591     \gtl_gpop_left:cN
1592     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1593   }
1594 }

```

(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.

```
1595 \cs_new_protected:Npn \_unravel_input_merge:
1596 {
1597   \int_gdecr:N \g__unravel_input_int
1598   \gtl_gconcat:ccc
1599     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1600     { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1601     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1602   \gtl_gclear:c
1603     { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1604 }
```

(End definition for `_unravel_input_merge:.`)

`_unravel_input_gpop_item:N` 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:N` is the correct one, which we return. Otherwise, merge the top two levels and repeat.

`_unravel_input_gpop_item_aux:NN`

```
1605 \prg_new_protected_conditional:Npnn \_unravel_input_gpop_item:N #1 { F }
1606 {
1607   \int_compare:nNnTF \g__unravel_input_int = 0
1608     { \prg_return_false: }
1609     {
1610       \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1611         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1612     }
1613 }
1614 \cs_new_protected:Npn \_unravel_input_gpop_item_aux:NN #1#2
1615 {
1616   \gtl_gpop_left_item:NNTF #1#2
1617     { \prg_return_true: }
1618     {
1619       \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
1620         { \prg_return_false: }
1621         {
1622           \int_compare:nNnTF \g__unravel_input_int = 1
1623             { \prg_return_false: }
1624             {
1625               \_unravel_input_merge:
1626               \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1627                 {
1628                   g__unravel_input_
1629                   \int_use:N \g__unravel_input_int _gtl
1630                 }
1631               #2
1632             }
1633         }
1634     }
1635 }
```

(End definition for `_unravel_input_gpop_item:NTF` and `_unravel_input_gpop_item_aux:NN`.)

`_unravel_input_gpop_tl:N`

```

1636 \cs_new_protected:Npn \_unravel_input_gpop_tl:N #1
1637   { \tl_clear:N #1 \_unravel_input_gpop_tl_aux:N #1 }
1638 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:N #1
1639   {
1640     \int_compare:nNnF \g__unravel_input_int = 0
1641     {
1642       \exp_args:Nc \_unravel_input_gpop_tl_aux:NN
1643         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1644     }
1645   }
1646 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:NN #1#2
1647   {
1648     \gtl_if_tl:NTF #1
1649     {
1650       \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1651       \gtl_gclear:N #1
1652       \int_gdecr:N \g__unravel_input_int
1653       \_unravel_input_gpop_tl_aux:N #2
1654     }
1655     {
1656       \int_compare:nNnTF \g__unravel_input_int > 1
1657       { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
1658       { \use_i:nn }
1659       {
1660         \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1661         \gtl_gpop_left_tl:N #1
1662       }
1663       {
1664         \_unravel_input_merge:
1665         \_unravel_input_gpop_tl_aux:N #2
1666       }
1667     }
1668   }

```

(End definition for `_unravel_input_gpop_tl:N`.)

`_unravel_back_input:n`
`_unravel_back_input:x`

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.

```

1669 \cs_new_protected:Npn \_unravel_back_input:n
1670   {
1671     \int_gincr:N \g__unravel_input_int
1672     \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1673     \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1674   }
1675 \cs_generate_variant:Nn \_unravel_back_input:n { V , o }
1676 \cs_new_protected:Npn \_unravel_back_input:x
1677   { \_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.

```

1678 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1679 {
1680   \gtl_if_tl:NTF #1
1681   { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1682   {
1683     \gtl_gconcat:cNc
1684     { g__unravel_input_ \int_use:N \g__unravel_input_int_gtl }
1685     #1
1686     { g__unravel_input_ \int_use:N \g__unravel_input_int_gtl }
1687   }
1688 }

```

(End definition for __unravel_back_input_gtl:N.)

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

```

1689 \cs_new_protected:Npn \__unravel_back_input:
1690 { \__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.

```

1691 \cs_new_protected:Npn \__unravel_back_input_tl_o:
1692 {
1693   \tl_set:Nx \l__unravel_tmpa_tl
1694   { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1695   \__unravel_back_input:V \l__unravel_tmpa_tl
1696   \__unravel_print_expansion:x
1697   { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1698 }

```

(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.

```

1699 \cs_new_protected:Npn \__unravel_insert_relax:
1700 {
1701   \__unravel_back_input:
1702   \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1703   \__unravel_back_input:
1704   \__unravel_print_action:
1705 }

```

(End definition for __unravel_insert_relax:.)

`__unravel_insert_group_begin_error:`

```

1706 \cs_new_protected:Npn \__unravel_insert_group_begin_error:
1707 {
1708   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
1709   \__unravel_back_input:

```

```

1710 \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1711 \__unravel_back_input:
1712 \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
1713 \__unravel_print_action:
1714 }

```

(End definition for __unravel_insert_group_begin_error:.)

__unravel_insert_dollar_error:

```

1715 \cs_new_protected:Npn \__unravel_insert_dollar_error:
1716 {
1717   \__unravel_back_input:
1718   \__unravel_back_input:n { $ } % $
1719   \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
1720   \__unravel_print_action:
1721 }

```

(End definition for __unravel_insert_dollar_error:.)

2.5.3 Macro calls

__unravel_macro_prefix:N
__unravel_macro_parameter:N
__unravel_macro_replacement:N

```

1722 \use:x
1723 {
1724   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
1725   {
1726     \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
1727     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1728     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1729     \exp_not:N \q_stop
1730   }
1731   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
1732   \exp_not:n {#1} \tl_to_str:n { : } \exp_not:n { #2 -> }
1733   \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1734   { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
1735 }
1736 \cs_new:Npn \__unravel_macro_prefix:N #1
1737 { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1738 \cs_new:Npn \__unravel_macro_parameter:N #1
1739 { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1740 \cs_new:Npn \__unravel_macro_replacement:N #1
1741 { \__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:
__unravel_macro_call_safe:
__unravel_macro_call_quick:
__unravel_macro_call_quick_loop:NNN
__unravel_macro_call_quick_runaway:Nw

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.

```

1742 \cs_new_protected:Npn \__unravel_macro_call:
1743 {
1744   \bool_if:NTF \g__unravel_speedup_macros_bool
1745   {

```

```

1746     \tl_set:Nx \l__unravel_tmpa_tl
1747     {^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1748     \__unravel_tl_if_in:ooTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1749     { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1750   }
1751   { \__unravel_macro_call_safe: }
1752   \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1753   \__unravel_print_expansion:
1754 }
1755 \cs_new_protected:Npn \__unravel_macro_call_safe:
1756 {
1757   \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1758   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1759 }
1760 \cs_new_protected:Npn \__unravel_macro_call_quick:
1761 {
1762   \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1763   { ? \use_none_delimit_by_q_stop:w } \q_stop
1764 }
1765 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1766 {
1767   \use_none:n #2
1768   \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1769   { \__unravel_macro_call_quick_runaway:Nw #3 }
1770   \tl_put_right:Nx \l__unravel_head_tl
1771   { { \exp_not:V \l__unravel_tmpa_tl } }
1772   \__unravel_macro_call_quick_loop:NNN
1773   #3
1774 }
1775 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1776 {
1777   \__unravel_error:nxxxx { runaway-macro-parameter }
1778   { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1779 }

```

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

```

1780 \cs_new_protected:Npn \__unravel_expand_do:N
1781 {
1782   \__unravel_set_action_text:
1783   \bool_if:NT \g__unravel_internal_debug_bool
1784     {
1785     \__unravel_set_cmd:
1786     \__unravel_exp_args:Nx \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_in
1787     }
1788   \token_if_macro:NTF \l__unravel_head_token
1789     { \__unravel_macro_call: }
1790     { \__unravel_expand_nonmacro: }
1791   }

```

(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.

```

1792 \cs_new_protected:Npn \__unravel_expand_nonmacro:
1793 {
1794   \__unravel_set_cmd_aux_meaning:
1795   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1796     {
1797     \cs_if_exist_use:cF
1798     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1799     { \__unravel_error:nxxxx { internal } { expandable } { } { } { } }
1800   }
1801   {
1802   \__unravel_error:nxxxx { unknown-primitive }
1803   { \l__unravel_head_meaning_tl } { } { } { }
1804   \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1805   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1806   \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1807   \__unravel_print_expansion:
1808   }
1809 }

```

(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:`.

```

1810 \cs_new_protected:Npn \__unravel_get_x_next:
1811 {
1812   \__unravel_get_next:
1813   \__unravel_token_if_expandable:NT \l__unravel_head_token
1814     { \__unravel_expand_do:N \__unravel_get_x_next: }
1815 }

```

(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.

```
1816 \cs_new_protected:Npn \_unravel_get_x_or_protected:
1817   {
1818     \_unravel_get_next:
1819     \_unravel_token_if_protected:NF \l__unravel_head_token
1820     { \_unravel_expand_do:N \_unravel_get_x_or_protected: }
1821   }
```

(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).

`_unravel_get_token_x:`

```
1822 \cs_new_protected:Npn \_unravel_get_token_xdef:
1823   {
1824     \_unravel_get_next:
1825     \_unravel_token_if_protected:NF \l__unravel_head_token
1826     { \_unravel_expand_do:N \_unravel_get_token_xdef: }
1827   }
1828 \cs_new_protected:Npn \_unravel_get_token_x:
1829   {
1830     \_unravel_get_next:
1831     \_unravel_token_if_protected:NF \l__unravel_head_token
1832     { \_unravel_expand_do:N \_unravel_get_token_x: }
1833   }
```

(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.

```
1834 \cs_new_protected:Npn \_unravel_get_x_non_blank:
1835   {
1836     \_unravel_get_x_next:
1837     \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
1838     { \_unravel_get_x_non_blank: }
1839   }
```

(End definition for `_unravel_get_x_non_blank:`.)

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

```
1840 \cs_new_protected:Npn \_unravel_get_x_non_relax:
1841   {
1842     \_unravel_get_x_next:
1843     \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
1844     { \_unravel_get_x_non_relax: }
1845     {
1846       \token_if_eq_meaning:NNTF \l__unravel_head_token \_unravel_special_relax:
1847       { \_unravel_get_x_non_relax: }
1848     }
```

```

1848     {
1849         \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1850         { \__unravel_get_x_non_relax: }
1851     }
1852 }
1853 }

```

(End definition for `__unravel_get_x_non_relax:`.)

`__unravel_skip_optional_space:`

```

1854 \cs_new_protected:Npn \__unravel_skip_optional_space:
1855 {
1856     \__unravel_get_x_next:
1857     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1858     { \__unravel_back_input: }
1859 }

```

(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.

```

1860 \cs_new_protected:Npn \__unravel_scan_optional_equals:
1861 {
1862     \__unravel_get_x_non_blank:
1863     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1864     { \__unravel_prev_input:n { = } }
1865     {
1866         \__unravel_prev_input_silent:n { = }
1867         \__unravel_back_input:
1868     }
1869 }

```

(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:`.

```

1870 \cs_new_protected:Npn \__unravel_scan_left_brace:
1871 {
1872     \__unravel_get_x_non_relax:
1873     \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
1874     {
1875         \__unravel_insert_group_begin_error:
1876         \__unravel_get_next:
1877     }
1878 }

```

(End definition for `__unravel_scan_left_brace:`.)

`__unravel_scan_keyword:n` 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

```

\__unravel_scan_keyword:nTF
  \__unravel_scan_keyword_loop:NNN
\__unravel_scan_keyword_test:NNTF
  \__unravel_scan_keyword_true:
  \__unravel_scan_keyword_false:w

```

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

1879 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1880 { \__unravel_scan_keyword:nTF {#1} { } { } }
1881 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1882 { T , F , TF }
1883 {
1884   \__unravel_prev_input_gpush_gtl:
1885   \__unravel_scan_keyword_loop:NNN \c_true_bool
1886   #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1887 }
1888 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1889 {
1890   \quark_if_recursion_tail_stop_do:nn {#2}
1891   { \__unravel_scan_keyword_true: }
1892   \quark_if_recursion_tail_stop_do:nn {#3}
1893   { \__unravel_error:nxxxx { internal } { odd-keyword-length } { } { } { } }
1894   \__unravel_get_x_next:
1895   \__unravel_scan_keyword_test:NNTF #2#3
1896   {
1897     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1898     \__unravel_scan_keyword_loop:NNN \c_false_bool
1899   }
1900   {
1901     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1902     { \__unravel_scan_keyword_false:w }
1903     \bool_if:NF #1
1904     { \__unravel_scan_keyword_false:w }
1905     \__unravel_scan_keyword_loop:NNN #1#2#3
1906   }
1907 }
1908 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1909 { TF }
1910 {

```

```

1911  \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
1912  { \prg_return_false: }
1913  {
1914    \str_if_eq:eeTF
1915    { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1916    { \prg_return_true: }
1917    {
1918      \str_if_eq:eeTF
1919      { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
1920      { \prg_return_true: }
1921      { \prg_return_false: }
1922    }
1923  }
1924 }
1925 \cs_new_protected:Npn \__unravel_scan_keyword_true:
1926 {
1927   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1928   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1929   \prg_return_true:
1930 }
1931 \cs_new_protected:Npn \__unravel_scan_keyword_false:w
1932 #1 \q_recursion_stop
1933 {
1934   \__unravel_back_input:
1935   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1936   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1937   \prg_return_false:
1938 }

```

(End definition for __unravel_scan_keyword:n and others.)

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

```

1939 \cs_new_protected:Npn \__unravel_scan_to:
1940 {
1941   \__unravel_scan_keyword:nF { tTo0 }
1942   {
1943     \__unravel_error:nnnnn { missing-to } { } { } { } { }
1944     \__unravel_prev_input:n { to }
1945   }
1946 }

```

(End definition for __unravel_scan_to:.)

`__unravel_scan_font_ident:` Find a font identifier.

```

1947 \cs_new_protected:Npn \__unravel_scan_font_ident:
1948 {
1949   \__unravel_get_x_non_blank:
1950   \__unravel_set_cmd:
1951   \int_case:nnF \l__unravel_head_cmd_int
1952   {
1953     { \__unravel_tex_use:n { def_font } }
1954     { \__unravel_prev_input:V \l__unravel_head_t1 }
1955     { \__unravel_tex_use:n { letterspace_font } }
1956     { \__unravel_prev_input:V \l__unravel_head_t1 }
1957     { \__unravel_tex_use:n { pdf_copy_font } }

```



```

1958         { \_unravel_prev_input:V \l\_unravel_head_tl }
1959     { \_unravel_tex_use:n { set_font } }
1960     { \_unravel_prev_input:V \l\_unravel_head_tl }
1961     { \_unravel_tex_use:n { def_family } }
1962     {
1963         \_unravel_prev_input:V \l\_unravel_head_tl
1964         \_unravel_scan_int:
1965     }
1966 }
1967 {
1968     \_unravel_error:nnnnn { missing-font-id } { } { } { } { }
1969     \_unravel_back_input:
1970     \_unravel_prev_input:n { \_unravel_nullfont: }
1971 }
1972 }

```

(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).

```

1973 \cs_new_protected:Npn \_unravel_scan_font_int:
1974 {
1975     \int_case:nnF \l\_unravel_head_char_int
1976     {
1977         { 0 } { \_unravel_scan_font_ident: }
1978         { 1 } { \_unravel_scan_font_ident: }
1979         { 6 } { \_unravel_scan_font_ident: }
1980     }
1981     { \_unravel_scan_font_ident: \_unravel_scan_int: }
1982 }

```

(End definition for _unravel_scan_font_int:.)

_unravel_scan_font_dimen: Find operands for \fontdimen.

```

1983 \cs_new_protected:Npn \_unravel_scan_font_dimen:
1984 {
1985     \_unravel_scan_int:
1986     \_unravel_scan_font_ident:
1987 }

```

(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).

```

1988 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1989 {
1990   \__unravel_set_cmd:
1991   \__unravel_set_action_text:
1992   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1993   \exp_after:wN \__unravel_scan_something_aux:nwn
1994   \l__unravel_tmpa_tl \q_stop {#1}
1995 }
1996 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1997 {
1998   \int_compare:nT { #3 < 4 <= #1 }
1999   {
2000     \__unravel_back_input:
2001     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2002     \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
2003     \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
2004     \__unravel_break:w
2005   }
2006   \int_compare:nNnT {#1} = { 8 }
2007   {
2008     \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
2009     \__unravel_scan_something_internal_auxii:nn 0 { 0 }
2010     \__unravel_break:w
2011   }
2012   \tl_if_empty:nF {#2}
2013   {
2014     \__unravel_prev_input_gpush:N \l__unravel_head_tl
2015     \__unravel_print_action:
2016     #2
2017     \__unravel_prev_input_gpop:N \l__unravel_head_tl

```

```

2018     }
2019     \int_compare:nNnTF {#3} < { 4 }
2020     { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
2021     { \tl_set:Nx \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
2022     \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
2023     \__unravel_break_point:
2024     \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
2025     }
2026     \cs_new_protected:Npn \__unravel_scan_something_internal_auxii:nn #1#2
2027     {
2028     \__unravel_prev_input_silent:n {#1}
2029     \__unravel_set_action_text:
2030     \__unravel_set_action_text:x
2031     { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
2032     \int_gset:Nn \g__unravel_val_level_int {#2}
2033     }
2034     \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.

```

2035     \cs_new:Npn \__unravel_thing_case:
2036     {
2037     \int_case:nnF \l__unravel_head_cmd_int
2038     {
2039     { 68 } { 0 } % char_given
2040     { 69 } { 0 } % math_given
2041     { 70 } { \__unravel_thing_last_item: } % last_item
2042     { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
2043     { 72 } { 5 } % assign_toks
2044     { 73 } { 0 } % assign_int
2045     { 74 } { 1 } % assign_dimen
2046     { 75 } { 2 } % assign_glue
2047     { 76 } { 3 } % assign_mu_glue
2048     { 77 } { 1 \__unravel_scan_font_dimen: } % assign_font_dimen
2049     { 78 } { 0 \__unravel_scan_font_int: } % assign_font_int
2050     { 79 } { \__unravel_thing_set_aux: } % set_aux
2051     { 80 } { 0 } % set_prev_graf
2052     { 81 } { 1 } % set_page_dimen
2053     { 82 } { 0 } % set_page_int
2054     { 83 } { 1 \__unravel_scan_int: } % set_box_dimen
2055     { 84 } { 0 \__unravel_scan_int: } % set_shape
2056     { 85 } { 0 \__unravel_scan_int: } % def_code
2057     { 86 } { 4 \__unravel_scan_int: } % def_family
2058     { 87 } { 4 } % set_font
2059     { 88 } { 4 } % def_font
2060     { 89 } { \__unravel_thing_register: } % register
2061     {101 } { 4 } % letterspace_font
2062     {102 } { 4 } % pdf_copy_font

```

```

2063     }
2064     { 8 }
2065 }
2066 \cs_new:Npn \__unravel_thing_set_aux:
2067 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
2068 \cs_new:Npn \__unravel_thing_last_item:
2069 {
2070   \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
2071   {
2072     \int_case:nnF \l__unravel_head_char_int
2073     {
2074       { 1 } { 1 } % lastkern
2075       { 2 } { 2 } % lastskip
2076     }
2077     { 0 } % other integer parameters
2078   }
2079   {
2080     \int_case:nnF \l__unravel_head_char_int
2081     {
2082       { 26 } { 0 \__unravel_scan_normal_glue: } % gluestretchorder
2083       { 27 } { 0 \__unravel_scan_normal_glue: } % glueshrinkorder
2084       { 28 } % fontcharwd
2085       { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2086       { 29 } % fontcharht
2087       { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2088       { 30 } % fontchardp
2089       { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2090       { 31 } % fontcharic
2091       { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
2092       { 32 } { 1 \__unravel_scan_int: } % parshapelength
2093       { 33 } { 1 \__unravel_scan_int: } % parshapeindent
2094       { 34 } { 1 \__unravel_scan_int: } % parshapedimen
2095       { 35 } { 1 \__unravel_scan_normal_glue: } % gluestretch
2096       { 36 } { 1 \__unravel_scan_normal_glue: } % glueshrink
2097       { 37 } { 2 \__unravel_scan_mu_glue: } % mutoglu
2098       { 38 } { 3 \__unravel_scan_normal_glue: } % gluetomu
2099       { 39 } % numepr
2100       { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
2101       { 40 } % dimexpr
2102       { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
2103       { 41 } % glueexpr
2104       { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
2105       { 42 } % muexpr
2106       { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
2107     }
2108     { }
2109   }
2110 }
2111 \cs_new:Npn \__unravel_thing_register:
2112 {
2113   \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
2114   \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
2115   { \__unravel_scan_int: }
2116 }

```

(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.

```

2117 \cs_new_protected:Npn \_unravel_scan_toks_register:
2118   {
2119     \int_compare:nNnT \l__unravel_head_char_int = 0
2120     { \_unravel_scan_int: }
2121   }

```

(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.

```

2122 \cs_new_protected:Npn \_unravel_thing_use_get:nnNN #1#2#3#4
2123   {
2124     \int_compare:nNnTF {#2} < { 3 }
2125     {
2126       \int_compare:nNnT {#1} = { 3 }
2127       { \_unravel_tex_error:nV { incompatible-units } #3 }
2128       \tl_set:Nx #4
2129       {
2130         \int_case:nn { \int_min:nn {#1} {#2} }
2131         {
2132           { 0 } \int_eval:n
2133           { 1 } \dim_eval:n
2134           { 2 } \skip_eval:n
2135         }
2136         { \int_compare:nNnT {#1} = { 3 } \tex_mutoglupe:D #3 }
2137       }
2138     }
2139     {
2140       \int_case:nnF {#1}
2141       {
2142         { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
2143         { 3 } { \tl_set:Nx #4 { \muskip_eval:n {#3} } }
2144       }
2145       {
2146         \_unravel_tex_error:nV { incompatible-units } #3
2147         \tl_set:Nx #4 { \muskip_eval:n { \tex_gluetomu:D #3 } }
2148       }
2149     }
2150   }

```

(End definition for `_unravel_thing_use_get:nnNN.`)

```

\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
2151 \cs_new_protected:Npn \_unravel_scan_expr:N #1
2152   { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
2153 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
2154   {
2155     \_unravel_get_x_non_blank:
2156     \_unravel_scan_factor:N #1
2157     \_unravel_scan_expr_op:NN #1#2

```

```

2158 }
2159 \cs_new_protected:Npn \__unravel_scan_expr_op:NN #1#2
2160 {
2161   \__unravel_get_x_non_blank:
2162   \tl_case:NnF \l__unravel_head_tl
2163   {
2164     \c__unravel_plus_tl
2165     {
2166       \__unravel_prev_input:V \l__unravel_head_tl
2167       \__unravel_scan_expr_aux:NN #1#2
2168     }
2169     \c__unravel_minus_tl
2170     {
2171       \__unravel_prev_input:V \l__unravel_head_tl
2172       \__unravel_scan_expr_aux:NN #1#2
2173     }
2174     \c__unravel_times_tl
2175     {
2176       \__unravel_prev_input:V \l__unravel_head_tl
2177       \__unravel_get_x_non_blank:
2178       \__unravel_scan_factor:N \__unravel_scan_int:
2179       \__unravel_scan_expr_op:NN #1#2
2180     }
2181     \c__unravel_over_tl
2182     {
2183       \__unravel_prev_input:V \l__unravel_head_tl
2184       \__unravel_get_x_non_blank:
2185       \__unravel_scan_factor:N \__unravel_scan_int:
2186       \__unravel_scan_expr_op:NN #1#2
2187     }
2188     \c__unravel_rp_tl
2189     {
2190       \bool_if:NTF #2
2191       { \__unravel_prev_input:V \l__unravel_head_tl }
2192       { \__unravel_back_input: }
2193     }
2194   }
2195   {
2196     \bool_if:NTF #2
2197     {
2198       \__unravel_error:nmnn { missing-rparen } { } { } { } { }
2199       \__unravel_back_input:
2200       \__unravel_prev_input:V \c__unravel_rp_tl
2201     }
2202     {
2203       \token_if_eq_meaning:NnF \l__unravel_head_token \scan_stop:
2204       { \__unravel_back_input: }
2205     }
2206   }
2207 }
2208 \cs_new_protected:Npn \__unravel_scan_factor:N #1
2209 {
2210   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2211   {

```

```

2212     \_unravel_prev_input:V \l_unravel_head_tl
2213     \_unravel_scan_expr_aux:NN #1 \c_true_bool
2214   }
2215   {
2216     \_unravel_back_input:
2217     #1
2218   }
2219 }

```

(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.

```

2220 \cs_new_protected:Npn \_unravel_scan_signs:
2221 {
2222   \_unravel_get_x_non_blank:
2223   \tl_if_eq:NNTF \l_unravel_head_tl \c_unravel_plus_tl
2224   {
2225     \_unravel_prev_input:V \l_unravel_head_tl
2226     \_unravel_scan_signs:
2227   }
2228   {
2229     \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_minus_tl
2230     {
2231       \_unravel_prev_input:V \l_unravel_head_tl
2232       \_unravel_scan_signs:
2233     }
2234   }
2235 }

```

(End definition for _unravel_scan_signs:.)

```

\_unravel_scan_int:
\_unravel_scan_int_char: 2236 \cs_new_protected:Npn \_unravel_scan_int:
\_unravel_scan_int_lq: 2237 {
  \_unravel_scan_int_explicit:n 2238   \_unravel_scan_signs:
2239   \_unravel_set_cmd:
2240   \_unravel_cmd_if_internal:TF
2241   { \_unravel_scan_something_internal:n { 0 } }
2242   { \_unravel_scan_int_char: }
2243 }
2244 \cs_new_protected:Npn \_unravel_scan_int_char:
2245 {
2246   \tl_case:NnF \l_unravel_head_tl
2247   {
2248     \c_unravel_lq_tl { \_unravel_scan_int_lq: }
2249     \c_unravel_rq_tl
2250     {
2251       \_unravel_prev_input:V \l_unravel_head_tl
2252       \_unravel_get_x_next:
2253       \_unravel_scan_int_explicit:Nn \c_false_bool { ' }
2254     }
2255     \c_unravel_dq_tl
2256     {

```

```

2257         \_unravel_prev_input:V \l__unravel_head_tl
2258         \_unravel_get_x_next:
2259         \_unravel_scan_int_explicit:Nn \c_false_bool { " }
2260     }
2261 }
2262 { \_unravel_scan_int_explicit:Nn \c_false_bool { } }
2263 }
2264 \cs_new_protected:Npn \_unravel_scan_int_lq:
2265 {
2266     \_unravel_get_next:
2267     \_unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2268     {
2269         \tl_set:Nx \l__unravel_head_tl
2270             { \_unravel_token_to_char:N \l__unravel_head_token }
2271     }
2272     \tl_set:Nx \l__unravel_tmpa_tl
2273         { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
2274     \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2275     \_unravel_print_action:x
2276         { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2277     \_unravel_skip_optional_space:
2278 }
2279 \cs_new_protected:Npn \_unravel_scan_int_explicit:Nn #1#2
2280 {
2281     \if_int_compare:w 1
2282         < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2283         \exp_after:wN \use_i:nn
2284     \else:
2285         \exp_after:wN \use_ii:nn
2286     \fi:
2287     {
2288         \_unravel_prev_input:V \l__unravel_head_tl
2289         \_unravel_get_x_next:
2290         \_unravel_scan_int_explicit:Nn \c_true_bool {#2}
2291     }
2292     {
2293         \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2294         { \_unravel_back_input: }
2295         \bool_if:NF #1
2296         {
2297             \_unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2298             \_unravel_prev_input:n { 0 }
2299         }
2300     }
2301 }

```

(End definition for _unravel_scan_int: and others.)

_unravel_scan_normal_dimen:

```

2302 \cs_new_protected:Npn \_unravel_scan_normal_dimen:
2303     { \_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`.

Ideally, `__unravel_scan_inf_unit_loop:` would produce an `unravel` error when reaching the third “L”, rather than letting TeX produce the error later on.

```

2304 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2305 {
2306   \__unravel_scan_signs:
2307   \__unravel_prev_input_gpush:
2308   \__unravel_set_cmd:
2309   \__unravel_cmd_if_internal:TF
2310   {
2311     \int_compare:nNnTF {#1} = { 3 }
2312     { \__unravel_scan_something_internal:n { 3 } }
2313     { \__unravel_scan_something_internal:n { 1 } }
2314     \int_compare:nNnT \g__unravel_val_level_int = { 0 }
2315     { \__unravel_scan_dim_unit:nN {#1} #2 }
2316   }
2317   { \__unravel_scan_dimen_char:nN {#1} #2 }
2318   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2319   \__unravel_prev_input_silent:V \l__unravel_head_tl
2320 }
2321 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
2322 {
2323   \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2324   { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2325   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
2326   {
2327     \__unravel_prev_input:n { . }
2328     \__unravel_scan_decimal_loop:
2329   }
2330   {
2331     \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl
2332     {
2333       \__unravel_back_input:
2334       \__unravel_scan_int:
2335       \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2336       { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2337       \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
2338       {
2339         \__unravel_input_gpop:N \l__unravel_tmpb_gtl
2340         \__unravel_prev_input:n { . }
2341         \__unravel_scan_decimal_loop:
2342       }
2343     }
2344     {
2345       \__unravel_back_input:
2346       \__unravel_scan_int:
2347     }
2348   }

```

```

2349   \__unravel_scan_dim_unit:nN {#1} #2
2350 }
2351 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
2352 {
2353   \bool_if:NT #2
2354   {
2355     \__unravel_scan_keyword:nT { fFillL }
2356     {
2357       \__unravel_scan_inf_unit_loop:
2358       \__unravel_break:w
2359     }
2360   }
2361   \__unravel_get_x_non_blank:
2362   \__unravel_set_cmd:
2363   \__unravel_cmd_if_internal:TF
2364   {
2365     \__unravel_prev_input_gpush:
2366     \__unravel_scan_something_internal:n {#1}
2367     \int_compare:nNnTF \g__unravel_val_level_int = { 0 }
2368     { \__unravel_prev_input_join_get:nnN {#1} { sp } \l__unravel_tmpa_tl }
2369     { \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl }
2370     \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2371     \exp_after:wN \use_none:n \__unravel_break:w
2372   }
2373   { }
2374   \__unravel_back_input:
2375   \int_compare:nNnT {#1} = { 3 }
2376   {
2377     \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2378     \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
2379     \__unravel_prev_input:n { mu }
2380     \__unravel_break:w
2381   }
2382   \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2383   \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2384   \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2385   \__unravel_scan_keyword:nT { tTrRuUeE }
2386   { \__unravel_prepare_mag: }
2387   \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2388   \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2389   \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2390   \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }
2391   \__unravel_scan_keyword:nT { mMmM } { \__unravel_break:w }
2392   \__unravel_scan_keyword:nT { bBpP } { \__unravel_break:w }
2393   \__unravel_scan_keyword:nT { dDdD } { \__unravel_break:w }
2394   \__unravel_scan_keyword:nT { cCcC } { \__unravel_break:w }
2395   \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
2396   \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
2397   \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
2398   \__unravel_tex_error:nV { missing-pt } \l__unravel_head_tl
2399   \__unravel_prev_input:n { pt }
2400   \__unravel_break_point:
2401   \__unravel_skip_optional_space:
2402 }

```

```

2403 \cs_new_protected:Npn \__unravel_scan_inf_unit_loop:
2404   { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
2405 \cs_new_protected:Npn \__unravel_scan_decimal_loop:
2406   {
2407     \__unravel_get_x_next:
2408     \tl_if_empty:NTF \l__unravel_head_tl
2409       { \use_ii:nn }
2410       { \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl }
2411       {
2412         \__unravel_prev_input:V \l__unravel_head_tl
2413         \__unravel_scan_decimal_loop:
2414       }
2415       {
2416         \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
2417         { \__unravel_back_input: }
2418         \__unravel_prev_input_silent:n { ~ }
2419       }
2420   }

```

(End definition for __unravel_scan_dimen:nN.)

```

\__unravel_scan_normal_glue:
  \__unravel_scan_mu_glue:
2421 \cs_new_protected:Npn \__unravel_scan_normal_glue:
2422   { \__unravel_scan_glue:n { 2 } }
2423 \cs_new_protected:Npn \__unravel_scan_mu_glue:
2424   { \__unravel_scan_glue:n { 3 } }

```

(End definition for __unravel_scan_normal_glue: and __unravel_scan_mu_glue:.)

```

\__unravel_scan_glue:n
2425 \cs_new_protected:Npn \__unravel_scan_glue:n #1
2426   {
2427     \__unravel_prev_input_gpush:
2428     \__unravel_scan_signs:
2429     \__unravel_prev_input_gpush:
2430     \__unravel_set_cmd:
2431     \__unravel_cmd_if_internal:TF
2432     {
2433       \__unravel_scan_something_internal:n {#1}
2434       \int_case:nnF \g__unravel_val_level_int
2435         {
2436           { 0 } { \__unravel_scan_dim_unit:nN {#1} \c_false_bool }
2437           { 1 } { }
2438         }
2439       { \__unravel_break:w }
2440     }
2441     { \__unravel_back_input: \__unravel_scan_dimen:nN {#1} \c_false_bool }
2442     \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
2443     \__unravel_prev_input_gpush:
2444     \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2445     \__unravel_scan_keyword:nT { pPlLuUsS }
2446     { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2447     \__unravel_scan_keyword:nT { mMiInNuUsS }
2448     { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2449     \__unravel_break_point:

```

```

2450     \_unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
2451     \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2452   }

```

(End definition for _unravel_scan_glue:n.)

_unravel_scan_file_name:

```

2453 \cs_new_protected:Npn \_unravel_scan_file_name:
2454 {
2455   \bool_gset_true:N \g__unravel_name_in_progress_bool
2456   \_unravel_get_x_non_blank:
2457   \_unravel_scan_file_name_loop:
2458   \bool_gset_false:N \g__unravel_name_in_progress_bool
2459   \_unravel_prev_input_silent:n { ~ }
2460 }
2461 \cs_new_protected:Npn \_unravel_scan_file_name_loop:
2462 {
2463   \_unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2464   { \_unravel_back_input: }
2465   {
2466     \tl_set:Nx \l__unravel_tmpa_tl
2467     { \_unravel_token_to_char:N \l__unravel_head_token }
2468     \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2469     {
2470       \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2471       \_unravel_get_x_next:
2472       \_unravel_scan_file_name_loop:
2473     }
2474   }
2475 }

```

(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.

```

2476 \cs_new_protected:Npn \_unravel_scan_r_token:
2477 {
2478   \bool_do_while:nn
2479   { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2480   { \_unravel_get_next: }
2481   \_unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2482   {
2483     \_unravel_error:nnnnn { missing-cs } { } { } { } { }
2484     \_unravel_back_input:
2485     \tl_set:Nn \l__unravel_head_tl { \_unravel_inaccessible:w }
2486   }
2487   \_unravel_prev_input_silent:V \l__unravel_head_tl
2488   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2489 }

```

(End definition for _unravel_scan_r_token:.)

_unravel_scan_toks_to_str:

```

2490 \cs_new_protected:Npn \_unravel_scan_toks_to_str:
2491 {

```

```

2492   \__unravel_prev_input_gpush:
2493   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2494   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2495   \__unravel_prev_input_silent:x
2496   { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2497   }

```

(End definition for __unravel_scan_toks_to_str:.)

__unravel_scan_pdf_ext_toks:

```

2498 \cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
2499 {
2500   \__unravel_prev_input_gpush:
2501   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2502   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2503   \__unravel_prev_input_silent:x
2504   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2505   }

```

(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).

```

2506 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2507 {
2508   \bool_if:NT #1 { \__unravel_scan_param: }
2509   \__unravel_scan_left_brace:
2510   \bool_if:NTF #2
2511     { \__unravel_scan_group_x:N #1 }
2512     { \__unravel_scan_group_n:N #1 }
2513   }

```

(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.

```

2514 \cs_new_protected:Npn \__unravel_scan_param:
2515 {
2516   \tl_clear:N \l__unravel_tmpa_tl
2517   \__unravel_scan_param_aux:
2518   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2519   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2520 }
2521 \cs_new_protected:Npn \__unravel_scan_param_aux:
2522 {
2523   \__unravel_get_next:
2524   \tl_concat:NNN \l__unravel_tmpa_tl
2525     \l__unravel_tmpa_tl \l__unravel_head_tl
2526   \tl_if_empty:NTF \l__unravel_head_tl
2527     { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2528   }

```

(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.

```

2529 \cs_new_protected:Npn \_unravel_scan_group_n:N #1
2530 {
2531   \gtl_set_eq:NN \l\_unravel_head_gtl \c_group_begin_gtl
2532   \_unravel_back_input:
2533   \_unravel_input_gpop_item:NF \l\_unravel_head_tl
2534   {
2535     \_unravel_error:nnnnn { runaway-text } { } { } { } { }
2536     \_unravel_exit_hard:w
2537   }
2538   \tl_set:Nx \l\_unravel_head_tl { { \exp_not:V \l\_unravel_head_tl } }
2539   \bool_if:NT #1
2540   { \tl_put_right:NV \l\_unravel_defining_tl \l\_unravel_head_tl }
2541   \_unravel_prev_input_silent:V \l\_unravel_head_tl
2542 }

```

(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.

```

2543 \cs_new_protected:Npn \_unravel_scan_group_x:N #1
2544 {
2545   \_unravel_input_gpop_tl:N \l\_unravel_head_tl
2546   \_unravel_back_input:V \l\_unravel_head_tl
2547   \bool_if:NTF #1
2548   {
2549     \_unravel_prev_input_silent:V \c_left_brace_str
2550     \tl_put_right:Nn \l\_unravel_defining_tl { { \if_false: } \fi: }
2551     \_unravel_scan_group_xdef:n { 1 }
2552   }
2553   {
2554     \_unravel_prev_input_gpush_gtl:
2555     \_unravel_prev_input_gtl:N \l\_unravel_head_gtl
2556     \_unravel_scan_group_x:n { 1 }
2557     \_unravel_prev_input_gpop_gtl:N \l\_unravel_tmpb_gtl
2558     \_unravel_prev_input_silent:x
2559     { \gtl_left_tl:N \l\_unravel_tmpb_gtl }
2560   }
2561 }

```

(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.

```

2562 \cs_new_protected:Npn \_unravel_scan_group_xdef:n #1
2563 {
2564   \_unravel_get_token_xdef:
2565   \tl_if_empty:NTF \l\_unravel_head_tl
2566   {
2567     \gtl_if_head_is_group_begin:NTF \l\_unravel_head_gtl
2568     {
2569       \_unravel_prev_input_silent:V \c_left_brace_str
2570       \tl_put_right:Nn \l\_unravel_defining_tl { { \if_false: } \fi: }

```

```

2571     \_unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2572   }
2573   {
2574     \_unravel_prev_input_silent:V \c_right_brace_str
2575     \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2576     \int_compare:nNnF {#1} = 1
2577     { \_unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2578   }
2579 }
2580 {
2581   \_unravel_prev_input_silent:V \l__unravel_head_tl
2582   \tl_put_right:Nx \l__unravel_defining_tl
2583   { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2584   \_unravel_scan_group_xdef:n {#1}
2585 }
2586 }
2587 \cs_generate_variant:Nn \_unravel_scan_group_xdef:n { f }

```

(End definition for _unravel_scan_group_xdef:n.)

_unravel_scan_group_x:n

```

2588 \cs_new_protected:Npn \_unravel_scan_group_x:n #1
2589 {
2590   \_unravel_get_token_x:
2591   \_unravel_prev_input_gtl:N \l__unravel_head_gtl
2592   \tl_if_empty:NTF \l__unravel_head_tl
2593   {
2594     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2595     { \_unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2596     {
2597       \int_compare:nNnF {#1} = 1
2598       { \_unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2599     }
2600   }
2601   { \_unravel_scan_group_x:n {#1} }
2602 }
2603 \cs_generate_variant:Nn \_unravel_scan_group_x:n { f }

```

(End definition for _unravel_scan_group_x:n.)

_unravel_scan_alt_rule:

```

2604 \cs_new_protected:Npn \_unravel_scan_alt_rule:
2605 {
2606   \_unravel_scan_keyword:nTF { wWiIdDtThH }
2607   {
2608     \_unravel_scan_normal_dimen:
2609     \_unravel_scan_alt_rule:
2610   }
2611   {
2612     \_unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2613     {
2614       \_unravel_scan_normal_dimen:
2615       \_unravel_scan_alt_rule:
2616     }
2617   }

```

```

2618         \__unravel_scan_keyword:nT { dDeEpPtThH }
2619         {
2620             \__unravel_scan_normal_dimen:
2621             \__unravel_scan_alt_rule:
2622         }
2623     }
2624 }
2625 }

```

(End definition for `__unravel_scan_alt_rule:`.)

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

```

2626 \cs_new_protected:Npn \__unravel_scan_spec:
2627 {
2628     \__unravel_scan_keyword:nTF { tTo0 } { \__unravel_scan_normal_dimen: }
2629     {
2630         \__unravel_scan_keyword:nT { sSpPrReEaAdD }
2631         { \__unravel_scan_normal_dimen: }
2632     }
2633     \__unravel_scan_left_brace:
2634 }

```

(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.

```

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

```



```

2650             { \__unravel_do_leaders_rule: }
2651             { \__unravel_do_box_error: }
2652         }
2653     { \__unravel_do_box_error: }
2654 }
2655 }

```

(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.

```

2656 \cs_new_protected:Npn \__unravel_do_box_error:
2657 {
2658     \__unravel_back_input:
2659     \__unravel_error:nnnnn { missing-box } { } { } { } { }
2660     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2661     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2662 }

```

(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.

```

2663 \cs_new_protected:Npn \__unravel_do_begin_box:N #1
2664 {
2665     \__unravel_prev_input:V \l__unravel_head_tl
2666     \int_case:nnTF \l__unravel_head_char_int
2667     {
2668         { 0 } { \__unravel_scan_int: } % box
2669         { 1 } { \__unravel_scan_int: } % copy
2670         { 2 } { } % lastbox
2671         { 3 } % vsplit
2672         {
2673             \__unravel_scan_int:
2674             \__unravel_scan_to:
2675             \__unravel_scan_normal_dimen:
2676         }
2677     }
2678     { \__unravel_do_simple_box:N #1 }
2679     { \__unravel_do_box_explicit:N #1 }
2680 }

```

(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.

```

2681 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2682 {
2683     \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }

```

```

2684     {
2685       \__unravel_prev_input_gpop:N \l__unravel_head_tl
2686       \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2687       { \__unravel_prepare_mag: }
2688       \tl_use:N \l__unravel_head_tl \scan_stop:
2689       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2690       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2691     }
2692   }

```

(End definition for `__unravel_do_simple_box:N`.)

`__unravel_do_leaders_fetch_skip:`

```

2693 \cs_new_protected:Npn \__unravel_do_leaders_fetch_skip:
2694   {
2695     \__unravel_get_x_non_relax:
2696     \__unravel_set_cmd:
2697     \int_compare:nNnTF \l__unravel_head_cmd_int
2698     = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2699     {
2700       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2701       \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2702       \__unravel_do_append_glue:
2703     }
2704     {
2705       \__unravel_back_input:
2706       \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2707       \__unravel_prev_input_gpop:N \l__unravel_head_tl
2708       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2709     }
2710   }

```

(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.

```

2711 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2712   {
2713     \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2714     { \__unravel_box_hook:N \tex_everyhbox:D }
2715     { \__unravel_box_hook:N \tex_everyvbox:D }
2716     % ^^A todo: TeX calls |normal_paragraph| here.
2717     \__unravel_scan_spec:
2718     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2719     \__unravel_set_action_text:x
2720     { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }

```

```

2721 \seq_push:Nf \l__unravel_leaders_box_seq
2722 { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2723 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2724 \gtl_gconcat:NNN \g__unravel_output_gtl
2725 \g__unravel_output_gtl \c_group_begin_gtl
2726 \tl_use:N \l__unravel_head_tl
2727 \c_group_begin_token \__unravel_box_hook_end:
2728 }

```

(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.

```

2729 \cs_new_protected:Npn \__unravel_box_hook:N #1
2730 {
2731 \tl_set:NV \l__unravel_tmpa_tl #1
2732 \str_if_eq:eeF
2733 { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2734 {
2735 \__unravel_exp_args:Nx #1
2736 {
2737 \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2738 \exp_not:V #1
2739 }
2740 }
2741 \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2742 {
2743 \exp_args:No #1 {##1}
2744 \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2745 \gtl_clear:N \l__unravel_after_group_gtl
2746 \__unravel_print_action:
2747 \__unravel_back_input:o {##1}
2748 \__unravel_set_action_text:x
2749 { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2750 \tl_if_empty:oF {##1} { \__unravel_print_action: }
2751 }
2752 }
2753 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2754 \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.

```

2755 \cs_new_protected:Npn \__unravel_do_leaders_rule:
2756 {
2757 \__unravel_prev_input:V \l__unravel_head_tl
2758 \__unravel_scan_alt_rule:
2759 \__unravel_do_leaders_fetch_skip:
2760 }

```

(End definition for `__unravel_do_leaders_rule:.`)

2.9 Paragraphs

`_unravel_charcode_if_safe:nTF`

```

2761 \prg_new_protected_conditional:Npnm \_unravel_charcode_if_safe:n #1 { TF }
2762 {
2763   \bool_if:nTF
2764   {
2765     \int_compare_p:n { #1 = '!' }
2766     || \int_compare_p:n { ' ' <= #1 <= '[' }
2767     || \int_compare_p:n { #1 = ']' }
2768     || \int_compare_p:n { ' ' <= #1 <= 'z' }
2769   }
2770   { \prg_return_true: }
2771   { \prg_return_false: }
2772 }

```

(End definition for _unravel_charcode_if_safe:nTF.)

`_unravel_char:n`
`_unravel_char:V`
`_unravel_char:x`

```

2773 \cs_new_protected:Npn \_unravel_char:n #1
2774 {
2775   \tex_char:D #1 \scan_stop:
2776   \_unravel_charcode_if_safe:nTF {#1}
2777   { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2778   {
2779     \tl_set:Nx \l__unravel_tmpa_tl
2780     { \exp_not:N \char \int_eval:n {#1} ~ }
2781   }
2782   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2783   \_unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2784 }
2785 \cs_generate_variant:Nn \_unravel_char:n { V }
2786 \cs_new_protected:Npn \_unravel_char:x
2787 { \_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`

```

2788 \cs_new_protected:Npn \_unravel_char_in_mmode:n #1
2789 {
2790   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2791   { % math active
2792     \_unravel_exp_args:NNx \gtl_set:Nn \l__unravel_head_gtl
2793     { \char_generate:nn {#1} { 12 } }
2794     \_unravel_back_input:
2795   }
2796   { \_unravel_char:n {#1} }
2797 }
2798 \cs_generate_variant:Nn \_unravel_char_in_mmode:n { V }
2799 \cs_new_protected:Npn \_unravel_char_in_mmode:x
2800 { \_unravel_exp_args:Nx \_unravel_char_in_mmode:n }

```

(End definition for _unravel_char_in_mmode:n.)

`__unravel_mathchar:n`

`__unravel_mathchar:x`

```
2801 \cs_new_protected:Npn \__unravel_mathchar:n #1
2802 {
2803   \tex_mathchar:D #1 \scan_stop:
2804   \tl_set:Nx \l__unravel_tmpa_tl
2805     { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2806   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2807   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2808 }
2809 \cs_new_protected:Npn \__unravel_mathchar:x
2810 { \__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.

```
2811 \cs_new_protected:Npn \__unravel_new_graf:N #1
2812 {
2813   \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
2814   \__unravel_everypar:w { }
2815   \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2816   \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl
2817   \__unravel_back_input:V \l__unravel_tmpa_tl
2818   \__unravel_print_action:x
2819     {
2820       \g__unravel_action_text_str \c_space_tl : ~
2821       \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2822     }
2823 }
```

(End definition for __unravel_new_graf:N.)

`__unravel_end_graf:`

```
2824 \cs_new_protected:Npn \__unravel_end_graf:
2825 { \mode_if_horizontal:T { \__unravel_normal_paragraph: } }
```

(End definition for __unravel_end_graf:.)

`__unravel_normal_paragraph:`

```
2826 \cs_new_protected:Npn \__unravel_normal_paragraph:
2827 {
2828   \tex_par:D
2829   \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2830   \__unravel_print_action:x { Paragraph~end. }
2831 }
```

(End definition for __unravel_normal_paragraph:.)

`__unravel_build_page:`

```
2832 \cs_new_protected:Npn \__unravel_build_page:
2833 {
2834 }
```

(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.

```

2835 \cs_new_protected:Npn \_unravel_handle_right_brace:
2836 {
2837   \int_compare:nTF { 1 <= \_unravel_currentgroupstype: <= 13 }
2838   {
2839     \gtl_gconcat:NNN \g__unravel_output_gtl
2840     \g__unravel_output_gtl \c_group_end_gtl
2841     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2842     \int_case:nn \_unravel_currentgroupstype:
2843     {
2844       { 1 } { \_unravel_end_simple_group: } % simple
2845       { 2 } { \_unravel_end_box_group: } % hbox
2846       { 3 } { \_unravel_end_box_group: } % adjusted_hbox
2847       { 4 } { \_unravel_end_graf: \_unravel_end_box_group: } % vbox
2848       { 5 } { \_unravel_end_graf: \_unravel_end_box_group: } % vtop
2849       { 6 } { \_unravel_end_align_group: } % align
2850       { 7 } { \_unravel_end_no_align_group: } % no_align
2851       { 8 } { \_unravel_end_output_group: } % output
2852       { 9 } { \_unravel_end_simple_group: } % math
2853       { 10 } { \_unravel_end_disc_group: } % disc
2854       { 11 } { \_unravel_end_graf: \_unravel_end_simple_group: } % insert
2855       { 12 } { \_unravel_end_graf: \_unravel_end_simple_group: } % vcenter
2856       { 13 } { \_unravel_end_math_choice_group: } % math_choice
2857     }
2858   }
2859   { % bottom_level, semi_simple, math_shift, math_left
2860     \l__unravel_head_token
2861     \_unravel_print_action:
2862   }
2863 }

```

(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.

```

2864 \cs_new_protected:Npn \_unravel_end_simple_group:
2865 {
2866   \l__unravel_head_token
2867   \_unravel_print_action:
2868 }

```

(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.

```

2869 \cs_new_protected:Npn \_unravel_end_box_group:
2870 {
2871   \seq_pop:NN \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2872   \exp_args:No \_unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
2873 }
2874 \cs_new_protected:Npn \_unravel_end_box_group_aux:n #1
2875 {

```

```

2876 \str_if_eq:eeTF {#1} { Z }
2877 { \__unravel_end_simple_group: }
2878 {
2879   \__unravel_get_x_non_relax:
2880   \__unravel_set_cmd:
2881   \int_compare:nNnTF \l__unravel_head_cmd_int
2882   = { \__unravel_tex_use:n { #1 skip } }
2883   {
2884     \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2885     \__unravel_do_append_glue:
2886   }
2887   {
2888     \__unravel_back_input:
2889     \c_group_end_token \group_begin: \group_end:
2890     \__unravel_print_action:
2891   }
2892 }
2893 }

```

(End definition for `__unravel_end_box_group:`.)

`__unravel_off_save:`

```

2894 \cs_new_protected:Npn \__unravel_off_save:
2895 {
2896   \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
2897   { % bottom-level
2898     \__unravel_error:nxxxx { extra-close }
2899     { \token_to_meaning:N \l__unravel_head_token } { } { } { }
2900   }
2901   {
2902     \__unravel_back_input:
2903     \int_case:nnF \__unravel_currentgrouptype:
2904     {
2905       { 14 } % semi_simple_group
2906       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
2907       { 15 } % math_shift_group
2908       { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2909       { 16 } % math_left_group
2910       { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2911     }
2912     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2913     \__unravel_back_input:
2914     \__unravel_error:nxxxx { off-save }
2915     { \gtl_to_str:N \l__unravel_head_gtl } { } { } { }
2916   }
2917 }

```

(End definition for `__unravel_off_save:`.)

2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n 2918 \cs_new_protected:Npn \__unravel_mode_math:n #1
\__unravel_mode_vertical:n 2919 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }

```

```

2920 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
2921   { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
2922 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
2923   {
2924     \mode_if_math:TF
2925       { \__unravel_insert_dollar_error: }
2926       { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
2927   }
2928 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
2929   {
2930     \mode_if_vertical:TF
2931       { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
2932       {#1}
2933   }

```

(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`.

```

2934 \cs_new_protected:Npn \__unravel_head_for_vmode:
2935   {
2936     \mode_if_inner:TF
2937     {
2938       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
2939       {
2940         \__unravel_error:nmnnn { hrulerule-bad-mode } { } { } { } { }
2941         \__unravel_print_action:
2942       }
2943       { \__unravel_off_save: }
2944     }
2945     {
2946       \__unravel_back_input:
2947       \gtl_set:Nn \l__unravel_head_gtl { \par }
2948       \__unravel_back_input:
2949     }
2950   }

```

(End definition for `__unravel_head_for_vmode:.`)

`__unravel_goto_inner_math:`

```

2951 \cs_new_protected:Npn \__unravel_goto_inner_math:
2952   {
2953     \__unravel_box_hook:N \tex_everymath:D
2954     $ % $
2955     \__unravel_box_hook_end:
2956   }

```

(End definition for `__unravel_goto_inner_math:.`)

`__unravel_goto_display_math:`

```

2957 \cs_new_protected:Npn \__unravel_goto_display_math:
2958   {
2959     \__unravel_box_hook:N \tex_everydisplay:D
2960     $ $
2961     \__unravel_box_hook_end:
2962   }

```


(End definition for `_unravel_goto_display_math:`.)

`_unravel_after_math:`

```
2963 \cs_new_protected:Npn \_unravel_after_math:
2964   {
2965     \mode_if_inner:TF
2966       {
2967         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2968         \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2969         $ % $
2970       }
2971       {
2972         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2973         \_unravel_get_x_next:
2974         \token_if_eq_catcode:NMF
2975           \l__unravel_head_token \c_math_toggle_token
2976           {
2977             \_unravel_back_input:
2978             \tl_set:Nn \l__unravel_head_tl { $ } % $
2979             \_unravel_error:nmnn { missing-dollar } { } { } { } { }
2980           }
2981         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2982         \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2983         $ $
2984       }
2985     \_unravel_print_action:
2986   }
```

(End definition for `_unravel_after_math:`.)

2.12 Commands

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

`_unravel_forbidden_case:`

```
2987 \cs_new_protected:Npn \_unravel_forbidden_case:
2988   { \_unravel_tex_error:nV { forbidden-case } \l__unravel_head_tl }
```

(End definition for `_unravel_forbidden_case:`.)

2.12.1 Characters: from 0 to 15

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

- `relax=0` for `\relax`.
- `begin-group_char=1` for begin-group characters (catcode 1).
- `end-group_char=2` for end-group characters (catcode 2).
- `math_char=3` for math shift (math toggle in `expl3`) characters (catcode 3).
- `tab_mark=4` for `\span`
- `alignment_char=4` for alignment tab characters (catcode 4).

- `car_ret=5` for `\cr` and `\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.

```

2989 \__unravel_new_tex_cmd:nn { relax } % 0
2990 {
2991   \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:
2992   {
2993     \exp_after:wN \__unravel_token_if_expandable:NTF \l__unravel_head_tl
2994     {
2995       \__unravel_set_action_text:x
2996       { \iow_char:N \notexpanded: \g__unravel_action_text_str }
2997     }
2998   { }
2999 }
3000 \__unravel_print_action:
3001 }

```

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.

```

3002 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
3003 {
3004   \gtl_gconcat:NNN \g__unravel_output_gtl
3005   \g__unravel_output_gtl \c_group_begin_gtl
3006   \__unravel_print_action:
3007   \l__unravel_head_token
3008   \gtl_clear:N \l__unravel_after_group_gtl
3009 }
3010 \__unravel_new_tex_cmd:nn { end-group_char } % 2
3011 { \__unravel_handle_right_brace: }

```

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

```

3012 \__unravel_new_tex_cmd:nn { math_char } % 3
3013 {
3014   \__unravel_mode_non_vertical:n
3015 }

```

```

3016     \mode_if_math:TF
3017     {
3018         \int_compare:nNnTF
3019             \__unravel_currentgrouptype: = { 15 } % math_shift_group
3020             { \__unravel_after_math: }
3021             { \__unravel_off_save: }
3022     }
3023     {
3024         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3025         \__unravel_get_next:
3026         \token_if_eq_catcode:NNTF
3027             \l__unravel_head_token \c_math_toggle_token
3028         {
3029             \mode_if_inner:TF
3030                 { \__unravel_back_input: \__unravel_goto_inner_math: }
3031                 {
3032                     \gtl_gput_right:NV
3033                     \g__unravel_output_gtl \l__unravel_head_tl
3034                     \__unravel_goto_display_math:
3035                 }
3036             }
3037             { \__unravel_back_input: \__unravel_goto_inner_math: }
3038         }
3039     }
3040 }

```

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.

```

3041 \__unravel_new_tex_cmd:nn { alignment_char } % 4
3042 { \l__unravel_head_token \__unravel_print_action: }
3043 \__unravel_new_tex_cmd:nn { car_ret } % 5
3044 { \l__unravel_head_token \__unravel_print_action: }
3045 \__unravel_new_tex_cmd:nn { macro_char } % 6
3046 { \l__unravel_head_token \__unravel_print_action: }
3047 \__unravel_new_tex_cmd:nn { superscript_char } % 7
3048 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3049 \__unravel_new_tex_cmd:nn { subscript_char } % 8
3050 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3051 \cs_new_protected:Npn \__unravel_sub_sup:
3052 {
3053     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3054     \__unravel_print_action:
3055     \__unravel_get_x_non_relax:
3056     \__unravel_set_cmd:
3057     \int_case:nnTF \l__unravel_head_cmd_int
3058     {
3059         { \__unravel_tex_use:n { the_char } }
3060         { \__unravel_prev_input:V \l__unravel_head_tl }
3061         { \__unravel_tex_use:n { other_char } }
3062         { \__unravel_prev_input:V \l__unravel_head_tl }
3063         { \__unravel_tex_use:n { char_given } }
3064         { \__unravel_prev_input:V \l__unravel_head_tl }
3065         { \__unravel_tex_use:n { char_num } }

```

```

3066     {
3067         \_unravel_prev_input:V \l_unravel_head_tl
3068         \_unravel_scan_int:
3069     }
3070 { \_unravel_tex_use:n { math_char_num } }
3071 {
3072     \_unravel_prev_input:V \l_unravel_head_tl
3073     \_unravel_scan_int:
3074 }
3075 { \_unravel_tex_use:n { math_given } }
3076 { \_unravel_prev_input:V \l_unravel_head_tl }
3077 { \_unravel_tex_use:n { delim_num } }
3078 { \_unravel_prev_input:V \l_unravel_head_tl \_unravel_scan_int: }
3079 }
3080 {
3081     \_unravel_prev_input_gpop:N \l_unravel_head_tl
3082     \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3083     \tl_use:N \l_unravel_head_tl \scan_stop:
3084 }
3085 {
3086     \_unravel_back_input:
3087     \_unravel_scan_left_brace:
3088     \_unravel_prev_input_gpop:N \l_unravel_head_tl
3089     \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3090     \gtl_gconcat:NNN \g_unravel_output_gtl
3091     \g_unravel_output_gtl \c_group_begin_gtl
3092     \tl_use:N \l_unravel_head_tl \c_group_begin_token
3093 }
3094 \_unravel_print_action:x { \tl_to_str:N \l_unravel_head_tl }
3095 }
3096 \_unravel_new_tex_cmd:nn { endv } % 9
3097 { \_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.
3098 \_unravel_new_tex_cmd:nn { blank_char } % 10
3099 {
3100     \mode_if_horizontal:T
3101     {
3102         \gtl_gput_right:Nn \g_unravel_output_gtl { ~ }
3103         \l_unravel_head_token
3104     }
3105     \_unravel_print_action:
3106 }

Letters and other characters leave vertical mode.
3107 \_unravel_new_tex_cmd:nn { the_char } % 11
3108 {
3109     \_unravel_mode_non_vertical:n
3110     {
3111         \tl_set:Nx \l_unravel_tmpa_tl
3112         { ' \_unravel_token_to_char:N \l_unravel_head_token }
3113         \mode_if_math:TF
3114         { \_unravel_char_in_mmode:V \l_unravel_tmpa_tl }

```

```

3115         { \_unravel_char:V \l\_unravel_tmpa_tl }
3116     }
3117 }
3118 \_unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12
3119 \_unravel_new_tex_cmd:nn { par_end } % 13
3120 {
3121     \_unravel_mode_non_math:n
3122     {
3123         \mode_if_vertical:TF
3124         { \_unravel_normal_paragraph: }
3125         {
3126             % if align_state<0 then off_save;
3127             \_unravel_end_graf:
3128             \mode_if_vertical:T
3129             { \mode_if_inner:F { \_unravel_build_page: } }
3130         }
3131     }
3132 }
3133 \_unravel_new_tex_cmd:nn { stop } % 14
3134 {
3135     \_unravel_mode_vertical:n
3136     {
3137         \mode_if_inner:TF
3138         { \_unravel_forbidden_case: }
3139         {
3140             % ^^A todo: unless its_all_over
3141             \int_gdecr:N \g\_unravel_ends_int
3142             \int_compare:nNnTF \g\_unravel_ends_int > 0
3143             {
3144                 \_unravel_back_input:
3145                 \_unravel_back_input:n
3146                 {
3147                     \_unravel_hbox:w to \tex_hsize:D { }
3148                     \tex_vfill:D
3149                     \tex_penalty:D - '1000000000 ~
3150                 }
3151                 \_unravel_build_page:
3152                 \_unravel_print_action:x { End-everything! }
3153             }
3154             {
3155                 \_unravel_print_outcome:
3156                 \l\_unravel_head_token
3157             }
3158         }
3159     }
3160 }
3161 \_unravel_new_tex_cmd:nn { delim_num } % 15
3162 {
3163     \_unravel_mode_math:n
3164     {
3165         \_unravel_prev_input_gpush:N \l\_unravel_head_tl
3166         \_unravel_print_action:
3167         \_unravel_scan_int:

```

```

3168     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3169     \tl_use:N \l__unravel_head_tl \scan_stop:
3170     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3171   }
3172 }

```

2.12.2 Boxes: from 16 to 31

- char_num=16 for `\char`
- math_char_num=17 for `\mathchar`
- mark=18 for `\mark` and `\marks`
- xray=19 for `\show`, `\showbox`, `\showthe`, `\showlists`, `\showgroups`, `\showtokens`, `\showifs`.
- make_box=20 for `\box`, `\copy`, `\lastbox`, `\vsplit`, `\vtop`, `\vbox`, and `\hbox` (106).
- hmove=21 for `\moveright` and `\moveleft`.
- vmove=22 for `\lower` and `\raise`.
- un_hbox=23 for `\unhbox` and `\unhcopy`.
- unvbox=24 for `\unvbox`, `\unvcopy`, `\pagediscards`, and `\splitdiscards`.
- remove_item=25 for `\unpenalty` (12), `\unkern` (11), `\unskip` (10).
- hskip=26 for `\hfil`, `\hfill`, `\hss`, `\hfilneg`, `\hskip`.
- vskip=27 for `\vfil`, `\vfill`, `\vss`, `\vfilneg`, `\vskip`.
- mskip=28 for `\mskip` (5).
- kern=29 for `\kern` (1).
- mkern=30 for `\mkern` (99).
- leader_ship=31 for `\shipout` (99), `\leaders` (100), `\cleaders` (101), `\xleaders` (102).

`\char` leaves vertical mode, then scans an integer operand, then calls `__unravel_char_in_mmode:n` or `__unravel_char:n` depending on the mode. See implementation of `the_char` and `other_char`.

```

3173 \__unravel_new_tex_cmd:nn { char_num } % 16
3174 {
3175   \__unravel_mode_non_vertical:n
3176   {
3177     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3178     \__unravel_print_action:
3179     \__unravel_scan_int:
3180     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3181     \mode_if_math:TF
3182     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
3183     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
3184   }
3185 }

```

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.

```

3186 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3187 {
3188   \__unravel_mode_math:n
3189   {
3190     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3191     \__unravel_print_action:
3192     \__unravel_scan_int:
3193     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3194     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
3195   }
3196 }

```

```

3197 \__unravel_new_tex_cmd:nn { mark } % 18
3198 {
3199   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3200   \__unravel_print_action:
3201   \int_compare:nNnF \l__unravel_head_char_int = 0
3202   { \__unravel_scan_int: }
3203   \__unravel_prev_input_gpush:
3204   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3205   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3206   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3207   \__unravel_print_action:x
3208   { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3209   \tl_put_right:Nx \l__unravel_head_tl
3210   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3211   \tl_use:N \l__unravel_head_tl
3212 }

```

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.

```

3213 \__unravel_new_tex_cmd:nn { xray } % 19
3214 {
3215   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3216   \__unravel_print_action:
3217   \int_case:nnF \l__unravel_head_char_int
3218   {
3219     { 0 }
3220     { % show
3221       \__unravel_get_next:
3222       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3223       \token_if_eq_meaning:NNTF
3224       \l__unravel_head_token \__unravel_special_relax:
3225       {
3226         \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
3227         \exp_after:wN \exp_not:N \l__unravel_head_tl
3228       }
3229     }
3230   }

```

```

3229         { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
3230     }
3231     { 2 }
3232     { % showthe
3233       \__unravel_get_x_next:
3234       \__unravel_scan_something_internal:n { 5 }
3235       \__unravel_prev_input_gpop:N \l__unravel_head_tl
3236       \__unravel_exp_args:Nx \use:n
3237       { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3238     }
3239   }
3240   { % no operand for showlists, showgroups, showifs
3241     \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
3242     { \__unravel_scan_int: }
3243     \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
3244     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3245     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3246     \tl_use:N \l__unravel_head_tl \scan_stop:
3247   }
3248 }
make_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
3249 \__unravel_new_tex_cmd:nn { make_box } % 20
3250 {
3251   \__unravel_prev_input_gpush:
3252   \__unravel_back_input:
3253   \__unravel_do_box:N \c_false_bool
3254 }
\__unravel_do_move: Scan a dimension and a box, and perform the shift, printing the appropriate action.
3255 \cs_new_protected:Npn \__unravel_do_move:
3256 {
3257   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3258   \__unravel_print_action:
3259   \__unravel_scan_normal_dimen:
3260   \__unravel_do_box:N \c_false_bool
3261 }
(End definition for \__unravel_do_move:.)
hmove=21 for \moveright and \moveleft.
3262 \__unravel_new_tex_cmd:nn { hmove } % 21
3263 {
3264   \mode_if_vertical:TF
3265   { \__unravel_do_move: } { \__unravel_forbidden_case: }
3266 }
vmove=22 for \lower and \raise.
3267 \__unravel_new_tex_cmd:nn { vmove } % 22
3268 {
3269   \mode_if_vertical:TF
3270   { \__unravel_forbidden_case: } { \__unravel_do_move: }
3271 }

```


`__unravel_do_unpackage:`

```
3272 \cs_new_protected:Npn \__unravel_do_unpackage:
3273 {
3274   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3275   \__unravel_print_action:
3276   \__unravel_scan_int:
3277   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3278   \tl_use:N \l__unravel_head_tl \scan_stop:
3279   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3280 }
```

(End definition for __unravel_do_unpackage:.)

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

```
3283 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3282 { \__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.

```
3283 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3284 {
3285   \__unravel_mode_vertical:n
3286   {
3287     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3288     { \l__unravel_head_token \__unravel_print_action: }
3289     { \__unravel_do_unpackage: }
3290   }
3291 }
```

`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.

```
3292 \__unravel_new_tex_cmd:nn { remove_item } % 25
3293 { \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.

```
3294 \cs_new_protected:Npn \__unravel_do_append_glue:
3295 {
3296   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3297   { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3298   {
3299     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3300     \__unravel_print_action:
3301     \exp_args:Nf \__unravel_scan_glue:n
3302     { \int_eval:n { \l__unravel_head_char_int - 2 } }
3303     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3304     \tl_use:N \l__unravel_head_tl \scan_stop:
3305     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3306   }
3307 }
```

(End definition for `_unravel_do_append_glue:`.)

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

3308 `_unravel_new_tex_cmd:nn { hskip }` % 26

3309 `{ _unravel_mode_non_vertical:n { _unravel_do_append_glue: } }`

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

3310 `_unravel_new_tex_cmd:nn { vskip }` % 27

3311 `{ _unravel_mode_vertical:n { _unravel_do_append_glue: } }`

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

3312 `_unravel_new_tex_cmd:nn { mskip }` % 28

3313 `{ _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.

3314 `\cs_new_protected:Npn _unravel_do_append_kern:`

3315 `{`

3316 `_unravel_prev_input_gpush:N \l__unravel_head_tl`

3317 `_unravel_print_action:`

3318 `\token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D`

3319 `{ _unravel_scan_dimen:nN { 2 } \c_false_bool }`

3320 `{ _unravel_scan_dimen:nN { 3 } \c_false_bool }`

3321 `_unravel_prev_input_gpop:N \l__unravel_head_tl`

3322 `\tl_use:N \l__unravel_head_tl \scan_stop:`

3323 `_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }`

3324 `}`

(End definition for `_unravel_do_append_kern:`.)

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

3325 `_unravel_new_tex_cmd:nn { kern }` % 29

3326 `{ _unravel_do_append_kern: }`

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

3327 `_unravel_new_tex_cmd:nn { mkern }` % 30

3328 `{ _unravel_mode_math:n { _unravel_do_append_kern: } }`

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

3329 `_unravel_new_tex_cmd:nn { leader_ship }` % 31

3330 `{`

3331 `_unravel_prev_input_gpush:N \l__unravel_head_tl`

3332 `_unravel_print_action:`

3333 `_unravel_do_box:N \c_true_bool`

3334 `}`

2.12.3 From 32 to 47

- `halign=32`
- `valign=33`
- `no_align=34`
- `vrule=35`
- `hrule=36`

- insert=37
- vadjust=38
- ignore_spaces=39
- after_assignment=40
- after_group=41
- break_penalty=42
- start_par=43
- ital_corr=44
- accent=45
- math_accent=46
- discretionary=47

```

3335 \_unravel_new_tex_cmd:nn { halign } % 32
3336 { \_unravel_not_implemented:n { halign } }
3337 \_unravel_new_tex_cmd:nn { valign } % 33
3338 { \_unravel_not_implemented:n { valign } }
3339 \_unravel_new_tex_cmd:nn { no_align } % 34
3340 { \_unravel_not_implemented:n { noalign } }

3341 \_unravel_new_tex_cmd:nn { vrule } % 35
3342 { \_unravel_mode_non_vertical:n { \_unravel_do_rule: } }
3343 \_unravel_new_tex_cmd:nn { hrule } % 36
3344 { \_unravel_mode_vertical:n { \_unravel_do_rule: } }
3345 \cs_new_protected:Npn \_unravel_do_rule:
3346 {
3347 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3348 \_unravel_print_action:
3349 \_unravel_scan_alt_rule:
3350 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3351 \tl_use:N \l__unravel_head_tl \scan_stop:
3352 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3353 }

3354 \_unravel_new_tex_cmd:nn { insert } % 37
3355 { \_unravel_begin_insert_or_adjust: }
3356 \_unravel_new_tex_cmd:nn { vadjust } % 38
3357 {
3358 \mode_if_vertical:TF
3359 { \_unravel_forbidden_case: } { \_unravel_begin_insert_or_adjust: }
3360 }

3361 \_unravel_new_tex_cmd:nn { ignore_spaces } % 39
3362 {
3363 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3364 {
3365 \_unravel_print_action:
3366 \_unravel_get_x_non_blank:
3367 \_unravel_set_cmd:

```

```

3368     \__unravel_do_step:
3369     }
3370     { \__unravel_not_implemented:n { pdfprimitive } }
3371 }
3372 \__unravel_new_tex_cmd:nn { after_assignment } % 40
3373 {
3374   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3375   \__unravel_get_next:
3376   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3377   \__unravel_print_action:x
3378   {
3379     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3380     \gtl_to_str:N \l__unravel_head_gtl
3381   }
3382 }

```

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.

```

3383 \__unravel_new_tex_cmd:nn { after_group } % 41
3384 {
3385   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3386   \__unravel_get_next:
3387   \int_compare:nNnTF \__unravel_currentgrouptype: = 0
3388   {
3389     \__unravel_print_action:x
3390     {
3391       Aftergroup~(level~0~=>~dropped):~
3392       \tl_to_str:N \l__unravel_tmpa_tl
3393       \gtl_to_str:N \l__unravel_head_gtl
3394     }
3395   }
3396   {
3397     \gtl_concat:NNN \l__unravel_after_group_gtl
3398     \l__unravel_after_group_gtl \l__unravel_head_gtl
3399     \__unravel_print_action:x
3400     {
3401       Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3402       \gtl_to_str:N \l__unravel_head_gtl
3403     }
3404   }
3405 }

```

See `__unravel_do_append_glue:`.

```

3406 \__unravel_new_tex_cmd:nn { break_penalty } % 42
3407 {
3408   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3409   \__unravel_print_action:
3410   \__unravel_scan_int:
3411   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3412   \tl_use:N \l__unravel_head_tl \scan_stop:
3413   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3414 }
3415 \__unravel_new_tex_cmd:nn { start_par } % 43
3416 {

```

```

3417 \mode_if_vertical:TF
3418 {
3419   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3420   { \__unravel_new_graf:N \c_false_bool }
3421   { \__unravel_new_graf:N \c_true_bool }
3422 }
3423 {
3424   \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3425   {
3426     \__unravel_hbox:w width \tex_parindent:D { }
3427     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3428   }
3429   \__unravel_print_action:
3430 }
3431 }
3432 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3433 {
3434   \mode_if_vertical:TF { \__unravel_forbidden_case: }
3435   { \l__unravel_head_token \__unravel_print_action: }
3436 }

```

__unravel_do_accent:

```

3437 \cs_new_protected:Npn \__unravel_do_accent:
3438 {
3439   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3440   \__unravel_print_action:
3441   \__unravel_scan_int:
3442   \__unravel_do_assignments:
3443   \bool_if:nTF
3444   {
3445     \token_if_eq_catcode_p:NN
3446     \l__unravel_head_token \c_catcode_letter_token
3447     ||
3448     \token_if_eq_catcode_p:NN
3449     \l__unravel_head_token \c_catcode_other_token
3450     ||
3451     \int_compare_p:nNn
3452     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3453   }
3454   { \__unravel_prev_input:V \l__unravel_head_tl }
3455   {
3456     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3457     {
3458       \__unravel_prev_input:V \l__unravel_head_tl
3459       \__unravel_scan_int:
3460     }
3461     { \__unravel_break:w }
3462   }
3463   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3464   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3465   \tl_use:N \l__unravel_head_tl \scan_stop:
3466   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3467   \__unravel_break_point:
3468 }

```

(End definition for `_unravel_do_accent:`)

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

```
3469 \cs_new_protected:Npn \_unravel_do_math_accent:
3470 {
3471   \_unravel_prev_input_gpush:N \l_unravel_head_tl
3472   \_unravel_print_action:
3473   \_unravel_scan_int:
3474   \_unravel_scan_math:
3475   \_unravel_prev_input_gpop:N \l_unravel_head_tl
3476   \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
3477   \tl_use:N \l_unravel_head_tl \scan_stop:
3478   \_unravel_print_action:x { \tl_to_str:N \l_unravel_head_tl }
3479 }
```

(End definition for `_unravel_do_math_accent:`)

```
3480 \_unravel_new_tex_cmd:nn { accent } % 45
3481 {
3482   \_unravel_mode_non_vertical:n
3483   {
3484     \mode_if_math:TF
3485     { \_unravel_do_math_accent: } { \_unravel_do_accent: }
3486   }
3487 }
3488 \_unravel_new_tex_cmd:nn { math_accent } % 46
3489 { \_unravel_mode_math:n { \_unravel_do_math_accent: } }
3490 \_unravel_new_tex_cmd:nn { discretionary } % 47
3491 { \_unravel_not_implemented:n { discretionary } }
```

2.12.4 Maths: from 48 to 56

- `eq_no=48`
- `left_right=49`
- `math_comp=50`
- `limit_switch=51`
- `above=52`
- `math_style=53`
- `math_choice=54`
- `non_script=55`
- `vcenter=56`

```
3492 \_unravel_new_tex_cmd:nn { eq_no } % 48
3493 { \_unravel_not_implemented:n { eqno } }
3494 \_unravel_new_tex_cmd:nn { left_right } % 49
3495 { \_unravel_not_implemented:n { left/right } }
```

```

3496 \_unravel_new_tex_cmd:nn { math_comp } % 50
3497 { \_unravel_not_implemented:n { math-comp } }
3498 \_unravel_new_tex_cmd:nn { limit_switch } % 51
3499 { \_unravel_not_implemented:n { limits } }
3500 \_unravel_new_tex_cmd:nn { above } % 52
3501 { \_unravel_not_implemented:n { above } }
3502 \_unravel_new_tex_cmd:nn { math_style } % 53
3503 { \_unravel_not_implemented:n { math~style } }
3504 \_unravel_new_tex_cmd:nn { math_choice } % 54
3505 { \_unravel_not_implemented:n { math-choice } }
3506 \_unravel_new_tex_cmd:nn { non_script } % 55
3507 { \_unravel_not_implemented:n { non-script } }
3508 \_unravel_new_tex_cmd:nn { vcenter } % 56
3509 { \_unravel_not_implemented:n { vcenter } }

```

2.12.5 From 57 to 70

- case_shift=57
- message=58
- extension=59
- in_stream=60
- begin_group=61
- end_group=62
- omit=63
- ex_space=64
- no_boundary=65
- radical=66
- end_cs_name=67
- char_given=68
- math_given=69
- last_item=70

```

3510 \_unravel_new_tex_cmd:nn { case_shift } % 57
3511 {
3512   \_unravel_prev_input_gpush:N \l_unravel_head_tl
3513   \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3514   \_unravel_prev_input_gpop:N \l_unravel_tmpa_tl
3515   \exp_after:wN \_unravel_case_shift:Nn \l_unravel_tmpa_tl
3516 }
3517 \cs_new_protected:Npn \_unravel_case_shift:Nn #1#2
3518 {

```

```

3519 #1 { \_unravel_back_input:n {#2} }
3520 \_unravel_print_action:x
3521 { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3522 }
3523 \_unravel_new_tex_cmd:nn { message } % 58
3524 {
3525 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3526 \_unravel_print_action:
3527 \_unravel_scan_toks_to_str:
3528 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3529 \tl_use:N \l__unravel_head_tl
3530 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3531 }

```

Extensions are implemented in a later section.

```

3532 \_unravel_new_tex_cmd:nn { extension } % 59
3533 {
3534 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3535 \_unravel_print_action:
3536 \_unravel_scan_extension_operands:
3537 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3538 \tl_use:N \l__unravel_head_tl \scan_stop:
3539 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3540 }
3541 \_unravel_new_tex_cmd:nn { in_stream } % 60
3542 {
3543 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3544 \_unravel_print_action:
3545 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3546 {
3547 \_unravel_scan_int:
3548 \_unravel_scan_optional_equals:
3549 \_unravel_scan_file_name:
3550 }
3551 { \_unravel_scan_int: }
3552 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3553 \tl_use:N \l__unravel_head_tl \scan_stop:
3554 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3555 }
3556 \_unravel_new_tex_cmd:nn { begin_group } % 61
3557 {
3558 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3559 \l__unravel_head_token
3560 \gtl_clear:N \l__unravel_after_group_gtl
3561 \_unravel_print_action:
3562 }
3563 \_unravel_new_tex_cmd:nn { end_group } % 62
3564 {
3565 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3566 \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
3567 \l__unravel_head_token
3568 \_unravel_print_action:
3569 }

```



```

3570 \__unravel_new_tex_cmd:nn { omit } % 63
3571 { \__unravel_not_implemented:n { omit } }

3572 \__unravel_new_tex_cmd:nn { ex_space } % 64
3573 {
3574   \__unravel_mode_non_vertical:n
3575   { \l__unravel_head_token \__unravel_print_action: }
3576 }

3577 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3578 {
3579   \__unravel_mode_non_vertical:n
3580   { \l__unravel_head_token \__unravel_print_action: }
3581 }

3582 \__unravel_new_tex_cmd:nn { radical } % 66
3583 {
3584   \__unravel_mode_math:n
3585   {
3586     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3587     \__unravel_print_action:
3588     \__unravel_scan_int:
3589     \__unravel_scan_math:
3590     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3591     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3592     \tl_use:N \l__unravel_head_tl \scan_stop:
3593     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3594   }
3595 }

3596 \__unravel_new_tex_cmd:nn { end_cs_name } % 67
3597 {
3598   \__unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
3599   \__unravel_print_action:
3600 }

See the_char and other_char.

3601 \__unravel_new_tex_cmd:nn { char_given } % 68
3602 {
3603   \__unravel_mode_non_vertical:n
3604   {
3605     \mode_if_math:TF
3606     { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
3607     { \__unravel_char:V \l__unravel_head_char_int }
3608   }
3609 }

See math_char_num.

3610 \__unravel_new_tex_cmd:nn { math_given } % 69
3611 {
3612   \__unravel_mode_math:n
3613   { \__unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3614 }

3615 \__unravel_new_tex_cmd:nn { last_item } % 70
3616 { \__unravel_forbidden_case: }

```

2.12.6 Extensions

_unravel_scan_extension_operands:

```
3617 \cs_new_protected:Npn \_unravel_scan_extension_operands:
3618 {
3619   \int_case:nnF \l__unravel_head_char_int
3620   {
3621     { 0 } % openout
3622     {
3623       \_unravel_scan_int:
3624       \_unravel_scan_optional_equals:
3625       \_unravel_scan_file_name:
3626     }
3627     { 1 } % write
3628     {
3629       \_unravel_scan_int:
3630       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3631     }
3632     { 2 } % closeout
3633     { \_unravel_scan_int: }
3634     { 3 } % special
3635     { \_unravel_scan_toks_to_str: }
3636     { 4 } % immediate
3637     { \_unravel_scan_immediate_operands: }
3638     { 5 } % setlanguage
3639     {
3640       \mode_if_horizontal:TF
3641       { \_unravel_scan_int: }
3642       { \_unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3643     }
3644     { 6 } % pdfliteral
3645     {
3646       \_unravel_scan_keyword:nF { dDiIrrReEcCtT }
3647       { \_unravel_scan_keyword:n { pPaAgGeE } }
3648       \_unravel_scan_pdf_ext_toks:
3649     }
3650     { 7 } % pdfobj
3651     {
3652       \_unravel_scan_keyword:nTF
3653       { rReEsSeErRvVeEoObBjJnNuUmM }
3654       { \_unravel_skip_optional_space: }
3655       {
3656         \_unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3657         { \_unravel_scan_int: }
3658         \_unravel_scan_keyword:nT { sStTrReEaAmM }
3659         {
3660           \_unravel_scan_keyword:nT { aAtTtTrR }
3661           { \_unravel_scan_pdf_ext_toks: }
3662         }
3663         \_unravel_scan_keyword:n { fFiIlLeE }
3664         \_unravel_scan_pdf_ext_toks:
3665       }
3666     }
3667     { 8 } % pdfrefobj
```

```

3668     { \_unravel_scan_int: }
3669 { 9 } % pdfxform
3670 {
3671     \_unravel_scan_keyword:nT { aAtTtTrR }
3672     { \_unravel_scan_pdf_ext_toks: }
3673     \_unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3674     { \_unravel_scan_pdf_ext_toks: }
3675     \_unravel_scan_int:
3676 }
3677 { 10 } % pdfrefxform
3678 { \_unravel_scan_int: }
3679 { 11 } % pdfximage
3680 { \_unravel_scan_image: }
3681 { 12 } % pdfrefximage
3682 { \_unravel_scan_int: }
3683 { 13 } % pdfannot
3684 {
3685     \_unravel_scan_keyword:nTF
3686     { rReEsSeErRvVeEoObBjJnNuUmM }
3687     { \_unravel_scan_optional_space: }
3688     {
3689         \_unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3690         { \_unravel_scan_int: }
3691         \_unravel_scan_alt_rule:
3692         \_unravel_scan_pdf_ext_toks:
3693     }
3694 }
3695 { 14 } % pdfstartlink
3696 {
3697     \mode_if_vertical:TF
3698     { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3699     {
3700         \_unravel_scan_rule_attr:
3701         \_unravel_scan_action:
3702     }
3703 }
3704 { 15 } % pdfendlink
3705 {
3706     \mode_if_vertical:T
3707     { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3708 }
3709 { 16 } % pdfoutline
3710 {
3711     \_unravel_scan_keyword:nT { aAtTtTrR }
3712     { \_unravel_scan_pdf_ext_toks: }
3713     \_unravel_scan_action:
3714     \_unravel_scan_keyword:nT { cCoOuUnNtT }
3715     { \_unravel_scan_int: }
3716     \_unravel_scan_pdf_ext_toks:
3717 }
3718 { 17 } % pdfdest
3719 { \_unravel_scan_pdfdest_operands: }
3720 { 18 } % pdfthread
3721 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }

```

```

3722 { 19 } % pdfstartthread
3723 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3724 { 20 } % pdfendthread
3725 { }
3726 { 21 } % pdfsavepos
3727 { }
3728 { 22 } % pdfinfo
3729 { \_unravel_scan_pdf_ext_toks: }
3730 { 23 } % pdfcatalog
3731 {
3732     \_unravel_scan_pdf_ext_toks:
3733     \_unravel_scan_keyword:n { o0pPeEnNaAcCtTiIoOnN }
3734     { \_unravel_scan_action: }
3735 }
3736 { 24 } % pdfnames
3737 { \_unravel_scan_pdf_ext_toks: }
3738 { 25 } % pdffontattr
3739 {
3740     \_unravel_scan_font_ident:
3741     \_unravel_scan_pdf_ext_toks:
3742 }
3743 { 26 } % pdfincludechars
3744 {
3745     \_unravel_scan_font_ident:
3746     \_unravel_scan_pdf_ext_toks:
3747 }
3748 { 27 } % pdfmapfile
3749 { \_unravel_scan_pdf_ext_toks: }
3750 { 28 } % pdfmapline
3751 { \_unravel_scan_pdf_ext_toks: }
3752 { 29 } % pdftrailer
3753 { \_unravel_scan_pdf_ext_toks: }
3754 { 30 } % pdfresettimer
3755 { }
3756 { 31 } % pdffontexpand
3757 {
3758     \_unravel_scan_font_ident:
3759     \_unravel_scan_optional_equals:
3760     \_unravel_scan_int:
3761     \_unravel_scan_int:
3762     \_unravel_scan_int:
3763     \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3764     { \_unravel_skip_optional_space: }
3765 }
3766 { 32 } % pdfsetrandomseed
3767 { \_unravel_scan_int: }
3768 { 33 } % pdfsnaprefpoint
3769 { }
3770 { 34 } % pdfsnapy
3771 { \_unravel_scan_normal_glue: }
3772 { 35 } % pdfsnapycomp
3773 { \_unravel_scan_int: }
3774 { 36 } % pdfglyphtounicode
3775 {

```

```

3776         \_unravel_scan_pdf_ext_toks:
3777         \_unravel_scan_pdf_ext_toks:
3778     }
3779     { 37 } % pdfcolorstack
3780     { \_unravel_scan_pdfcolorstack_operands: }
3781     { 38 } % pdfsetmatrix
3782     { \_unravel_scan_pdf_ext_toks: }
3783     { 39 } % pdfsave
3784     { }
3785     { 40 } % pdfrestore
3786     { }
3787     { 41 } % pdfnoblendintounicode
3788     { \_unravel_scan_font_ident: }
3789 }
3790 { } % no other cases.
3791 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```

3792 \cs_new_protected:Npn \_unravel_scan_pdfcolorstack_operands:
3793 {
3794     \_unravel_scan_int:
3795     \_unravel_scan_keyword:nF { sSeEtT }
3796     {
3797         \_unravel_scan_keyword:nF { pPuUsShH }
3798         {
3799             \_unravel_scan_keyword:nF { pPoOpP }
3800             {
3801                 \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
3802                 {
3803                     \_unravel_error:nnnnn { color-stack-action-missing }
3804                     { } { } { } { }
3805                 }
3806             }
3807         }
3808     }
3809 }

```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```

3810 \cs_new_protected:Npn \_unravel_scan_rule_attr:
3811 {
3812     \_unravel_scan_alt_rule:
3813     \_unravel_scan_keyword:nT { aAtTtTrR }
3814     { \_unravel_scan_pdf_ext_toks: }
3815 }

```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```

3816 \cs_new_protected:Npn \_unravel_scan_action:
3817 {
3818     \_unravel_scan_keyword:nTF { uUsSeErR }

```

```

3819     { \_unravel_scan_pdf_ext_toks: }
3820     {
3821     \_unravel_scan_keyword:nF { gGoOtToO }
3822     {
3823     \_unravel_scan_keyword:nF { tThHrReEaAdD }
3824     { \_unravel_error:nnnn { action-type-missing } { } { } { } { } }
3825     }
3826     }
3827 \_unravel_scan_keyword:nT { fFiIlLeE }
3828 { \_unravel_scan_pdf_ext_toks: }
3829 \_unravel_scan_keyword:nTF { pPaAgGeE }
3830 {
3831 \_unravel_scan_int:
3832 \_unravel_scan_pdf_ext_toks:
3833 }
3834 {
3835 \_unravel_scan_keyword:nTF { nNaAmMeE }
3836 { \_unravel_scan_pdf_ext_toks: }
3837 {
3838 \_unravel_scan_keyword:nTF { nNuUmM }
3839 { \_unravel_scan_int: }
3840 { \_unravel_error:nnnn { identifier-type-missing } { } { } { } { } }
3841 }
3842 }
3843 \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3844 { \_unravel_skip_optional_space: }
3845 {
3846 \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3847 { \_unravel_skip_optional_space: }
3848 }
3849 }

```

(End definition for _unravel_scan_action:.)

_unravel_scan_image: Used by \pdfximage.

```

3850 \cs_new_protected:Npn \_unravel_scan_image:
3851 {
3852 \_unravel_scan_rule_attr:
3853 \_unravel_scan_keyword:nTF { nNaAmMeEdD }
3854 { \_unravel_scan_pdf_ext_toks: }
3855 {
3856 \_unravel_scan_keyword:nT { pPaAgGeE }
3857 { \_unravel_scan_int: }
3858 }
3859 \_unravel_scan_keyword:nT { cCoO1LoOrRsSpPaAcCeE }
3860 { \_unravel_scan_int: }
3861 \_unravel_scan_pdf_ext_toks:
3862 }

```

(End definition for _unravel_scan_image:.)

_unravel_scan_immediate_operands:

```

3863 \cs_new_protected:Npn \_unravel_scan_immediate_operands:
3864 {
3865 \_unravel_get_x_next:

```

```

3866 \__unravel_set_cmd:
3867 \int_compare:nNnTF
3868 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
3869 {
3870   \int_compare:nNnTF
3871     \l__unravel_head_char_int < { 3 } % openout, write, closeout
3872     { \__unravel_scan_immediate_operands_aux: }
3873     {
3874       \int_case:nnF \l__unravel_head_char_int
3875       {
3876         { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
3877         { 9 } { \__unravel_scan_extension_operands_aux: } % pdfxform
3878         { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
3879       }
3880       { \__unravel_scan_immediate_operands_bad: }
3881     }
3882   }
3883   { \__unravel_scan_immediate_operands_bad: }
3884 }
3885 \cs_new_protected:Npn \__unravel_scan_immediate_operands_aux:
3886 {
3887   \__unravel_prev_input:V \l__unravel_head_tl
3888   \__unravel_scan_extension_operands:
3889 }
3890 \cs_new_protected:Npn \__unravel_scan_immediate_operands_bad:
3891 {
3892   \__unravel_back_input:
3893   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3894   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3895   \__unravel_prev_input_gpush:
3896 }
3897

```

(End definition for __unravel_scan_immediate_operands:.)

__unravel_scan_pdfdest_operands:

```

3898 \cs_new_protected:Npn \__unravel_scan_pdfdest_operands:
3899 {
3900   \__unravel_scan_keyword:nTF { nNuUmM }
3901     { \__unravel_scan_int: }
3902     {
3903       \__unravel_scan_keyword:nTF { nNaAmMeE }
3904         { \__unravel_scan_pdf_ext_toks: }
3905         { \__unravel_error:nmnn { identifier-type-missing } { } { } { } { } }
3906     }
3907   \__unravel_scan_keyword:nTF { xXyYzZ }
3908     {
3909       \__unravel_scan_keyword:nT { zZoOoOmM }
3910       { \__unravel_scan_int: }
3911     }
3912     {
3913       \__unravel_scan_keyword:nF { fFiItTbBhH }
3914       {
3915         \__unravel_scan_keyword:nF { fFiItTbBvV }

```

```

3916     {
3917         \_unravel_scan_keyword:nF { fFiItTbB }
3918         {
3919             \_unravel_scan_keyword:nF { fFiItThHhH }
3920             {
3921                 \_unravel_scan_keyword:nF { fFiItTvV }
3922                 {
3923                     \_unravel_scan_keyword:nTF
3924                     { fFiItTrR }
3925                     {
3926                         \_unravel_skip_optional_space:
3927                         \_unravel_scan_alt_rule:
3928                         \use_none:n
3929                     }
3930                     {
3931                         \_unravel_scan_keyword:nF
3932                         { fFiItT }
3933                         {
3934                             \_unravel_error:nnnn { destination-type-missing }
3935                             { } { } { } { }
3936                         }
3937                     }
3938                 }
3939             }
3940         }
3941     }
3942 }
3943 }
3944 \_unravel_skip_optional_space:
3945 }

```

(End definition for _unravel_scan_pdfdest_operands:.)

2.12.7 Assignments

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

```

3946 \cs_set_protected:Npn \_unravel_tmp:w
3947 {
3948     \_unravel_prev_input_gpush:
3949     \_unravel_prefixed_command:
3950 }
3951 \int_step_inline:nnn
3952 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
3953 { 1 }
3954 { \_unravel_tex_use:n { max_command } }
3955 { \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.

```

3956 \cs_new_protected:Npn \_unravel_prefixed_command:
3957 {

```



```

3958 \int_while_do:nNnn
3959 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
3960 {
3961   \__unravel_prev_input:V \l__unravel_head_tl
3962   \__unravel_get_x_non_relax:
3963   \__unravel_set_cmd:
3964   \int_compare:nNnF \l__unravel_head_cmd_int
3965   > { \__unravel_tex_use:n { max_non_prefixed_command } }
3966   {
3967     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3968     \__unravel_error:nxxxx { erroneous-prefixes }
3969     { \tl_to_str:N \l__unravel_tmpa_tl }
3970     { \tl_to_str:N \l__unravel_head_tl }
3971     { } { }
3972     \__unravel_back_input:
3973     \__unravel_omit_after_assignment:w
3974   }
3975 }
3976 % ^^A todo: Discard non-\global prefixes if they are irrelevant
3977 % ^^A todo: Adjust for the setting of \globaldefs
3978 \cs_if_exist_use:cF
3979 { \__unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3980 {
3981   \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
3982   \__unravel_omit_after_assignment:w
3983 }
3984 \__unravel_after_assignment:
3985 }

```

(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
3986 \cs_new_protected:Npn \__unravel_after_assignment:
3987 {
3988   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3989   \gtl_gclear:N \g__unravel_after_assignment_gtl
3990 }
3991 \cs_new_protected:Npn \__unravel_omit_after_assignment:w
3992 #1 \__unravel_after_assignment: { }

```

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

```
\__unravel_prefixed_new:nn
```

```

3993 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3994 {
3995   \cs_new_protected:cpn
3996   { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3997 }

```

(End definition for `__unravel_prefixed_new:nn`.)

_unravel_assign_token:n

```
3998 \cs_new_protected:Npn \_unravel_assign_token:n #1
3999 {
4000   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4001   #1
4002   \tl_use:N \l__unravel_head_tl \scan_stop:
4003   \_unravel_print_assigned_token:
4004 }
```

(End definition for _unravel_assign_token:n.)

_unravel_assign_register:

```
4005 \cs_new_protected:Npn \_unravel_assign_register:
4006 {
4007   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4008   \tl_use:N \l__unravel_head_tl \scan_stop:
4009   \_unravel_print_assigned_register:
4010 }
```

(End definition for _unravel_assign_register:.)

_unravel_assign_value:nn

```
4011 \cs_new_protected:Npn \_unravel_assign_value:nn #1#2
4012 {
4013   \tl_if_empty:nF {#1}
4014   {
4015     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4016     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4017     #1
4018     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4019   }
4020   \_unravel_prev_input:V \l__unravel_head_tl
4021   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4022   \_unravel_scan_optional_equals:
4023   #2
4024   \_unravel_assign_register:
4025 }
```

(End definition for _unravel_assign_value:nn.)

_unravel_assign_toks:

```
4026 \_unravel_prefixed_new:nn { toks_register } % 71
4027 {
4028   \int_compare:nNnT \l__unravel_head_char_int = 0
4029   { % \toks
4030     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4031     \_unravel_print_action:
4032     \_unravel_scan_int:
4033     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4034   }
4035   \_unravel_assign_toks:
4036 }
4037 \_unravel_prefixed_new:nn { assign_toks } % 72
4038 { \_unravel_assign_toks: }
```

```

4039 \cs_new_protected:Npn \__unravel_assign_toks:
4040 {
4041   \__unravel_prev_input_silent:V \l__unravel_head_tl
4042   \__unravel_print_action:
4043   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4044   \__unravel_scan_optional_equals:
4045   \__unravel_get_x_non_relax:
4046   \__unravel_set_cmd:
4047   \int_compare:nNnTF
4048     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
4049     {
4050       \__unravel_prev_input:V \l__unravel_head_tl
4051       \int_compare:nNnT \l__unravel_head_char_int = 0
4052       { \__unravel_scan_int: }
4053     }
4054     {
4055       \int_compare:nNnTF
4056         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
4057         { \__unravel_prev_input:V \l__unravel_head_tl }
4058         {
4059           \__unravel_back_input:
4060           \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4061         }
4062     }
4063   \__unravel_assign_register:
4064 }

```

(End definition for __unravel_assign_toks.)

```

4065 \__unravel_prefixed_new:nn { assign_int } % 73
4066 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4067 \__unravel_prefixed_new:nn { assign_dimen } % 74
4068 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4069 \__unravel_prefixed_new:nn { assign_glue } % 75
4070 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
4071 \__unravel_prefixed_new:nn { assign_mu_glue } % 76
4072 { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
4073 \__unravel_prefixed_new:nn { assign_font_dimen } % 77
4074 {
4075   \__unravel_assign_value:nn
4076   { \__unravel_scan_int: \__unravel_scan_font_ident: }
4077   { \__unravel_scan_normal_dimen: }
4078 }
4079 \__unravel_prefixed_new:nn { assign_font_int } % 78
4080 {
4081   \__unravel_assign_value:nn
4082   { \__unravel_scan_font_int: } { \__unravel_scan_int: }
4083 }
4084 \__unravel_prefixed_new:nn { set_aux } % 79
4085 { % prevdepth = 1, spacefactor = 102
4086   \int_compare:nNnTF \l__unravel_head_char_int = 1
4087     { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
4088     { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
4089 }
4090 \__unravel_prefixed_new:nn { set_prev_graf } % 80

```

```

4091 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4092 \_unravel_prefixed_new:nn { set_page_dimen } % 81
4093 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4094 \_unravel_prefixed_new:nn { set_page_int } % 82
4095 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4096 \_unravel_prefixed_new:nn { set_box_dimen } % 83
4097 {
4098   \_unravel_assign_value:nn
4099   { \_unravel_scan_int: } { \_unravel_scan_normal_dimen: }
4100 }
4101 \_unravel_prefixed_new:nn { set_shape } % 84
4102 {
4103   \_unravel_assign_value:nn { \_unravel_scan_int: }
4104   {
4105     \prg_replicate:nn
4106     {
4107       \tl_if_head_eq_meaning:VNT
4108       \l_unravel_defined_tl \tex_parshape:D { 2 * }
4109       \tl_tail:N \l_unravel_defined_tl
4110     }
4111     { \_unravel_scan_int: }
4112   }
4113 }
4114 \_unravel_prefixed_new:nn { def_code } % 85
4115 {
4116   \_unravel_assign_value:nn
4117   { \_unravel_scan_int: } { \_unravel_scan_int: }
4118 }
4119 \_unravel_prefixed_new:nn { def_family } % 86
4120 {
4121   \_unravel_assign_value:nn
4122   { \_unravel_scan_int: } { \_unravel_scan_font_ident: }
4123 }
4124 \_unravel_prefixed_new:nn { set_font } % 87
4125 {
4126   \_unravel_prev_input_gpop:N \l_unravel_tmpa_tl
4127   \tl_put_left:NV \l_unravel_head_tl \l_unravel_tmpa_tl
4128   \tl_use:N \l_unravel_head_tl \scan_stop:
4129   \gtl_gput_right:NV \g_unravel_output_gtl \l_unravel_head_tl
4130   \_unravel_print_action:
4131 }
4132 \_unravel_prefixed_new:nn { def_font } % 88
4133 {
4134   \_unravel_prev_input_silent:V \l_unravel_head_tl
4135   \_unravel_set_action_text:x { \tl_to_str:N \l_unravel_head_tl }
4136   \_unravel_scan_r_token:
4137   \_unravel_print_action:x
4138   { \g_unravel_action_text_str \tl_to_str:N \l_unravel_defined_tl }
4139   \_unravel_scan_optional_equals:
4140   \_unravel_scan_file_name:
4141   \bool_gset_true:N \g_unravel_name_in_progress_bool
4142   \_unravel_scan_keyword:nTF { aAT }
4143   { \_unravel_scan_normal_dimen: }

```

```

4144     {
4145       \__unravel_scan_keyword:nT { sScCaAlLeEdD }
4146       { \__unravel_scan_int: }
4147     }
4148     \bool_gset_false:N \g__unravel_name_in_progress_bool
4149     \__unravel_assign_token:n { }
4150   }

```

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

let, futurelet

```

4151 \__unravel_prefixed_new:nn { let } % 94
4152 {
4153   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4154   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4155   { % |let|
4156     \__unravel_scan_r_token:
4157     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4158     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4159     \__unravel_get_next:
4160     \bool_while_do:nn
4161     { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4162     { \__unravel_get_next: }
4163     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4164     { \__unravel_get_next: }
4165     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4166     { \__unravel_get_next: }
4167   }
4168   { % |futurelet|
4169     \__unravel_scan_r_token:
4170     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4171     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4172     \__unravel_get_next:
4173     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4174     \__unravel_get_next:
4175     \__unravel_back_input:
4176     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4177     \__unravel_back_input:
4178   }
4179   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4180   \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4181   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4182   \__unravel_exp_args:Nx \use:n
4183   {
4184     \exp_not:V \l__unravel_head_tl
4185     \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4186   }
4187   \__unravel_print_assigned_token:
4188 }
4189 \__unravel_prefixed_new:nn { shorthand_def } % 95
4190 {
4191   \__unravel_prev_input_silent:V \l__unravel_head_tl
4192   \tl_set:Nx \l__unravel_prev_action_tl
4193   { \tl_to_str:N \l__unravel_head_tl }

```

```

4194   \__unravel_scan_r_token:
4195   \__unravel_print_action:x
4196     { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4197   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4198   \__unravel_scan_optional_equals:
4199   \__unravel_scan_int:
4200   \__unravel_assign_token:n { }
4201 }

```

`__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.

```

4202 \__unravel_prefixed_new:nn { read_to_cs } % 96
4203 {
4204   \__unravel_prev_input_silent:V \l__unravel_head_tl
4205   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4206   \__unravel_scan_int:
4207   \__unravel_scan_to:
4208   \__unravel_scan_r_token:
4209   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4210   \__unravel_read_to_cs_safe:fTF
4211     { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
4212     { \__unravel_assign_token:n { } }
4213     {
4214       \__unravel_prev_input_gpop:N \l__unravel_head_tl
4215       \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4216     }
4217 }
4218 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
4219 {
4220   \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
4221     { \prg_return_true: }
4222     {
4223       \int_compare:nNnTF {#1} < { 0 }
4224         { \prg_return_false: }
4225         {
4226           \int_compare:nNnTF {#1} > { 15 }
4227             { \prg_return_false: }
4228             {
4229               \tex_ifeof:D #1 \exp_stop_f:
4230                 \prg_return_false:
4231               \else:
4232                 \prg_return_true:
4233             \fi:
4234           }
4235         }
4236     }
4237 }
4238 \cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }

```

(End definition for `__unravel_read_to_cs_safe:nTF`.)

```

4239 \__unravel_prefixed_new:nn { def } % 97
4240 {
4241   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4242   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4243   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4244   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4245   \int_compare:nNnTF \l__unravel_head_char_int < 2
4246     { % def/gdef
4247       \__unravel_scan_r_token:
4248       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4249       \__unravel_scan_toks:NN \c_true_bool \c_false_bool
4250     }
4251     { % edef/xdef
4252       \__unravel_scan_r_token:
4253       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4254       \__unravel_scan_toks:NN \c_true_bool \c_true_bool
4255     }
4256   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4257   \__unravel_prev_input:V \l__unravel_head_tl
4258   \__unravel_assign_token:n
4259   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4260 }

```

\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.

```

4261 \__unravel_prefixed_new:nn { set_box } % 98
4262 {
4263   \__unravel_prev_input:V \l__unravel_head_tl
4264   \__unravel_scan_int:
4265   \__unravel_scan_optional_equals:
4266   \bool_if:NTF \g__unravel_set_box_allowed_bool
4267     { \__unravel_do_box:N \c_false_bool }
4268     {
4269       \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
4270       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4271       \__unravel_omit_after_assignment:w
4272     }
4273 }

```

\hyphenation and \patterns

```

4274 \__unravel_prefixed_new:nn { hyph_data } % 99
4275 {
4276   \__unravel_prev_input:V \l__unravel_head_tl
4277   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4278   \__unravel_assign_token:n { }
4279 }

```

```

4280 \__unravel_prefixed_new:nn { set_interaction } % 100
4281 {
4282   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4283   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4284   \tl_use:N \l__unravel_head_tl \scan_stop:
4285   \__unravel_print_assignment:x { \tl_to_str:N \l__unravel_head_tl }
4286 }

```

```

4287 \__unravel_prefixed_new:nn { letterspace_font } % 101
4288 {
4289   \__unravel_prev_input_silent:V \l__unravel_head_tl
4290   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4291   \__unravel_scan_r_token:
4292   \__unravel_print_action:x
4293   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4294   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4295   \__unravel_scan_optional_equals:
4296   \__unravel_scan_font_ident:
4297   \__unravel_scan_int:
4298   \__unravel_assign_token:n { }
4299 }
4300 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4301 {
4302   \__unravel_prev_input_silent:V \l__unravel_head_tl
4303   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4304   \__unravel_scan_r_token:
4305   \__unravel_print_action:x
4306   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4307   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4308   \__unravel_scan_optional_equals:
4309   \__unravel_scan_font_ident:
4310   \__unravel_assign_token:n { }
4311 }

```

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

```

4312 \__unravel_prefixed_new:nn { register } % 89
4313 { \__unravel_do_register:N 0 }
4314 \__unravel_prefixed_new:nn { advance } % 90
4315 { \__unravel_do_operation:N 1 }
4316 \__unravel_prefixed_new:nn { multiply } % 91
4317 { \__unravel_do_operation:N 2 }
4318 \__unravel_prefixed_new:nn { divide } % 92
4319 { \__unravel_do_operation:N 3 }

```

`__unravel_do_operation:N`

`__unravel_do_operation_fail:w`

```

4320 \cs_new_protected:Npn \__unravel_do_operation:N #1
4321 {
4322   \__unravel_prev_input_silent:V \l__unravel_head_tl
4323   \__unravel_print_action:
4324   \__unravel_get_x_next:
4325   \__unravel_set_cmd:
4326   \int_compare:nNnTF
4327     \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4328     {
4329       \int_compare:nNnTF
4330         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4331         { \__unravel_do_register:N #1 }
4332         { \__unravel_do_operation_fail:w }
4333     }
4334     {
4335       \int_compare:nNnTF

```



```

4336         \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
4337     { \__unravel_do_operation_fail:w }
4338     {
4339         \__unravel_prev_input:V \l__unravel_head_tl
4340         \exp_args:NNf \__unravel_do_register_set:Nn #1
4341         {
4342             \int_eval:n
4343             {
4344                 \l__unravel_head_cmd_int
4345                 - \__unravel_tex_use:n { assign_toks }
4346             }
4347         }
4348     }
4349 }
4350 }
4351 \cs_new_protected:Npn \__unravel_do_operation_fail:w
4352 {
4353     \__unravel_error:nnnnn { after-advance } { } { } { } { }
4354     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4355     \__unravel_omit_after_assignment:w
4356 }

```

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

__unravel_do_register:N
 __unravel_do_register_aux:Nn

```

4357 \cs_new_protected:Npn \__unravel_do_register:N #1
4358 {
4359     \exp_args:NNV \__unravel_do_register_aux:Nn #1
4360     \l__unravel_head_char_int
4361 }
4362 \cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
4363 {
4364     \int_compare:nNnTF { \tl_tail:n {#2} } = 0
4365     {
4366         \__unravel_prev_input_gpush:N \l__unravel_head_tl
4367         \__unravel_print_assignment:
4368         \__unravel_scan_int:
4369         \__unravel_prev_input_gpop:N \l__unravel_head_tl
4370         \__unravel_prev_input_silent:V \l__unravel_head_tl
4371     }
4372     {
4373         \__unravel_prev_input_silent:V \l__unravel_head_tl
4374         \__unravel_print_assignment:
4375     }
4376     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4377     \exp_args:NNf \__unravel_do_register_set:Nn #1
4378     { \int_eval:n { #2 / 1 000 000 } }
4379 }

```

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

__unravel_do_register_set:Nn

```

4380 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
4381 {
4382     \int_compare:nNnTF {#1} = 0

```

```

4383     { % truly register command
4384       \__unravel_scan_optional_equals:
4385     }
4386     { % \advance, \multiply, \divide
4387       \__unravel_scan_keyword:nF { bByY }
4388       { \__unravel_prev_input_silent:n { by } }
4389     }
4390   \int_compare:nNnTF {#1} < 2
4391   {
4392     \int_case:nnF {#2}
4393     {
4394       { 1 } { \__unravel_scan_int:          } % count
4395       { 2 } { \__unravel_scan_normal_dimen: } % dim
4396       { 3 } { \__unravel_scan_normal_glue: } % glue
4397       { 4 } { \__unravel_scan_mu_glue:     } % muglue
4398     }
4399     { \__unravel_error:nxxxx { internal } { do-reg=#2 } { } { } { } }
4400   }
4401   { \__unravel_scan_int: }
4402   \__unravel_assign_register:
4403 }

```

(End definition for `__unravel_do_register_set:Nn`.)

The following is used for instance when making accents.

```

4404 \cs_new_protected:Npn \__unravel_do_assignments:
4405 {
4406   \__unravel_get_x_non_relax:
4407   \__unravel_set_cmd:
4408   \int_compare:nNnT
4409     \l__unravel_head_cmd_int
4410     > { \__unravel_tex_use:n { max_non_prefixed_command } }
4411   {
4412     \bool_gset_false:N \g__unravel_set_box_allowed_bool
4413     \__unravel_prev_input_gpush:
4414     \__unravel_prefixed_command:
4415     \bool_gset_true:N \g__unravel_set_box_allowed_bool
4416     \__unravel_do_assignments:
4417   }
4418 }

```

2.13 Expandable primitives

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

- `undefined_cs=103` for undefined control sequences (not quite a primitive).
- `expand_after=104` for `\expandafter` and `\unless`.
- `no_expand=105` for `\noexpand` and `\pdfprimitive`.
- `input=106` for `\input`, `\endinput` and `\scantokens`.
- `if_test=107` for the conditionals, `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\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`, `\pdfinsertht`, `\pdfximagebbox`, `\jobname`, `\expanded`, and in Lua \TeX `\directlua`, `\luaescapestring`, and in X \TeX `\Ucharcat`.
- `the=111` for `\the`, `\unexpanded`, and `\detokenize`.
- `top_bot_mark=112` `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`, `\topmarks`, `\firstmarks`, `\botmarks`, `\splitfirstmarks`, and `\splitbotmarks`.
- `call=113` for macro calls, implemented by `__unravel_macro_call:`.
- `end_template=117` for \TeX 's end template.

Let \TeX trigger an error.

```

4419 \__unravel_new_tex_expandable:nm { undefined_cs } % 103
4420 { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }

\__unravel_expandafter:
  \__unravel_unless:
  \__unravel_unless_bad:
4421 \__unravel_new_tex_expandable:nm { expand_after } % 104
4422 {
4423   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4424   { \__unravel_expandafter: } { \__unravel_unless: }
4425 }
4426 \cs_new_protected:Npn \__unravel_expandafter:
4427 {
4428   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4429   \__unravel_get_next:
4430   \gtl_concat:NNN \l__unravel_head_gtl
4431   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4432   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4433   \__unravel_print_expansion:x { \gtl_to_str:N \l__unravel_head_gtl }
4434   \__unravel_get_next:
4435   \__unravel_token_if_expandable:NTF \l__unravel_head_token
4436   { \__unravel_expand_do:N \prg_do_nothing: }
4437   { \__unravel_back_input: }
4438   \__unravel_prev_input_gpop:N \l__unravel_head_gtl
4439   \__unravel_set_action_text:x
4440   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4441   \gtl_pop_left:N \l__unravel_head_gtl
4442   \__unravel_back_input:
4443   \__unravel_print_expansion:
4444 }
4445 \cs_new_protected:Npn \__unravel_unless:
4446 {
4447   \__unravel_get_token:

```

```

4448 \int_compare:nNnTF
4449 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4450 {
4451   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4452   { \__unravel_unless_bad: }
4453   {
4454     \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4455     % \int_add:Nn \l__unravel_head_char_int { 32 }
4456     \__unravel_expand_nonmacro:
4457   }
4458 }
4459 { \__unravel_unless_bad: }
4460 }
4461 \cs_new_protected:Npn \__unravel_unless_bad:
4462 {
4463   \__unravel_error:nnnnn { bad-unless } { } { } { } { }
4464   \__unravel_back_input:
4465 }

```

(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.

```

4466 \__unravel_new_tex_expandable:nm { no_expand } % 105
4467 {
4468   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4469   { \__unravel_noexpand:N }
4470   { \__unravel_pdfprimitive: }
4471 }
4472 \cs_new_protected:Npn \__unravel_noexpand:N #1
4473 {
4474   \__unravel_get_token:
4475   \cs_if_eq:NNTF #1 \prg_do_nothing:
4476   {
4477     \tl_if_empty:NTF \l__unravel_head_tl
4478     { \__unravel_back_input: }
4479     {
4480       \exp_after:wN \__unravel_token_if_definable:NNTF \l__unravel_head_tl
4481       { \__unravel_noexpand_after: }

```

```

4482         { \_unravel_back_input: }
4483     }
4484 }
4485 {
4486     \_unravel_back_input:
4487     \_unravel_get_next:
4488     \_unravel_token_if_expandable:NT \l_unravel_head_token
4489     { \cs_set_eq:NN \l_unravel_head_token \_unravel_special_relax: }
4490 }
4491 }
4492 \cs_new_protected:Npn \_unravel_noexpand_after:
4493 {
4494     \group_begin:
4495     \_unravel_set_escapechar:n { 92 }
4496     \exp_args:NNc
4497     \group_end:
4498     \_unravel_noexpand_after:N
4499     { notexpanded: \exp_after:wN \token_to_str:N \l_unravel_head_tl }
4500 }
4501 \cs_new_protected:Npn \_unravel_noexpand_after:N #1
4502 {
4503     \cs_gset_eq:NN #1 \_unravel_special_relax:
4504     \_unravel_back_input:n {#1}
4505 }
4506 \cs_new_protected:Npn \_unravel_pdfprimitive:
4507 { \_unravel_not_implemented:n { pdfprimitive } }

(End definition for \_unravel_noexpand:N, \_unravel_noexpand_after:, and \_unravel_pdfprimitive:.)

\_unravel_endinput:
\_unravel_scantokens: 4508 \_unravel_new_tex_expandable:nn { input } % 106
\_unravel_input: 4509 {
4510     \int_case:nnF \l_unravel_head_char_int
4511     {
4512         { 1 } { \_unravel_endinput: } % \endinput
4513         { 2 } { \_unravel_scantokens: } % \scantokens
4514     }
4515     { % 0=\input
4516         \bool_if:NTF \g_unravel_name_in_progress_bool
4517         { \_unravel_insert_relax: } { \_unravel_input: }
4518     }
4519 }
4520 \cs_new_protected:Npn \_unravel_endinput:
4521 {
4522     \group_begin:
4523     \msg_warning:nn { unravel } { endinput-ignored }
4524     \group_end:
4525     \_unravel_print_expansion:
4526 }
4527 \cs_new_protected:Npn \_unravel_scantokens:
4528 {
4529     \_unravel_prev_input_gpush:
4530     \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4531     \_unravel_prev_input_gpop:N \l_unravel_tmpa_tl

```

```

4532     \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4533     \__unravel_back_input:V \l__unravel_head_tl
4534     \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_tmpa_tl }
4535   }
4536 \cs_new_protected:Npn \__unravel_input:
4537 {
4538   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4539   \__unravel_scan_file_name:
4540   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4541   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4542   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4543   \__unravel_back_input:V \l__unravel_tmpa_tl
4544   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
4545 }

```

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

__unravel_csname_loop:

```

4546 \__unravel_new_tex_expandable:nn { cs_name } % 109
4547 {
4548   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4549   \__unravel_print_expansion:
4550   \__unravel_csname_loop:
4551   \__unravel_prev_input_silent:V \l__unravel_head_tl
4552   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4553   \__unravel_back_input_tl_o:
4554 }
4555 \cs_new_protected:Npn \__unravel_csname_loop:
4556 {
4557   \__unravel_get_x_next:
4558   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4559   {
4560     \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4561     {
4562       \__unravel_back_input:
4563       \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4564       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4565     }
4566   }
4567   {
4568     \__unravel_prev_input_silent:x
4569     { \__unravel_token_to_char:N \l__unravel_head_token }
4570     \__unravel_csname_loop:
4571   }
4572 }

```

(End definition for __unravel_csname_loop:.)

```

4573 \__unravel_new_tex_expandable:nn { convert } % 110
4574 {
4575   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4576   \__unravel_print_expansion:
4577   \int_case:nn \l__unravel_head_char_int
4578   {
4579     0      \__unravel_scan_int:

```

```

4580     1     \__unravel_scan_int:
4581     2     \__unravel_convert_string:
4582     3     \__unravel_convert_meaning:
4583     4     \__unravel_scan_font_ident:
4584     8     \__unravel_scan_font_ident:
4585     9     \__unravel_scan_font_ident:
4586     { 10 } \__unravel_scan_font_ident:
4587     { 11 } \__unravel_scan_int:
4588     { 12 } \__unravel_scan_int:
4589     { 13 } \__unravel_scan_pdf_ext_toks:
4590     { 14 } \__unravel_scan_pdf_ext_toks:
4591     { 15 } \__unravel_scan_int:
4592     { 16 } \__unravel_scan_int:
4593     { 17 } \__unravel_scan_pdfstrcmp:
4594     { 18 } \__unravel_scan_pdfcolorstackinit:
4595     { 19 } \__unravel_scan_pdf_ext_toks:
4596     { 20 } \__unravel_scan_pdf_ext_toks:
4597     { 22 } \__unravel_scan_pdf_ext_toks:
4598     { 23 } \__unravel_scan_pdf_ext_toks:
4599     { 24 }
4600     {
4601         \__unravel_scan_keyword:n { fFillLeE }
4602         \__unravel_scan_pdf_ext_toks:
4603     }
4604     { 25 } \__unravel_scan_pdffiledump:
4605     { 26 } \__unravel_scan_pdfmatch:
4606     { 27 } \__unravel_scan_int:
4607     { 28 } \__unravel_scan_int:
4608     { 30 } \__unravel_scan_int:
4609     { 31 } \__unravel_scan_pdfximagebbox:
4610     { 33 } \__unravel_scan_directlua:
4611     { 34 } \__unravel_scan_pdf_ext_toks:
4612     { 35 } \__unravel_scan_pdf_ext_toks:
4613     { 40 }
4614     {
4615         \__unravel_scan_int:
4616         \__unravel_prev_input_silent:n { ~ }
4617         \__unravel_scan_int:
4618     }
4619 }
4620 \__unravel_prev_input_gpop:N \l__unravel_head_tl
4621 \__unravel_back_input_tl_o:
4622 }
4623 \cs_new_protected:Npn \__unravel_convert_string:
4624 {
4625     \__unravel_get_next:
4626     \tl_if_empty:NTF \l__unravel_head_tl
4627     { \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_head_gtl } }
4628     { \__unravel_prev_input:V \l__unravel_head_tl }
4629 }
4630 \cs_new_protected:Npn \__unravel_convert_meaning:
4631 {
4632     \__unravel_get_next:
4633     \tl_if_empty:NTF \l__unravel_head_tl

```

```

4634     { \__unravel_prev_input:n { \l__unravel_head_token } }
4635     { \__unravel_prev_input:V \l__unravel_head_tl }
4636   }
4637 \cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
4638   {
4639     \__unravel_scan_toks_to_str:
4640     \__unravel_scan_toks_to_str:
4641   }
4642 \cs_new_protected:Npn \__unravel_scan_pdfximagebbox:
4643   { \__unravel_scan_int: \__unravel_scan_int: }
4644 \cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:
4645   {
4646     \__unravel_scan_keyword:nTF { pPaAgGeE }
4647     { \bool_set_true:N \l__unravel_tmpa_bool }
4648     { \bool_set_false:N \l__unravel_tmpb_bool }
4649     \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4650     { \__unravel_scan_keyword:n { pPaAgGeE } }
4651     \__unravel_scan_toks_to_str:
4652   }
4653 \cs_new_protected:Npn \__unravel_scan_pdffiledump:
4654   {
4655     \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
4656     \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
4657     \__unravel_scan_pdf_ext_toks:
4658   }
4659 \cs_new_protected:Npn \__unravel_scan_pdfmatch:
4660   {
4661     \__unravel_scan_keyword:n { iIcCaAsSeE }
4662     \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
4663     { \__unravel_scan_int: }
4664     \__unravel_scan_pdf_ext_toks:
4665     \__unravel_scan_pdf_ext_toks:
4666   }
4667 \sys_if_engine luatex:T
4668   {
4669     \cs_new_protected:Npn \__unravel_scan_directlua:
4670     {
4671       \__unravel_get_x_non_relax:
4672       \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
4673       { \__unravel_back_input: }
4674       {
4675         \__unravel_scan_int:
4676         \__unravel_get_x_non_relax:
4677       }
4678       \__unravel_scan_pdf_ext_toks:
4679     }
4680   }

```

__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.

```

4681 \__unravel_new_tex_expandable:nn { the } % 111
4682   { \__unravel_get_the:N }
4683 \cs_new_protected:Npn \__unravel_get_the:N #1
4684   {

```



```

4685 \__unravel_prev_input_gpush:N \l__unravel_head_tl
4686 \__unravel_print_expansion:
4687 \int_if_odd:nTF \l__unravel_head_char_int
4688 { % \unexpanded, \detokenize
4689   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4690   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4691   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4692 }
4693 { % \the
4694   \__unravel_get_x_next:
4695   \__unravel_scan_something_internal:n { 5 }
4696   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4697   \__unravel_set_action_text:x
4698   {
4699     \tl_head:N \l__unravel_head_tl
4700     => \tl_tail:N \l__unravel_head_tl
4701   }
4702   \tl_set:Nx \l__unravel_head_tl
4703   { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4704 }
4705 \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
4706 {
4707   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4708   \__unravel_prev_input_silent:x { \l__unravel_head_tl }
4709   \__unravel_print_action:
4710 }
4711 {
4712   \cs_if_eq:NNTF #1 \__unravel_get_token_x:
4713   {
4714     \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
4715     \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
4716   }
4717   {
4718     \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4719     \__unravel_back_input:V \l__unravel_tmpa_tl
4720   }
4721   \__unravel_print_expansion:
4722 }
4723 #1
4724 }

```

(End definition for __unravel_get_the:N.)

```

4725 \__unravel_new_tex_expandable:mn { top_bot_mark } % 112
4726 { \__unravel_back_input_tl_o: }
4727 \__unravel_new_tex_expandable:mn { end_template } % 117
4728 {
4729   \__unravel_not_implemented:n { end-template } { } { } { }
4730   \__unravel_back_input_tl_o:
4731 }

```

2.13.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w

```

```

4732 \cs_new_protected:Npn \__unravel_pass_text:
4733 {
4734   \__unravel_input_if_empty:TF
4735   { \__unravel_pass_text_empty: }
4736   {
4737     \__unravel_input_get:N \l__unravel_tmpb_gtl
4738     \if_true:
4739       \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
4740       \exp_after:wN \__unravel_pass_text_done:w
4741       \fi:
4742       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4743       \exp_after:wN \__unravel_pass_text:
4744     \else:
4745       \use:c { fi: }
4746       \int_set:Nn \l__unravel_if_nesting_int { 1 }
4747       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4748       \exp_after:wN \__unravel_pass_text_nested:
4749     \fi:
4750   }
4751 }
4752 \cs_new_protected:Npn \__unravel_pass_text_done:w
4753 {
4754   \__unravel_get_next:
4755   \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4756   \else:
4757 }

```

(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:.`

```

4758 \cs_new_protected:Npn \__unravel_pass_text_nested:
4759 {
4760   \__unravel_input_if_empty:TF
4761   { \__unravel_pass_text_empty: }
4762   {
4763     \__unravel_input_get:N \l__unravel_tmpb_gtl
4764     \if_true:
4765     \if_true:
4766       \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4767       \int_decr:N \l__unravel_if_nesting_int
4768       \use_none:nnnnn
4769     \fi:

```

```

4770         \use_none:nnn
4771     \fi:
4772     \int_incr:N \l__unravel_if_nesting_int
4773     \fi:
4774     \__unravel_input_gpop:N \l__unravel_unused_gtl
4775     \int_compare:nNnTF \l__unravel_if_nesting_int = 0
4776         { \__unravel_pass_text: }
4777         { \__unravel_pass_text_nested: }
4778     }
4779 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

4780 \cs_new_protected:Npn \__unravel_pass_text_empty:
4781 {
4782     \__unravel_error:nnnnn { runaway-if } { } { } { } { }
4783     \__unravel_exit_hard:w
4784 }

```

(End definition for __unravel_pass_text_empty:.)

__unravel_cond_push:

__unravel_cond_pop:

```

4785 \cs_new_protected:Npn \__unravel_cond_push:
4786 {
4787     \tl_gput_left:Nx \g__unravel_if_limit_tl
4788     { { \int_use:N \g__unravel_if_limit_int } }
4789     \int_gincr:N \g__unravel_if_depth_int
4790     \int_gzero:N \g__unravel_if_limit_int
4791 }
4792 \cs_new_protected:Npn \__unravel_cond_pop:
4793 {
4794     \int_gset:Nn \g__unravel_if_limit_int
4795     { \tl_head:N \g__unravel_if_limit_tl }
4796     \tl_gset:Nx \g__unravel_if_limit_tl
4797     { \tl_tail:N \g__unravel_if_limit_tl }
4798     \int_gdecr:N \g__unravel_if_depth_int
4799 }

```

(End definition for __unravel_cond_push: and __unravel_cond_pop:.)

__unravel_change_if_limit:nn

```

4800 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
4801 {
4802     \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4803     { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4804     {
4805         \tl_clear:N \l__unravel_tmpa_tl
4806         \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
4807         {
4808             \tl_put_right:Nx \l__unravel_tmpa_tl
4809             { { \tl_head:N \g__unravel_if_limit_tl } }
4810             \tl_gset:Nx \g__unravel_if_limit_tl
4811             { \tl_tail:N \g__unravel_if_limit_tl }
4812         }

```

```

4813     \tl_gset:Nx \g__unravel_if_limit_tl
4814     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4815   }
4816 }

```

(End definition for __unravel_change_if_limit:nn.)

```

4817 \__unravel_new_tex_expandable:nn { if_test } % 107
4818 {
4819   \__unravel_cond_push:
4820   \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
4821 }

```

__unravel_cond_aux:nn

```

4822 \cs_new_protected:Npn \__unravel_cond_aux:n #1
4823 {
4824   \int_case:nnF \l__unravel_head_char_int
4825   {
4826     { 0 } { \__unravel_test_two_chars:nn { 0 } {#1} } % if
4827     { 1 } { \__unravel_test_two_chars:nn { 1 } {#1} } % ifcat
4828     { 12 } { \__unravel_test_ifx:n {#1} }
4829     { 16 } { \__unravel_test_case:n {#1} }
4830     { 21 } { \__unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4831   }
4832   {
4833     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4834     \__unravel_print_expansion:
4835     \int_case:nn \l__unravel_head_char_int
4836     {
4837       { 2 } % ifnum
4838       { \__unravel_test_two_vals:N \__unravel_scan_int: }
4839       { 3 } % ifdim
4840       { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4841       { 4 } { \__unravel_scan_int: } % ifodd
4842       % { 5 } { } % ifvmode
4843       % { 6 } { } % ifhmode
4844       % { 7 } { } % ifmmode
4845       % { 8 } { } % ifinner
4846       { 9 } { \__unravel_scan_int: } % ifvoid
4847       { 10 } { \__unravel_scan_int: } % ifhbox
4848       { 11 } { \__unravel_scan_int: } % ifvbox
4849       { 13 } { \__unravel_scan_int: } % ifeof
4850       % { 14 } { } % iftrue
4851       % { 15 } { } % iffalse
4852       { 17 } { \__unravel_test_ifdefined: } % ifdefined
4853       { 18 } { \__unravel_test_ifcsname: } % ifcsname
4854       { 19 } % iffontchar
4855       { \__unravel_scan_font_ident: \__unravel_scan_int: }
4856       % { 20 } { } % ifincsname % ^^A todo: something?
4857       { 22 } % ifpdfabsnum
4858       { \__unravel_test_two_vals:N \__unravel_scan_int: }
4859       { 23 } % ifpdfabsdim
4860       { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4861     }
4862     \__unravel_prev_input_gpop:N \l__unravel_head_tl

```

```

4863     \_unravel_set_action_text:x { \tl_to_str:N \l_unravel_head_tl }
4864     \l_unravel_head_tl \scan_stop:
4865         \exp_after:wN \_unravel_cond_true:n
4866     \else:
4867         \exp_after:wN \_unravel_cond_false:n
4868     \fi:
4869     {#1}
4870 }
4871 }

```

(End definition for _unravel_cond_aux:nn.)

_unravel_cond_true:n

```

4872 \cs_new_protected:Npn \_unravel_cond_true:n #1
4873 {
4874     \_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4875     \_unravel_print_expansion:x { \g_unravel_action_text_str = true }
4876 }

```

(End definition for _unravel_cond_true:n.)

_unravel_cond_false:n

_unravel_cond_false_loop:n

_unravel_cond_false_common:

```

4877 \cs_new_protected:Npn \_unravel_cond_false:n #1
4878 {
4879     \_unravel_cond_false_loop:n {#1}
4880     \_unravel_cond_false_common:
4881     \_unravel_print_expansion:x
4882     {
4883         \g_unravel_action_text_str = false ~
4884         => ~ skip ~ to ~ \tl_to_str:N \l_unravel_head_tl
4885     }
4886 }
4887 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4888 {
4889     \_unravel_pass_text:
4890     \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
4891     {
4892         \token_if_eq_meaning:NNT \l_unravel_head_token \or:
4893         {
4894             \_unravel_error:nnnnn { extra-or } { } { } { } { }
4895             \_unravel_cond_false_loop:n {#1}
4896         }
4897     }
4898     {
4899         \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
4900         { \_unravel_cond_pop: }
4901         \_unravel_cond_false_loop:n {#1}
4902     }
4903 }
4904 \cs_new_protected:Npn \_unravel_cond_false_common:
4905 {
4906     \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
4907     { \_unravel_cond_pop: }
4908     { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
4909 }

```

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

_unravel_test_two_vals:N

```

4910 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
4911 {
4912   #1
4913   \_unravel_get_x_non_blank:
4914   \_unravel_tl_if_in:ooTF { < = > } \l__unravel_head_tl { }
4915   {
4916     \_unravel_error:nmnnn { missing-equals } { } { } { } { }
4917     \_unravel_back_input:
4918     \tl_set:Nn \l__unravel_head_tl { = }
4919   }
4920   \_unravel_prev_input:V \l__unravel_head_tl
4921   #1
4922 }

```

(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

```

4923 \cs_new_protected:Npn \_unravel_test_two_chars:nn #1
4924 {
4925   \_unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4926   \_unravel_print_expansion:
4927   \_unravel_test_two_chars_get:n {#1}
4928   \_unravel_test_two_chars_get:n {#1}
4929   \_unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
4930   \_unravel_set_action_text:x { \gtl_to_str:N \l__unravel_head_gtl }
4931   \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
4932   \gtl_pop_left:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4933   \_unravel_test_two_chars_gtl:N \l__unravel_tmpb_gtl
4934   \_unravel_test_two_chars_gtl:N \l__unravel_head_gtl
4935   \l__unravel_head_tl \scan_stop:
4936   \exp_after:wN \_unravel_cond_true:n
4937   \else:
4938   \exp_after:wN \_unravel_cond_false:n
4939   \fi:
4940 }
4941 \cs_new_protected:Npn \_unravel_test_two_chars_get:n #1
4942 {
4943   \_unravel_get_x_next:
4944   \int_compare:nNnT {#1} = 0
4945   {
4946     \gtl_if_head_is_N_type:NF \l__unravel_head_gtl
4947     { \gtl_set:Nx \l__unravel_head_gtl { \gtl_to_str:N \l__unravel_head_gtl } }
4948   }
4949   \_unravel_prev_input_gtl:N \l__unravel_head_gtl
4950   \_unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
4951 }
4952 \cs_new_protected:Npn \_unravel_test_two_chars_gtl:N #1
4953 {
4954   \tl_put_right:Nx \l__unravel_head_tl
4955   {

```

```

4956     \gtl_if_head_is_group_begin:NTF #1 { \c_group_begin_token }
4957     {
4958         \gtl_if_head_is_group_end:NTF #1 { \c_group_end_token }
4959         {
4960             \exp_not:N \exp_not:N
4961             \exp_not:f { \gtl_head_do:NN #1 \exp_stop_f: }
4962         }
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:NNTF
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:nnnnn { extra-fi-or-else } { } { } { } { } }
5076         { \_unravel_insert_relax: }
5077       }
5078       { \_unravel_error:nnnnn { 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         => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
5089       }
5090     }
5091 }
```

```

5091     \__unravel_print_expansion:
5092     \__unravel_cond_pop:
5093   }
5094 }
5095 \cs_new_protected:Npn \__unravel_fi_or_else_loop:
5096 {
5097   \int_compare:nNnF \l__unravel_head_char_int = 2
5098   {
5099     \__unravel_pass_text:
5100     \__unravel_set_cmd:
5101     \__unravel_fi_or_else_loop:
5102   }
5103 }

```

2.14 User interaction

2.14.1 Print

Let us start with the procedure which prints to the terminal: this will help me test the code while I'm writing it.

`__unravel_print_normalize_null:` Change the null character to an explicit `^^@` in LuaTeX to avoid a bug whereby a null character ends a string prematurely.

`\l__unravel_print_tl`

```

5104 \tl_new:N \l__unravel_print_tl
5105 \sys_if_engine luatex:TF
5106 {
5107   \cs_new_protected:Npx \__unravel_print_normalize_null:
5108   {
5109     \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
5110     { \char_generate:nn { 0 } { 12 } }
5111     { \tl_to_str:n { ^^@ } }
5112   }
5113 }
5114 { \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`

```

5115 \cs_new_protected:Npn \__unravel_print:n #1
5116 {
5117   \tl_set:Nn \l__unravel_print_tl {#1}
5118   \__unravel_print_normalize_null:
5119   \__unravel_exp_args:Nx \iow_term:n { \l__unravel_print_tl }
5120 }
5121 \cs_new_protected:Npn \__unravel_print:x
5122 { \__unravel_exp_args:Nx \__unravel_print:n }
5123 \cs_new_protected:Npn \__unravel_log:n #1
5124 {
5125   \tl_set:Nn \l__unravel_print_tl {#1}
5126   \__unravel_print_normalize_null:
5127   \__unravel_exp_args:Nx \iow_log:n { \l__unravel_print_tl }
5128 }

```

(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`.

```

5129 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
5130 {
5131   \int_compare:nNnF \g__unravel_online_int < 0
5132   {
5133     \int_compare:nNnTF \g__unravel_online_int = 0
5134     { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_log:n }
5135     { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_print:n }
5136   }
5137 }

```

(End definition for `__unravel_print_message:nn`.)

`__unravel_set_action_text:x`

```

5138 \cs_new_protected:Npn \__unravel_set_action_text:x #1
5139 {
5140   \group_begin:
5141   \__unravel_set_escapechar:n { 92 }
5142   \str_gset:Nx \g__unravel_action_text_str {#1}
5143   \group_end:
5144 }

```

(End definition for `__unravel_set_action_text:x`.)

`__unravel_set_action_text:`

```

5145 \cs_new_protected:Npn \__unravel_set_action_text:
5146 {
5147   \__unravel_set_action_text:x
5148   {
5149     \tl_to_str:N \l__unravel_head_tl
5150     \tl_if_single_token:VT \l__unravel_head_tl
5151     { = ~ \token_to_meaning:N \l__unravel_head_token }
5152   }
5153 }

```

(End definition for `__unravel_set_action_text:.`)

`__unravel_print_state:`

```

5154 \cs_new_protected:Npn \__unravel_print_state:
5155 {
5156   \group_begin:
5157   \__unravel_set_escapechar:n { 92 }
5158   \tl_use:N \g__unravel_before_print_state_tl
5159   \int_compare:nNnT \g__unravel_online_int > 0
5160   {
5161     \__unravel_print_state_output:
5162     \__unravel_print_state_prev:
5163     \__unravel_print_state_input:
5164   }
5165   \group_end:
5166 }

```

(End definition for `__unravel_print_state:.`)

`_unravel_print_state_output:` Unless empty, print #1 with each line starting with <|~. The `_unravel_str_truncate_left:nn` function trims #1 if needed, to fit in a maximum of `\g__unravel_max_output_int` characters.

```

5167 \cs_new_protected:Npn \_unravel_print_state_output:
5168   {
5169     \_unravel_exp_args:Nx \_unravel_print_state_output:n
5170     { \gtl_to_str:N \g__unravel_output_gtl }
5171   }
5172 \cs_new_protected:Npn \_unravel_print_state_output:n #1
5173   {
5174     \tl_if_empty:nF {#1}
5175     {
5176       \_unravel_print_message:nn { <| ~ }
5177       { \_unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
5178     }
5179   }

```

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

`_unravel_print_state_prev:` Never trim ##1.

```

5180 \cs_new_protected:Npn \_unravel_print_state_prev:
5181   {
5182     \seq_set_map:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
5183     { \_unravel_to_str:n {##1} }
5184     \seq_remove_all:Nn \l__unravel_tmpa_seq { }
5185     \seq_if_empty:NTF \l__unravel_tmpa_seq
5186     { \_unravel_print_message:nn { || ~ } { } }
5187     {
5188       \seq_map_inline:Nn \l__unravel_tmpa_seq
5189       {
5190         \_unravel_print_message:nn { || ~ } {##1}
5191       }
5192     }
5193   }

```

(End definition for _unravel_print_state_prev:.)

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

```

5194 \cs_new_protected:Npn \_unravel_print_state_input:
5195   {
5196     \_unravel_exp_args:Nx \_unravel_print_state_input:n
5197     { \_unravel_input_to_str: }
5198   }
5199 \cs_new_protected:Npn \_unravel_print_state_input:n #1
5200   {
5201     \_unravel_print_message:nn { |> ~ }
5202     { \_unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
5203   }

```

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

__unravel_print_meaning:

```
5204 \cs_new_protected:Npn \__unravel_print_meaning:
5205 {
5206   \__unravel_input_if_empty:TF
5207     { \__unravel_print_message:nn { } { Empty-input! } }
5208     {
5209       \__unravel_input_get:N \l__unravel_tmpb_gtl
5210       \__unravel_print_message:nn { }
5211       {
5212         \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
5213         = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
5214       }
5215     }
5216 }
```

(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: 5217 \cs_new_protected:Npn \__unravel_print_action:
   \__unravel_print_assignment:x 5218 { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
\__unravel_print_expansion: 5219 \cs_new_protected:Npn \__unravel_print_action:x #1
   \__unravel_print_expansion:x 5220 {
   \__unravel_print_action_aux:N 5221   \__unravel_set_action_text:x {#1}
   \__unravel_print_action: 5222   \__unravel_print_action:
5223 }
5224 \cs_new_protected:Npn \__unravel_print_assignment:
5225 { \__unravel_print_action_aux:N \g__unravel_trace_assigns_bool }
5226 \cs_new_protected:Npn \__unravel_print_assignment:x #1
5227 {
5228   \__unravel_set_action_text:x {#1}
5229   \__unravel_print_assignment:
5230 }
5231 \cs_new_protected:Npn \__unravel_print_expansion:
5232 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
5233 \cs_new_protected:Npn \__unravel_print_expansion:x #1
5234 {
5235   \__unravel_set_action_text:x {#1}
5236   \__unravel_print_expansion:
5237 }
5238 \cs_new_protected:Npn \__unravel_print_action_aux:N #1
5239 {
5240   \int_gdecr:N \g__unravel_nonstop_int
5241   \int_gincr:N \g__unravel_step_int
5242   \bool_if:NT #1
5243   {
5244     \__unravel_print:x
5245     {
5246       [====
5247       \bool_if:NT \g__unravel_number_steps_bool
5248       { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
5249       =====]~
5250       \int_compare:nNnTF
5251       { \str_count:N \g__unravel_action_text_str }
```

```

5252         > { \g__unravel_max_action_int }
5253         {
5254             \str_range:Nnn \g__unravel_action_text_str
5255                 { 1 } { \g__unravel_max_action_int - 3 } ...
5256         }
5257         { \g__unravel_action_text_str }
5258     }
5259     \__unravel_print_state:
5260     \__unravel_prompt:
5261 }
5262 }

```

(End definition for __unravel_print_action: and others.)

__unravel_print_assigned_token:
__unravel_print_assigned_register:

```

5263 \cs_new_protected:Npn \__unravel_print_assigned_token:
5264 {
5265     \__unravel_after_assignment: % ^^A todo: simplify
5266     \__unravel_print_assignment:x
5267     {
5268         Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5269             = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
5270     }
5271     \__unravel_omit_after_assignment:w
5272 }
5273 \cs_new_protected:Npn \__unravel_print_assigned_register:
5274 {
5275     \__unravel_after_assignment: % ^^A todo: simplify
5276     \__unravel_exp_args:Nx \__unravel_print_assignment:x
5277     {
5278         \exp_not:n
5279         {
5280             Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5281                 \tl_if_single:NT \l__unravel_defined_tl
5282                 { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
5283         }
5284         = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
5285     }
5286     \__unravel_omit_after_assignment:w
5287 }

```

(End definition for __unravel_print_assigned_token: and __unravel_print_assigned_register:.)

__unravel_print_welcome: Welcome message.

```

5288 \cs_new_protected:Npn \__unravel_print_welcome:
5289 {
5290     \__unravel_print_message:nn { }
5291     {
5292         \bool_if:NTF \g__unravel_welcome_message_bool
5293         {
5294             \\\
5295             =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5296             \iow_indent:n
5297             {
5298                 "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\

```

```

5299         "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\
5300         "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
5301         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
5302     }
5303 }
5304 { [====~Start~====] }
5305 }
5306 \__unravel_print_state:
5307 \__unravel_prompt:
5308 }

```

(End definition for __unravel_print_welcome:.)

__unravel_print_outcome: Final message.

```

5309 \cs_new_protected:Npn \__unravel_print_outcome:
5310 { \__unravel_print_message:nn { } { [====~End~====] } }

```

(End definition for __unravel_print_outcome:.)

2.14.2 Prompt

__unravel_ior_str_get:NN
__unravel_ior_str_get:Nc

```

5311 \cs_new_protected:Npn \__unravel_ior_str_get:NN #1#2
5312 { \tex_readline:D #1 to #2 }
5313 \cs_generate_variant:Nn \__unravel_ior_str_get:NN { Nc }

```

(End definition for __unravel_ior_str_get:NN.)

__unravel_prompt:

```

5314 \cs_new_protected:Npn \__unravel_prompt:
5315 {
5316   \int_compare:nNnF \g__unravel_nonstop_int > 0
5317   {
5318     \group_begin:
5319     \__unravel_set_escapechar:n { -1 }
5320     \int_set:Nn \tex_endlinechar:D { -1 }
5321     \tl_use:N \g__unravel_before_prompt_tl
5322     \__unravel_prompt_aux:
5323     \group_end:
5324   }
5325 }
5326 \cs_new_protected:Npn \__unravel_prompt_aux:
5327 {
5328   \clist_if_empty:NTF \g__unravel_prompt_input_clist
5329   {
5330     \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
5331     {
5332       \bool_if:NTF \g__unravel_explicit_prompt_bool
5333       { \__unravel_ior_str_get:Nc \c__unravel_prompt_ior }
5334       { \__unravel_ior_str_get:Nc \c__unravel_noprompt_ior }
5335       { Your~input }
5336       \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5337     }
5338   }
5339   {

```

```

5340 \clist_gpop:NN \g__unravel_prompt_input_clist \l__unravel_tmpa_tl
5341 \group_begin:
5342 \__unravel_set_escapechar:n { 92 }
5343 \__unravel_print:x
5344 {
5345 \bool_if:NT \g__unravel_explicit_prompt_bool { Your~input= }
5346 \tl_to_str:N \l__unravel_tmpa_tl
5347 }
5348 \group_end:
5349 \exp_args:NW \__unravel_prompt_treat:n \l__unravel_tmpa_tl
5350 }
5351 }
5352 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5353 {
5354 \tl_if_empty:nF {#1}
5355 {
5356 \__unravel_exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
5357 {
5358 { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
5359 { q }
5360 {
5361 \int_gset:Nn \g__unravel_online_int { -1 }
5362 \int_gzero:N \g__unravel_nonstop_int
5363 }
5364 { x }
5365 {
5366 \group_end:
5367 \__unravel_exit_hard:w
5368 }
5369 { X }
5370 {
5371 \tex_batchmode:D
5372 \tex_read:D -1 to \l__unravel_tmpa_tl
5373 }
5374 { s } { \__unravel_prompt_scan_int:nn {#1}
5375 \__unravel_prompt_silent_steps:n }
5376 { o } { \__unravel_prompt_scan_int:nn {#1}
5377 { \int_gset:Nn \g__unravel_online_int } }
5378 { C }
5379 {
5380 \__unravel_exp_args:Nx \use:n
5381 {
5382 \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5383 { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5384 }
5385 \tl_gput_left:Nn \g__unravel_tmpc_tl
5386 { \tl_gclear:N \g__unravel_tmpc_tl }
5387 \group_insert_after:N \g__unravel_tmpc_tl
5388 \group_insert_after:N \__unravel_prompt:
5389 }
5390 { | } { \__unravel_prompt_scan_int:nn {#1}
5391 \__unravel_prompt_vert:n }
5392 { u } { \__unravel_prompt_until:n {#1} }
5393 { a } { \__unravel_prompt_all: }

```



```

5394     }
5395     { \_unravel_prompt_help: }
5396   }
5397 }
5398 \cs_new_protected:Npn \_unravel_prompt_scan_int:nn #1
5399 {
5400   \tex_afterassignment:D \_unravel_prompt_scan_int_after:wn
5401   \l_unravel_prompt_tmpa_int =
5402   \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5403   \use_ii:nn #1 \scan_stop:
5404 }
5405 \cs_new_protected:Npn \_unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5406 {
5407   #2 \l_unravel_prompt_tmpa_int
5408   \tl_if_blank:nF {#1} { \_unravel_prompt_treat:n {#1} }
5409 }
5410 \cs_new_protected:Npn \_unravel_prompt_help:
5411 {
5412   \_unravel_print:n { "m":~meaning-of-first-token }
5413   \_unravel_print:n { "a":~print-state-again,~without-truncating }
5414   \_unravel_print:n { "s<num>":~do-<num>-steps-silently }
5415   \_unravel_print:n { "|<num>":~silent-steps-until-<num>-fewer-||" }
5416   \_unravel_print:n { "u<text>":~silent-steps-until-the-input-starts-with-<text> }
5417   \_unravel_print:n
5418     { "o<num>":~1~=>~log-and-terminal,~0~=>~only-log,~-1~=>~neither.}
5419   \_unravel_print:n { "q":~semi-quiet~(same-as~"o-1") }
5420   \_unravel_print:n { "C<code>":~run-some-exp13-code-immediately }
5421   \_unravel_print:n { "x"/"X":~exit-this-instance-of-unravel/TeX }
5422   \_unravel_prompt_aux:
5423 }
5424 \cs_new_protected:Npn \_unravel_prompt_silent_steps:n #1
5425 {
5426   \int_compare:nNnF {#1} < 0
5427   {
5428     \int_gset:Nn \g_unravel_online_int { -1 }
5429     \tl_gset:Nn \g_unravel_before_prompt_tl
5430     {
5431       \int_gset:Nn \g_unravel_online_int { 1 }
5432       \tl_gclear:N \g_unravel_before_prompt_tl
5433     }
5434     \int_gset:Nn \g_unravel_nonstop_int {#1}
5435   }
5436 }
5437 \cs_new_protected:Npn \_unravel_prompt_vert:n #1
5438 {
5439   \int_compare:nNnTF {#1} < { 0 }
5440     { \_unravel_prompt_vert:Nn > {#1} }
5441     { \_unravel_prompt_vert:Nn < {#1} }
5442 }
5443 \cs_new_protected:Npn \_unravel_prompt_vert:Nn #1#2
5444 {
5445   \int_gset:Nn \g_unravel_online_int { -1 }
5446   \tl_gset:Nf \g_unravel_before_print_state_tl
5447   {

```

```

5448     \exp_args:NNf \exp_stop_f: \int_compare:nNnTF
5449     { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5450     #1 { \__unravel_prev_input_count: }
5451     {
5452       \int_gset:Nn \g__unravel_nonstop_int
5453       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5454     }
5455     {
5456       \int_gset:Nn \g__unravel_online_int { 1 }
5457       \tl_gclear:N \g__unravel_before_print_state_tl
5458     }
5459   }
5460 }
5461 \cs_new_protected:Npn \__unravel_prompt_all:
5462 {
5463   \tl_gset:Nx \g__unravel_tmpc_tl
5464   {
5465     \exp_not:n
5466     {
5467       \tl_gclear:N \g__unravel_tmpc_tl
5468       \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5469       \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5470       \__unravel_print_state:
5471       \int_gdecr:N \g__unravel_nonstop_int
5472       \__unravel_prompt:
5473     }
5474     \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5475     \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5476   }
5477   \group_insert_after:N \g__unravel_tmpc_tl
5478 }
5479 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5480 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for __unravel_prompt:.)

```

\__unravel_prompt_until:n
  \g__unravel_until_tl
5481 \tl_new:N \g__unravel_until_tl
5482 \cs_new_protected:Npn \__unravel_prompt_until:n #1
5483 {
5484   \tl_gset:Nx \g__unravel_until_tl { \tl_tail:n {#1} }
5485   \int_gset:Nn \g__unravel_online_int { -1 }
5486   \tl_gset:Nn \g__unravel_before_print_state_tl
5487   {
5488     \__unravel_input_get_left:N \l__unravel_tmpa_tl
5489     \__unravel_exp_args:Nx \use:n
5490     {
5491       \exp_not:N \tl_if_in:nnTF
5492       { \exp_not:N \__unravel:nn \tl_to_str:N \l__unravel_tmpa_tl }
5493       { \exp_not:N \__unravel:nn \tl_to_str:N \g__unravel_until_tl }
5494     }
5495     {
5496       \int_gzero:N \g__unravel_nonstop_int
5497       \int_gset:Nn \g__unravel_online_int { 1 }

```

```

5498         \tl_gclear:N \g__unravel_before_print_state_tl
5499     }
5500     {
5501         \int_gset:Nn \g__unravel_nonstop_int
5502             { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5503     }
5504 }
5505 }

```

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

2.14.3 Errors

`__unravel_not_implemented:n`

```

5506 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5507 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End definition for `__unravel_not_implemented:n`.)

`__unravel_error:nnnnn`
`__unravel_error:nxxxx`

Errors within a group to make sure that none of the `l3msg` variables (or others) that may be currently in use in the code being debugged are modified.

```

5508 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5509 {
5510     \group_begin:
5511     \msg_error:nnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5512     \group_end:
5513 }
5514 \cs_new_protected:Npn \__unravel_error:nxxxx #1#2#3#4#5
5515 {
5516     \group_begin:
5517     \msg_error:nxxxx { unravel } {#1} {#2} {#3} {#4} {#5}
5518     \group_end:
5519 }

```

(End definition for `__unravel_error:nnnnn`.)

`__unravel_tex_msg_new:nnn`

This stores a \TeX error message.

```

5520 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5521 {
5522     \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
5523     \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
5524 }

```

(End definition for `__unravel_tex_msg_new:nnn`.)

`__unravel_tex_error:nn`
`__unravel_tex_error:nV`

Throw the `tex-error` message, with arguments: `#2` which triggered the error, \TeX 's error message, and \TeX 's help text.

```

5525 \cs_new_protected:Npn \__unravel_tex_error:nn #1#2
5526 {
5527     \group_begin:
5528     \msg_error:nxxxx { unravel } { tex-error }
5529     { \tl_to_str:n {#2} }
5530     { \use:c { __unravel_tex_msg_error_#1: } }
5531     { \use:c { __unravel_tex_msg_help_#1: } }
5532     \group_end:

```

```

5533 }
5534 \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, `__unravel_tex_fatal_error:nV` `TeX`'s error message, and `TeX`'s help text.

```

5535 \cs_new_protected:Npn \__unravel_tex_fatal_error:nn #1#2
5536 {
5537   \__unravel_error:nxxxx { tex-fatal }
5538   { \tl_to_str:n {#2} }
5539   { \use:c { __unravel_tex_msg_error_#1: } }
5540   { \use:c { __unravel_tex_msg_help_#1: } }
5541   { }
5542 }
5543 \cs_generate_variant:Nn \__unravel_tex_fatal_error:nn { nV }

```

(End definition for `__unravel_tex_fatal_error:nn`.)

2.15 Keys

Each key needs to be defined twice: for its default setting and for its setting applying to a single `\unravel`. This is due to the fact that we cannot use grouping to keep settings local to a single `\unravel` since the `<code>` argument of `\unravel` may open or close groups.

```

5544 \keys_define:nn { unravel/defaults }
5545 {
5546   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5547   internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5548   max-action .int_gset:N = \g__unravel_default_max_action_int ,
5549   max-output .int_gset:N = \g__unravel_default_max_output_int ,
5550   max-input .int_gset:N = \g__unravel_default_max_input_int ,
5551   number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
5552   online .int_gset:N = \g__unravel_default_online_int ,
5553   prompt-input .clist_gset:N = \g__unravel_default_prompt_input_clist ,
5554   trace-assigns .bool_gset:N = \g__unravel_default_trace_assign_bool ,
5555   trace-expansion .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
5556   trace-other .bool_gset:N = \g__unravel_default_trace_other_bool ,
5557   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5558 }
5559 \keys_define:nn { unravel }
5560 {
5561   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5562   internal-debug .bool_gset:N = \g__unravel_internal_debug_bool ,
5563   max-action .int_gset:N = \g__unravel_max_action_int ,
5564   max-output .int_gset:N = \g__unravel_max_output_int ,
5565   max-input .int_gset:N = \g__unravel_max_input_int ,
5566   number-steps .bool_gset:N = \g__unravel_number_steps_bool ,
5567   online .int_gset:N = \g__unravel_online_int ,
5568   prompt-input .clist_gset:N = \g__unravel_prompt_input_clist ,
5569   trace-assigns .bool_gset:N = \g__unravel_trace_assigns_bool ,
5570   trace-expansion .bool_gset:N = \g__unravel_trace_expansion_bool ,
5571   trace-other .bool_gset:N = \g__unravel_trace_other_bool ,
5572   welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,

```

```
5573 }
```

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.

```
5574 \tl_map_inline:nn { { /defaults } { } }
5575 {
5576   \keys_define:nn { unravel #1 }
5577   {
5578     machine      .meta:nn =
5579     { unravel #1 }
5580     {
5581       explicit-prompt = false ,
5582       internal-debug  = false ,
5583       max-action      = \c_max_int ,
5584       max-output      = \c_max_int ,
5585       max-input       = \c_max_int ,
5586       number-steps    = false ,
5587       welcome-message = false ,
5588     } ,
5589     mute         .meta:nn =
5590     { unravel #1 }
5591     {
5592       trace-assigns = false ,
5593       trace-expansion = false ,
5594       trace-other = false ,
5595       welcome-message = false ,
5596       online = -1 ,
5597     }
5598   }
5599 }
```

2.16 Main command

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

```
5600 \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.

```
5601 \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.

```
5602 \cs_new_protected:Npn \unravel_setup:n #1
5603 {
5604   \keys_set:nn { unravel/defaults } {#1}
5605   \keys_set:nn { unravel } {#1}
5606 }
```

(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` in `\unravel` and avoid re-initializing important variables. Initialize and setup keys.
`__unravel:nn`
`__unravel_unravel_marker:` Initialize and setup other variables including the input. Welcome the user. Then comes the main loop: until the input is exhausted, print the current status and do one step. The main loop is exited by skipping to the first `__unravel_exit_point:`, while some abort procedures jump to the second (and last) one instead. If the main loop finished correctly, print its outcome and finally test that everything is all right.

```

5607 \cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
5608 \cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
5609 \cs_new_protected:Npn \__unravel:nn #1#2
5610 {
5611   \__unravel_init_key_vars:
5612   \keys_set:nn { unravel } {#1}
5613   \__unravel_init_vars:
5614   \__unravel_input_gset:n {#2}
5615   \__unravel_print_welcome:
5616   \__unravel_main_loop:
5617   \__unravel_exit_point:
5618   \__unravel_print_outcome:
5619   \__unravel_final_test:
5620   \__unravel_final_after_assignment:
5621   \__unravel_exit_point:
5622 }
5623 \cs_new_protected:Npn \unravel_get:nnN #1#2#3
5624 {
5625   \unravel:nn {#1} {#2}
5626   \tl_set:Nx #3 { \gtl_left_tl:N \g__unravel_output_gtl }
5627 }

```

(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).

```

5628 \cs_new_protected:Npn \__unravel_init_key_vars:
5629 {
5630   \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bool
5631   \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5632   \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5633   \int_gset_eq:NN \g__unravel_online_int \g__unravel_default_online_int
5634   \clist_gset_eq:NN \g__unravel_prompt_input_clist \g__unravel_default_prompt_input_clist
5635   \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
5636   \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
5637   \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
5638   \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
5639   \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5640   \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5641   \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
5642   \int_gzero:N \g__unravel_nonstop_int
5643 }

```

(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.

```

5644 \cs_new_protected:Npn \__unravel_init_vars:
5645 {
5646   \seq_gclear:N \g__unravel_prev_input_seq
5647   \gtl_gclear:N \g__unravel_output_gtl
5648   \int_gzero:N \g__unravel_step_int
5649   \tl_gclear:N \g__unravel_if_limit_tl
5650   \int_gzero:N \g__unravel_if_limit_int
5651   \int_gzero:N \g__unravel_if_depth_int
5652   \gtl_gclear:N \g__unravel_after_assignment_gtl
5653   \bool_gset_true:N \g__unravel_set_box_allowed_bool
5654   \bool_gset_false:N \g__unravel_name_in_progress_bool
5655   \gtl_clear:N \l__unravel_after_group_gtl
5656 }

```

(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.

```

5657 \cs_new_protected:Npn \__unravel_get_x_next_or_done:
5658 {
5659   \__unravel_input_if_empty:TF { \__unravel_exit:w } { }
5660   \__unravel_get_next:
5661   \__unravel_token_if_expandable:NT \l__unravel_head_token
5662   { \__unravel_expand_do:N \__unravel_get_x_next_or_done: }
5663 }
5664 \cs_new_protected:Npn \__unravel_main_loop:
5665 {
5666   \__unravel_get_x_next_or_done:
5667   \__unravel_set_cmd:
5668   \__unravel_do_step:
5669   \__unravel_main_loop:
5670 }

```

(End definition for __unravel_main_loop: and __unravel_get_x_next_or_done:.)

`__unravel_do_step:` Perform the action if the corresponding command exists. If that command does not exist, complain, and leave the token in the output.

```

5671 \cs_new_protected:Npn \__unravel_do_step:
5672 {
5673   \__unravel_set_action_text:
5674   \bool_if:NT \g__unravel_internal_debug_bool
5675   { \__unravel_exp_args:Nx \iow_term:n { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_in
5676   \cs_if_exist_use:cF
5677   { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
5678   { \__unravel_error:nxxxx { internal } { unknown-command } { } { } { } }
5679 }

```

(End definition for __unravel_do_step:.)

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

```

\__unravel_final_bad:
5680 \cs_new_protected:Npn \__unravel_final_test:
5681 {
5682   \bool_if:nTF
5683     {
5684       \tl_if_empty_p:N \g__unravel_if_limit_tl
5685       && \int_compare_p:nNn \g__unravel_if_limit_int = 0
5686       && \int_compare_p:nNn \g__unravel_if_depth_int = 0
5687       && \seq_if_empty_p:N \g__unravel_prev_input_seq
5688     }
5689     { \__unravel_input_if_empty:TF { } { \__unravel_final_bad: } }
5690     { \__unravel_final_bad: }
5691   }
5692 \cs_new_protected:Npn \__unravel_final_bad:
5693 {
5694   \__unravel_error:nnnnn { internal }
5695   { the-last-unravel-finished-badly } { } { } { }
5696 }

```

(End definition for `__unravel_final_test:` and `__unravel_final_bad:`.)

`__unravel_final_after_assignment:` Salvage any remaining `\afterassignment` token.

```

5697 \cs_new_protected:Npn \__unravel_final_after_assignment:
5698 {
5699   \gtl_if_empty:NF \g__unravel_after_assignment_gtl
5700   { \gtl_head_do:NN \g__unravel_after_assignment_gtl \tex_afterassignment:D }
5701 }

```

(End definition for `__unravel_final_after_assignment:`.)

2.17 Messages

```

5702 \msg_new:nnn { unravel } { unknown-primitive }
5703 { Internal~error:~the-primitive~'#1'~is-not-known. }
5704 \msg_new:nnn { unravel } { extra-fi-or-else }
5705 { Extra-fi,~or,~or-else. }
5706 \msg_new:nnn { unravel } { missing-dollar }
5707 { Missing~dollar~inserted. }
5708 \msg_new:nnn { unravel } { unknown-expandable }
5709 { Internal~error:~the-expandable-command~'#1'~is-not-known. }
5710 \msg_new:nnn { unravel } { missing-font-id }
5711 { Missing~font~identifier.~\iow_char:N\nullfont~inserted. }
5712 \msg_new:nnn { unravel } { missing-rparen }
5713 { Missing~right~parenthesis~inserted~for~expression. }
5714 \msg_new:nnn { unravel } { missing-cs }
5715 { Missing~control~sequence.~\iow_char:N\inaccessible~inserted. }
5716 \msg_new:nnn { unravel } { missing-box }
5717 { Missing~box~inserted. }
5718 \msg_new:nnn { unravel } { missing-to }
5719 { Missing~keyword~'to'~inserted. }
5720 \msg_new:nnn { unravel } { improper-leaders }
5721 { Leaders~not~followed~by~proper~glue. }
5722 \msg_new:nnn { unravel } { extra-close }
5723 { Extra~right~brace~or~\iow_char:N\endgroup. }

```



```

5724 \msg_new:nnn { unravel } { off-save }
5725 { Something~is~wrong~with~groups. }
5726 \msg_new:nnn { unravel } { hrule-bad-mode }
5727 { \iow_char:\hrule~used~in~wrong~mode. }
5728 \msg_new:nnn { unravel } { invalid-mode }
5729 { Invalid~mode~for~this~command. }
5730 \msg_new:nnn { unravel } { color-stack-action-missing }
5731 { Missing~color~stack~action. }
5732 \msg_new:nnn { unravel } { action-type-missing }
5733 { Missing~action~type. }
5734 \msg_new:nnn { unravel } { identifier-type-missing }
5735 { Missing~identifier~type. }
5736 \msg_new:nnn { unravel } { destination-type-missing }
5737 { Missing~destination~type. }
5738 \msg_new:nnn { unravel } { erroneous-prefixes }
5739 { Prefixes~applied~to~non~assignment~command. }
5740 \msg_new:nnn { unravel } { improper-setbox }
5741 { \iow_char:N\setbox~while~fetching~base~of~an~accent. }
5742 \msg_new:nnn { unravel } { after-advance }
5743 {
5744   Missing~register~after~\iow_char:N\advance,~
5745   \iow_char:N\multiply,~or~\iow_char:N\divide.
5746 }
5747 \msg_new:nnn { unravel } { bad-unless }
5748 { \iow_char:N\unless~not~followed~by~conditional. }
5749 \msg_new:nnn { unravel } { runaway-if }
5750 { Runaway~\iow_char:N\if...~Exiting~\iow_char:N\unravel }
5751 \msg_new:nnn { unravel } { runaway-macro-parameter }
5752 {
5753   Runaway~macro~parameter~\# #2~after \\\
5754   \iow_indent:n {#1}
5755 }
5756 \msg_new:nnn { unravel } { runaway-text }
5757 { Runaway~braced~argument~for~TeX~primitive.~Exiting~\iow_char:N\unravel }
5758 \msg_new:nnn { unravel } { extra-or }
5759 { Extra~\iow_char:N\or. }
5760 \msg_new:nnn { unravel } { missing-equals }
5761 { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
5762 \msg_new:nnn { unravel } { internal }
5763 { Internal~error:~'#1'.~\ Please~report. }
5764 \msg_new:nnn { unravel } { not-implemented }
5765 { The~following~feature~is~not~implemented:~'#1'. }
5766 \msg_new:nnn { unravel } { endinput-ignored }
5767 { The~primitive~\iow_char:N\endinput~was~ignored. }
5768 \msg_new:nnn { unravel } { missing-something }
5769 { Something~is~missing,~sorry! }
5770 \msg_new:nnn { unravel } { nested-unravel }
5771 { The~\iow_char:N\unravel~command~may~not~be~nested. }
5772 \msg_new:nnnn { unravel } { tex-error }
5773 { TeX~sees~"#1"~and~throws~an~error:~\\\ \iow_indent:n {#2} }
5774 {
5775   \tl_if_empty:nTF {#3}
5776   { TeX~provides~no~further~help~for~this~error. }
5777   { TeX's~advice~is:~\\\ \iow_indent:n {#3} }

```

```

5778 }
5779 \msg_new:nnnn { unravel } { tex-fatal }
5780 { TeX~sees~"#1"~and~throws~a~fatal~error:\\\\ \iow_indent:n {#2} }
5781 {
5782   \tl_if_empty:nTF {#3}
5783     { TeX~provides~no~further~help~for~this~error. }
5784     { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
5785 }
5786 \msg_new:nnnn { unravel } { runaway-unravel }
5787 { Runaway~\iow_char:N\\unravel,~so~\iow_char:N\\relax~inserted. }
5788 {
5789   Some~TeX~command~expects~input~beyond~the~end~of~
5790   the~argument~of~\iow_char:N\\unravel.
5791 }

```

Some error messages from T_EX itself.

```

5792 \__unravel_tex_msg_new:nnn { forbidden-case }
5793 {
5794   You~can't~use~'\exp_after:wN \token_to_str:N \l__unravel_head_tl'~in-
5795   \mode_if_vertical:TF { vertical }
5796   {
5797     \mode_if_horizontal:TF { horizontal }
5798     { \mode_if_math:TF { math } { no } }
5799   } ~ mode.
5800 }
5801 {
5802   Sorry,~but~I'm~not~programmed~to~handle~this~case;~
5803   I'll~just~pretend~that~you~didn't~ask~for~it.~
5804   If~you're~in~the~wrong~mode,~you~might~be~able~to~
5805   return~to~the~right~one~by~typing~'\iow_char:N\\}'~or~
5806   '\iow_char:N\\$'~or~'\iow_char:N\\par'.
5807 }
5808 \__unravel_tex_msg_new:nnn { incompatible-mag }
5809 {
5810   Incompatible~magnification~
5811   ( \int_to_arabic:n { \__unravel_mag: } );~
5812   the~previous~value~will~be~retained
5813 }
5814 {
5815   I~can~handle~only~one~magnification~ratio~per~job.~So~I've~
5816   reverted~to~the~magnification~you~used~earlier~on~this~run.
5817 }
5818 \__unravel_tex_msg_new:nnn { illegal-mag }
5819 {
5820   Illegal~magnification~has~been~changed~to~1000~
5821   ( \int_to_arabic:n { \__unravel_mag: } )
5822 }
5823 { The~magnification~ratio~must~be~between~1~and~32768. }
5824 \__unravel_tex_msg_new:nnn { missing-number }
5825 { Missing~number,~treated~as~zero }
5826 {
5827   A~number~should~have~been~here;~I~inserted~'0'.~
5828   If~you~can't~figure~out~why~I~needed~to~see~a~number,~
5829   look~up~'weird~error'~in~the~index~to~The~TeXbook.
5830 }

```

```

5831 \_unravel_tex_msg_new:nnn { the-cannot }
5832 { You-can't-use~'\tl_to_str:N\l_unravel_head_tl'~after~\iow_char:N\the }
5833 { I'm-forgetting-what-you-said-and-using-zero-instead. }
5834 \_unravel_tex_msg_new:nnn { incompatible-units }
5835 { Incompatible-glue-units }
5836 { I'm-going-to-assume-that~1mu=1pt-when-they're-mixed. }
5837 \_unravel_tex_msg_new:nnn { missing-mu }
5838 { Illegal-unit-of-measure~(mu-inserted) }
5839 {
5840   The-unit-of-measurement-in-math-glue-must-be-mu.~
5841   To-recover-gracefully-from-this-error,-it's-best-to~
5842   delete-the-erroneous-units;~e.g.,~type-'2'-to-delete~
5843   two-letters.~(See-Chapter-27-of-The-Texbook.)
5844 }
5845 \_unravel_tex_msg_new:nnn { missing-pt }
5846 { Illegal-unit-of-measure~(pt-inserted) }
5847 {
5848   Dimensions-can-be-in-units-of-em,~ex,~in,~pt,~pc,~
5849   cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but-yours-is-a-new-one!~
5850   I'll-assume-that-you-meant-to-say-pt,~for-printer's-points.~
5851   To-recover-gracefully-from-this-error,-it's-best-to~
5852   delete-the-erroneous-units;~e.g.,~type-'2'-to-delete~
5853   two-letters.~(See-Chapter-27-of-The-Texbook.)
5854 }
5855 \_unravel_tex_msg_new:nnn { missing-lbrace }
5856 { Missing~\iow_char:N\{~inserted }
5857 {
5858   A-left-brace-was-mandatory-here,~so-I've-put-one-in.~
5859   You-might-want-to-delete-and/or-insert-some-corrections~
5860   so-that-I-will-find-a-matching-right-brace-soon.~
5861   (If-you're-confused-by-all-this,~try-typing-'I\iow_char:N\}'~now.)
5862 }
5863 \_unravel_tex_msg_new:nnn { extra-endcsname }
5864 { Extra~\token_to_str:c{endcsname} }
5865 { I'm-ignoring-this,~since-I-wasn't-doing-a~\token_to_str:c{csname}. }
5866 \_unravel_tex_msg_new:nnn { missing-endcsname }
5867 { Missing~\token_to_str:c{endcsname}-inserted }
5868 {
5869   The-control-sequence-marked~<to-be-read-again>~should~
5870   not-appear-between~\token_to_str:c{csname}~and~
5871   \token_to_str:c{endcsname}.
5872 }

```

Fatal T_EX error messages.

```

5873 \_unravel_tex_msg_new:nnn { cannot-read }
5874 { ***~(cannot~\iow_char:N\read-from-terminal-in-nonstop-modes) }
5875 { }
5876 \_unravel_tex_msg_new:nnn { file-error }
5877 { ***~(job-aborted,~file-error-in-nonstop-mode) }
5878 { }
5879 \_unravel_tex_msg_new:nnn { interwoven-preambles }
5880 { (interwoven-alignment-preambles-are-not-allowed) }
5881 { }

```

Restore catcodes to their original values.

```
5882 \_unravel_setup_restore:  
5883 </package>
```