

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2015/09/22

Contents

1	<code>unravel</code> documentation	1
1.1	Options	3
1.2	Differences between <code>unravel</code> and TeX's processing	3
1.3	Future perhaps	4
2	<code>unravel</code> implementation	4
2.1	Primitives, variants, and helpers	4
2.1.1	Renamed primitives	4
2.1.2	Variants	5
2.1.3	Miscellaneous helpers	6
2.1.4	String helpers	6
2.1.5	Helpers for control flow	8
2.1.6	Helpers concerning tokens	9
2.1.7	Helpers for previous input	11
2.2	Variables	12
2.2.1	User interaction	12
2.2.2	Working with tokens	14
2.2.3	Numbers and conditionals	16
2.2.4	Boxes and groups	16
2.2.5	Constants	17
2.3	Numeric codes	17
2.4	Get next token	31
2.5	Manipulating the input	36
2.5.1	Elementary operations	36
2.5.2	Insert token for error recovery	42
2.5.3	Macro calls	42
2.6	Expand next token	44
2.7	Basic scanning subroutines	45

*This file has version number 0.1, last revised 2015/09/22.

2.8	Working with boxes	65
2.9	Paragraphs	69
2.10	Groups	71
2.11	Modes	73
2.12	One step	75
2.13	Commands	75
2.13.1	Characters: from 0 to 15	75
2.13.2	Boxes: from 16 to 31	79
2.13.3	From 32 to 47	85
2.13.4	Maths: from 48 to 56	88
2.13.5	From 57 to 70	89
2.13.6	Extensions	92
2.13.7	Assignments	99
2.14	Expandable primitives	109
2.14.1	Conditionals	115
2.15	User interaction	123
2.15.1	Print	123
2.15.2	Prompt	128
2.16	Keys	131
2.17	Main command	132
2.18	Messages	134

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.

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

This command shows in the terminal the steps performed by T_EX when running the *<code>*. By default, it pauses to let the user read the description of every step: simply press `<return>` to proceed. Typing `s<integer>` instead will go forward *<integer>* steps somewhat silently. In the future it will be possible to use a negative *<integer>* to go back a few steps. Typing `h` gives a list of various other possibilities. The available *<key-value>* options are described in Section 1.1.

`\unravelsetup` `\unravelsetup {<options>}`

Sets *<options>* that apply to all subsequent `\unravel`. See options in Section 1.1.

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

See `\unravel`.

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

See `\unravelsetup`.

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 1.2, should be reported on the issue tracker (<https://github.com/blefloch/latex-unravel/issues>).

As a simple example, one can run L^AT_EX 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 ten minutes on my machine, and finishes after slightly more than 20000 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 as many steps to complete, is four times slower, because `\newcommand` uses delimited arguments, which prevent some optimizations that `unravel` can otherwise obtain. For comparison, `\unravel{\lipsum[1-30]}`, which also takes 20000 step, takes 8 minutes to complete.

1.1 Options

<u>explicit-prompt</u>	Boolean option (default <code>false</code>) determining whether to give an explicit prompt. If <code>true</code> , the text “ <code>Your input=</code> ” will appear at the beginning of lines where user input is expected.
<u>internal-debug</u>	Boolean option (default <code>false</code>) used to debug <code>unravel</code> itself.
<u>machine</u>	Option which takes no value and makes <code>unravel</code> produce an output that is somewhat more suitable for automatic processing. In particular, it sets <code>max-action</code> , <code>max-output</code> , <code>max-input</code> to very large values, and <code>number-steps</code> to <code>false</code> .
<u>max-action max-output max-input</u>	Integer options (defaults 50, 300, 300) determining the maximum number of characters displayed for the action, the output part, and the input part.
<u>number-steps</u>	Boolean option (default <code>true</code>) determining whether to number steps.
<u>welcome-message</u>	Boolean option (default <code>true</code>) determining whether to display the welcome message.

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

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

- Some primitives are not implemented yet: alignments (`\halign`, `\valign`, `\noalign`, `\omit`, `\span`, `\cr`, `\crrcr`, &), many math mode primitives, and `\pdfprimitive`, `\aftergroup`, `\discretionary`, as well as all primitives specific to engines other than pdf`TEX`. This list is sadly incomplete!
- For `unravel`, category codes are fixed when a file is read using `\input`, while `TEX` only fixes category codes when the corresponding characters are converted to tokens. Similarly, the argument of `\scantokens` is converted to the new category code régime in one go, and the result must be balanced.
- Explicit begin-group and end-group characters other than the usual left and right braces may make `unravel` choke, or may be silently replaced by the usual left and right braces.
- `\endinput` is ignored with a warning, as it is very difficult to implement it in a way similar to `TEX`'s, and as it is most often used at the very end of files, in a redundant way.
- `\outer` is not supported.

1.3 Future perhaps

- Allow users to change some settings globally/for one `\unravel`.
- Allow to replay steps that have already been run.
- Fix the display for `\if` and `\ifcat` (remove extraneous `\exp_not:N`).

2 `unravel` implementation

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

```
1 <*package>
2 \RequirePackage{expl3,xparse}[2015/09/11]
3 \RequirePackage{gtl}[2015/09/21]
4 \ProvidesExplPackage
5   {unravel} {2015/09/22} {0.1} {Watching TeX digest tokens}

   Load l3str if expl3 is too old and does not define \str_range:nnn. Otherwise loading
   l3str would give an error.
6 \cs_if_exist:NF \str_range:nnn { \RequirePackage{l3str} }
7 <@@=unravel>
```

2.1 Primitives, variants, and helpers

2.1.1 Renamed primitives

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

```

\__unravel_everyeof:w
\__unravel_everypar:w
\__unravel_set_escapechar:n
\__unravel_nullfont:
\__unravel_hbox:w
\__unravel_the:w
8 \cs_new_eq:NN \__unravel_currentgrouptype: \etex_currentgrouptype:D
9 \cs_new_protected_nopar:Npn \__unravel_set_escapechar:n
10 { \int_set:Nn \tex_escapechar:D }
11 \cs_new_eq:NN \__unravel_everyeof:w \etex_everyeof:D
12 \cs_new_eq:NN \__unravel_everypar:w \tex_everypar:D
13 \cs_new_eq:NN \__unravel_hbox:w \tex_hbox:D
14 \cs_new_eq:NN \__unravel_nullfont: \tex_nullfont:D
15 \cs_new_eq:NN \__unravel_the:w \tex_the:D

```

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

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

```

\c_unravel_noprompt_ior
16 \cs_new_eq:NN \c_unravel_prompt_ior \c_sixteen
17 \cs_new_eq:NN \c_unravel_noprompt_ior \c_minus_one

```

(End definition for `\c_unravel_prompt_ior` and `\c_unravel_noprompt_ior`.)

2.1.2 Variants

Variants that we need.

```

18 \cs_if_exist:NF \exp_last_unbraced:NNn
19 { \cs_new_eq:NN \exp_last_unbraced:NNn \use:nnn }
20 \cs_generate_variant:Nn \exp_last_unbraced:NNn { NNv }
21 \cs_generate_variant:Nn \str_head:n { f }
22 \cs_generate_variant:Nn \tl_to_str:n { o }
23 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
24 \cs_generate_variant:Nn \tl_if_head_is_space:nTF { o }
25 \cs_generate_variant:Nn \tl_if_head_is_space:nT { V }
26 \cs_generate_variant:Nn \tl_if_head_is_N_type:nTF { o }
27 \cs_generate_variant:Nn \tl_if_in:nnF { nV }
28 \cs_generate_variant:Nn \tl_if_in:nnTF { nV }
29 \cs_generate_variant:Nn \tl_if_eq:nnT { V }
30 \cs_generate_variant:Nn \tl_if_in:NnTF { No , NV }
31 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
32 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nnx }
33 \cs_generate_variant:Nn \gtl_gput_left:Nn { Nx , NV , No }
34 \cs_generate_variant:Nn \gtl_gput_right:Nn { Nx , NV }
35 \cs_generate_variant:Nn \gtl_put_right:Nn { NV }
36 \cs_generate_variant:Nn \ior_get_str:NN { Nc }
37 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
38 \cs_generate_variant:Nn \gtl_to_str:N { c }

```

```

39 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
40 \cs_generate_variant:Nn \gtl_get_left:NN { c }
41 \cs_generate_variant:Nn \gtl_gset:Nn { c }
42 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
43 \cs_generate_variant:Nn \gtl_gclear:N { c }
44 \cs_generate_variant:Nn \gtl_gclear_new:N { c }

```

2.1.3 Miscellaneous helpers

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

```

45 \cs_new_protected_nopar:Npn \__unravel_tmp:w { }

```

(End definition for __unravel_tmp:w.)

```

\__unravel_tl_gset_input:Nnn
\__unravel_tl_gset_input:Nno
  \__unravel_tl_gset_input_aux:wN
46 \cs_new_protected:Npn \__unravel_tl_gset_input:Nnn #1#2#3
47 {
48   \group_begin:
49   \__unravel_everyeof:w \exp_after:wN { \token_to_str:N @ @ #1 }
50   #2
51   \tl_gclear:N #1
52   \exp_after:wN \__unravel_tl_gset_input_aux:wN
53   \exp_after:wN \prg_do_nothing:
54   \tex_input:D \tl_to_str:n {#3} \scan_stop:
55   \group_end:
56   \tl_gput_right:NV #1 \__unravel_everyeof:w
57 }
58 \cs_generate_variant:Nn \__unravel_tl_gset_input:Nnn { Nno }
59 \use:x
60 {
61   \cs_new_protected:Npn \exp_not:N \__unravel_tl_gset_input_aux:wN
62   ##1 \token_to_str:N @ @ ##2
63 } { \tl_gset:No #2 {#1} }

```

(End definition for __unravel_tl_gset_input:Nnn and __unravel_tl_gset_input:Nno.)

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 `-__int_value:w \fi: \c_zero`. If the escape character was a space, the test was true, and `__int_value:w` converts `\c_zero` 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`, thus does not remove anything from what follows.

```

64 \cs_new_nopar:Npn \__unravel_strip_escape:w

```

```

65 {
66   \tex_romannumeral:D
67   \if_charcode:w \token_to_str:N \ \_unravel_strip_escape_aux:w \fi:
68   \_unravel_strip_escape_aux:N
69 }
70 \cs_new:Npn \_unravel_strip_escape_aux:N #1 { \c_zero }
71 \cs_new:Npn \_unravel_strip_escape_aux:w #1#2
72 { - \_int_value:w #1 \c_zero }

```

(End definition for _unravel_strip_escape:w.)

_unravel_to_str:n Use the type-appropriate conversion to string. This unavoidably uses an internal function of gtl.

```

73 \cs_new:Npn \_unravel_to_str:n #1
74 {
75   \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
76   { \_unravel_to_str_auxi:w #1 ? \q_stop }
77   { \tl_to_str:n }
78   {#1}
79 }
80 \cs_set:Npn \_unravel_tmp:w #1
81 {
82   \cs_new:Npn \_unravel_to_str_auxi:w ##1##2 \q_stop
83   {
84     \exp_after:wN \_unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark
85     #1 \tl \q_mark \q_stop
86   }
87   \cs_new:Npn \_unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
88   { \cs_if_exist_use:cF { \_unravel_ ##2 _to_str:n } { \tl_to_str:n } }
89 }
90 \exp_args:No \_unravel_tmp:w { \tl_to_str:n { s_ } }
91 \cs_new:Npn \_unravel_gtl_to_str:n #1 { \_gtl_to_str:w #1 }

```

(End definition for _unravel_to_str:n.)

_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.

```

92 \cs_new:Npn \_unravel_str_truncate_left:nn #1#2
93 {
94   \exp_args:Nf \_unravel_str_truncate_left_aux:nnn
95   { \str_count:n {#1} } {#1} {#2}
96 }
97 \cs_new:Npn \_unravel_str_truncate_left_aux:nnn #1#2#3
98 {
99   \int_compare:nNnTF {#1} > {#3}
100   {
101     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
102     \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
103   }

```



```

104     { \tl_to_str:n {#2} }
105   }

```

(End definition for `__unravel_str_truncate_left:nn`.)

`__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.

```

106 \cs_new:Npn \__unravel_str_truncate_right:nn #1#2
107   {
108     \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
109     { \str_count:n {#1} } {#1} {#2}
110   }
111 \cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
112   {
113     \int_compare:nNnTF {#1} > {#3}
114     {
115       \str_range:nnn {#2} { 1 } { #3 - 25 } ~
116       ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
117     }
118     { \tl_to_str:n {#2} }
119   }

```

(End definition for `__unravel_str_truncate_right:nn`.)

2.1.5 Helpers for control flow

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

```

\__unravel_exit_point: 120 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
121 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }

```

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

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

```

\__unravel_break_point: 122 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
123 \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).

```

124 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
125   {
126     \int_compare:nNnTF
127     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
128     { \prg_return_false: }
129     {
130       \int_compare:nNnTF
131       \l__unravel_head_cmd_int
132       > { \__unravel_tex_use:n { max_internal } }

```

```

133         { \prg_return_false: }
134         { \prg_return_true: }
135     }
136 }

```

(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 137 \cs_new:Npn \_unravel_meaning_to_char:n #1
\__unravel_meaning_to_char_auxii:w 138 { \_unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
139 \cs_new:Npn \_unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
140 { \_unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
141 \cs_new:Npn \_unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
142 { \tl_if_empty:NTF {#2} { ~ } {#2} }
143 \cs_generate_variant:Nn \_unravel_meaning_to_char:n { o }
144 \cs_new:Npn \_unravel_token_to_char:N #1
145 { \_unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

(End definition for `_unravel_token_to_char:N`.)

`_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.

```

146 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
147 { p , T , F , TF }
148 {
149     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
150     \prg_return_false:
151     \else:
152     \prg_return_true:
153     \fi:
154 }

```

(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.

```

\__unravel_token_if_protected:N 155 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
\__unravel_token_if_protected:N 156 { p , T , F , TF }
157 {
158     \_unravel_token_if_expandable:N #1
159     {
160         \token_if_protected_macro:N #1
161         { \prg_return_true: }
162         {
163             \token_if_protected_long_macro:N #1
164             { \prg_return_true: }
165             { \prg_return_false: }

```

```

166         }
167     }
168     { \prg_return_true: }
169 }

```

(End definition for `_unravel_token_if_protected:NTF`.)

`_unravel_token_if_definable:NTF` Within a group, set the escape character to a non-space value (backslash). Convert the token to a string with `\token_to_str:N`. The result is multiple characters if the token is a control sequence, and a single character otherwise (even for explicit catcode 6 character tokens which would be doubled if we used `\tl_to_str:n` instead of `\token_to_str:N`). Thus `\str_tail:n` gives a non-empty result exactly for control sequences. Those are definable (technically, not always: `\expandafter\font\csname\endcsname=cmr10\expandafter\def\the\csname\endcsname{}`). For characters, there remains to determine if #1 is an active character. One option would be to build the active character with that character code and compare them using a delimited-argument test, but that needlessly pollutes the hash table in X_YTeX (and LuaTeX?) if the character was in fact not active. Instead, use the `\lowercase` primitive to convert the character to a fixed character code Z. Compare with an active Z. In all cases, remember to end the group.

```

170 \group_begin:
171   \char_set_catcode_active:n { 'Z }
172   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
173     { TF }
174     {
175       \group_begin:
176         \_unravel_set_escapechar:n { 92 }
177         \tl_set:Nx \l__unravel_tmpa_tl
178           { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
179         \tl_if_empty:NTF \l__unravel_tmpa_tl
180           {
181             \exp_args:Nx \char_set_lccode:nn
182               { ' \str_head:n {#1} } { 'Z }
183             \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
184             { \group_end: \prg_return_true: }
185             { \group_end: \prg_return_false: }
186           }
187         { \group_end: \prg_return_true: }
188       }
189   \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.

```

190 \prg_new_protected_conditional:Npnn \_unravel_gtl_if_head_is_definable:N #1
191   { TF , F }
192   {

```

```

193 \gtl_if_single_token:NTF #1
194 {
195     \gtl_if_head_is_N_type:NTF #1
196     {
197         \exp_last_unbraced:Nx \__unravel_token_if_definable:NTF
198         { \gtl_head:N #1 }
199         { \prg_return_true: }
200         { \prg_return_false: }
201     }
202     { \prg_return_false: }
203 }
204 { \prg_return_false: }
205 }

```

(End definition for __unravel_gtl_if_head_is_definable:NTF.)

2.1.7 Helpers for previous input

```

\__unravel_prev_input_silent:n
\__unravel_prev_input_silent:V
\__unravel_prev_input_silent:x
206 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
207 {
208     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_tl
209     \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
210     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_tl
211 }
212 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V , x }
213 \cs_new_protected:Npn \__unravel_prev_input:n #1
214 {
215     \__unravel_prev_input_silent:n {#1}
216     \__unravel_print_action:x { \tl_to_str:n {#1} }
217 }
218 \cs_generate_variant:Nn \__unravel_prev_input:n { V , x }

```

(End definition for __unravel_prev_input_silent:n, __unravel_prev_input_silent:V, and __unravel_prev_input_silent:x.)

__unravel_prev_input_gtl:N

```

219 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
220 {
221     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
222     \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
223     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
224 }

```

(End definition for __unravel_prev_input_gtl:N.)

__unravel_prev_input_join_get:nN Pops \g__unravel_prev_input_seq 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.

```

225 \cs_new_protected:Npn \__unravel_prev_input_join_get:nN #1

```

```

226 {
227   \int_case:nnF {#1}
228   {
229     { 2 } { \__unravel_join_get_aux:NNN \skip_eval:n \etex_glueexpr:D }
230     { 3 } { \__unravel_join_get_aux:NNN \muskip_eval:n \etex_muexpr:D }
231   }
232   {
233     \msg_error:nnn { unravel } { internal } { join-factor }
234     \__unravel_join_get_aux:NNN \use:n \prg_do_nothing:
235   }
236 }
237 \cs_new_protected:Npn \__unravel_join_get_aux:NNN #1#2#3
238 {
239   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
240   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
241   \tl_set:Nx #3 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl } }
242 }

```

(End definition for __unravel_prev_input_join_get:nN.)

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

```

243 \tl_new:N \g__unravel_before_print_state_tl
244 \tl_new:N \g__unravel_before_prompt_tl

```

(End definition for \g__unravel_before_print_state_tl and \g__unravel_before_prompt_tl. These variables are documented on page ??.)

\l__unravel_prompt_tmpa_int

```

245 \int_new:N \l__unravel_prompt_tmpa_int

```

(End definition for \l__unravel_prompt_tmpa_int. This variable is documented on page ??.)

\g__unravel_nonstop_int The number of prompts to skip.
\g__unravel_noise_int

```

246 \int_new:N \g__unravel_nonstop_int
247 \int_new:N \g__unravel_noise_int
248 \int_gset_eq:NN \g__unravel_noise_int \c_one

```

(End definition for \g__unravel_nonstop_int and \g__unravel_noise_int. These variables are documented on page ??.)

\g__unravel_default_explicit_prompt_bool Variables for the options explicit-prompt, internal-debug, number-steps. The default_ booleans store the default value of these options, and are affected by \unravelsetup or \unravel_setup:n.
\g__unravel_explicit_prompt_bool
\g__unravel_default_internal_debug_bool
\g__unravel_internal_debug_bool
\g__unravel_default_number_steps_bool
\g__unravel_number_steps_bool
\g__unravel_default_welcome_message_bool
\g__unravel_welcome_message_bool

```

249 \bool_new:N \g__unravel_default_explicit_prompt_bool
250 \bool_new:N \g__unravel_default_internal_debug_bool
251 \bool_new:N \g__unravel_default_number_steps_bool
252 \bool_gset_true:N \g__unravel_default_number_steps_bool

```

```

253 \bool_new:N \g__unravel_default_welcome_message_bool
254 \bool_gset_true:N \g__unravel_default_welcome_message_bool
255 \bool_new:N \g__unravel_explicit_prompt_bool
256 \bool_new:N \g__unravel_internal_debug_bool
257 \bool_new:N \g__unravel_number_steps_bool
258 \bool_new:N \g__unravel_welcome_message_bool

```

(End definition for \g__unravel_default_explicit_prompt_bool and others. These variables are documented on page ??.)

\g__unravel_step_int Current expansion step.

```

259 \int_new:N \g__unravel_step_int

```

(End definition for \g__unravel_step_int. This variable is documented on page ??.)

\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.

```

260 \str_new:N \g__unravel_action_text_str

```

(End definition for \g__unravel_action_text_str. This variable is documented on page ??.)

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

```

261 \int_new:N \g__unravel_default_max_action_int
262 \int_new:N \g__unravel_default_max_output_int
263 \int_new:N \g__unravel_default_max_input_int
264 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
265 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
266 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
267 \int_new:N \g__unravel_max_action_int
268 \int_new:N \g__unravel_max_output_int
269 \int_new:N \g__unravel_max_input_int

```

(End definition for \g__unravel_default_max_action_int and others. These variables are documented on page ??.)

\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.

```

270 \bool_new:N \g__unravel_speedup_macros_bool
271 \bool_gset_true:N \g__unravel_speedup_macros_bool

```

(End definition for \g__unravel_speedup_macros_bool. This variable is documented on page ??.)

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

```

272 \int_new:N \l__unravel_print_int

```

(End definition for \l__unravel_print_int. This variable is documented on page ??.)

2.2.2 Working with tokens

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

```
273 \int_new:N \g__unravel_input_int
```

(End definition for `\g__unravel_input_int`. This variable is documented on page ??.)

`\g__unravel_input_tmpa_int`
`\l__unravel_input_tmpa_tl`

```
274 \int_new:N \g__unravel_input_tmpa_int
```

```
275 \tl_new:N \l__unravel_input_tmpa_tl
```

(End definition for `\g__unravel_input_tmpa_int`. This variable is documented on page ??.)

`\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.

```
276 \seq_new:N \g__unravel_prev_input_seq
```

```
277 \tl_new:N \l__unravel_prev_input_tl
```

```
278 \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`. These variables are documented on page ??.)

`\g__unravel_output_gtl`

Material that is “typeset” or otherwise sent further down T_EX's digestion.

```
279 \gtl_new:N \g__unravel_output_gtl
```

(End definition for `\g__unravel_output_gtl`. This variable is documented on page ??.)

`\l__unravel_head_gtl`
`\l__unravel_head_tl`
`\l__unravel_head_token`
`\l__unravel_head_cmd_int`
`\l__unravel_head_char_int`

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

```
280 \gtl_new:N \l__unravel_head_gtl
```

```
281 \tl_new:N \l__unravel_head_tl
```

```
282 \token_new:Nn \l__unravel_head_token { ? }
```

```
283 \int_new:N \l__unravel_head_cmd_int
```

```
284 \int_new:N \l__unravel_head_char_int
```

(End definition for `\l__unravel_head_gtl` and others. These variables are documented on page ??.)

`\l__unravel_head_meaning_tl`

```
285 \tl_new:N \l__unravel_head_meaning_tl
```

(End definition for `\l__unravel_head_meaning_tl`. This variable is documented on page ??.)

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

`\l__unravel_tmpb_gtl`

`\g__unravel_tmpc_tl` 286 `\tl_new:N \l__unravel_tmpa_tl`

`\l__unravel_tmpa_seq` 287 `\gtl_new:N \l__unravel_unused_gtl`

`\l__unravel_unused_gtl` 288 `\gtl_new:N \l__unravel_tmpb_gtl`

289 `\tl_new:N \g__unravel_tmpc_tl`

290 `\seq_new:N \l__unravel_tmpa_seq`

(End definition for `\l__unravel_tmpa_tl` and others. These variables are documented on page ??.)

`\l__unravel_defined_tl` The token that is defined by the prefixed command (such as `\chardef` or `\futurelet`), and the code to define it. We do not use the `\g__unravel_prev_input_seq` to store that code: rather, this sequence contains a string representation of the code, which is not suitable for the definition. This is safe, as definitions cannot be nested. This is needed for expanding assignments, as expansion should be shown to the user, but then later should not be performed again when defining.

`\l__unravel_defining_tl`

291 `\tl_new:N \l__unravel_defined_tl`

292 `\tl_new:N \l__unravel_defining_tl`

(End definition for `\l__unravel_defined_tl` and `\l__unravel_defining_tl`. These variables are documented on page ??.)

`__unravel_inaccessible:w`

293 `\cs_new_eq:NN __unravel_inaccessible:w ?`

(End definition for `__unravel_inaccessible:w`.)

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

`\g__unravel_set_box_allowed_bool`

`\g__unravel_name_in_progress_bool`

294 `\gtl_new:N \g__unravel_after_assignment_gtl`

295 `\bool_new:N \g__unravel_set_box_allowed_bool`

296 `\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`. These variables are documented on page ??.)

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

297 `\group_begin:`

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

299 `\tl_const:Nx \c__unravel_parameters_tl`

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

301 `\group_end:`

(End definition for `\c__unravel_parameters_tl`. This variable is documented on page ??.)

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.

```
302 \int_new:N \g__unravel_val_level_int
```

(End definition for `\g__unravel_val_level_int`. This variable is documented on page ??.)

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

```
\g__unravel_if_limit_int 303 \tl_new:N \g__unravel_if_limit_tl
```

```
\g__unravel_if_depth_int 304 \int_new:N \g__unravel_if_limit_int
```

```
305 \int_new:N \g__unravel_if_depth_int
```

(End definition for `\g__unravel_if_limit_tl`. This variable is documented on page ??.)

`\l__unravel_if_nesting_int`

```
306 \int_new:N \l__unravel_if_nesting_int
```

(End definition for `\l__unravel_if_nesting_int`. This variable is documented on page ??.)

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.

```
307 \seq_new:N \l__unravel_leaders_box_seq
```

(End definition for `\l__unravel_leaders_box_seq`. This variable is documented on page ??.)

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

```
308 \int_new:N \g__unravel_ends_int
```

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

(End definition for `\g__unravel_ends_int`. This variable is documented on page ??.)

2.2.5 Constants

```

\c__unravel_plus_tl
\c__unravel_minus_tl 310 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 311 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 312 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 313 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 314 \tl_const:Nn \c__unravel_lq_tl { ' }
\c__unravel_dq_tl 315 \tl_const:Nn \c__unravel_rq_tl { ' }
\c__unravel_lp_tl 316 \tl_const:Nn \c__unravel_dq_tl { " }
\c__unravel_rp_tl 317 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 318 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 319 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 320 \tl_const:Nn \c__unravel_comma_tl { , }
321 \tl_const:Nn \c__unravel_point_tl { . }

```

(End definition for \c__unravel_plus_tl and others. These variables are documented on page ??.)

```

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

```

(End definition for \c__unravel_frozen_relax_gtl. This variable is documented on page ??.)

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 323 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
324 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
325 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }

```

(End definition for __unravel_tex_const:nn.)

```

\__unravel_tex_primitive:nnn
326 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
327 {
328   \tl_const:cx { c__unravel_tex_#1_tl }
329   { { \__unravel_tex_use:n {#2} } {#3} }
330 }

```

(End definition for __unravel_tex_primitive:nnn.)

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn 331 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
332 {
333   \cs_new_protected_nopar:cpn
334   { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}

```

```

335 }
336 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
337 {
338   \cs_new_eq:cc
339     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
340     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
341 }

```

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

__unravel_new_tex_expandable:nn

```

342 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
343 {
344   \cs_new_protected_nopar:cpn
345     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
346 }

```

(End definition for __unravel_new_tex_expandable:nn.)

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

```

347 \__unravel_tex_const:nn { relax } { 0 }
348 \__unravel_tex_const:nn { begin-group_char } { 1 }
349 \__unravel_tex_const:nn { end-group_char } { 2 }
350 \__unravel_tex_const:nn { math_char } { 3 }
351 \__unravel_tex_const:nn { tab_mark } { 4 }
352 \__unravel_tex_const:nn { alignment_char } { 4 }
353 \__unravel_tex_const:nn { car_ret } { 5 }
354 \__unravel_tex_const:nn { macro_char } { 6 }
355 \__unravel_tex_const:nn { superscript_char } { 7 }
356 \__unravel_tex_const:nn { subscript_char } { 8 }
357 \__unravel_tex_const:nn { endv } { 9 }
358 \__unravel_tex_const:nn { blank_char } { 10 }
359 \__unravel_tex_const:nn { the_char } { 11 }
360 \__unravel_tex_const:nn { other_char } { 12 }
361 \__unravel_tex_const:nn { par_end } { 13 }
362 \__unravel_tex_const:nn { stop } { 14 }
363 \__unravel_tex_const:nn { delim_num } { 15 }
364 \__unravel_tex_const:nn { max_char_code } { 15 }
365 \__unravel_tex_const:nn { char_num } { 16 }
366 \__unravel_tex_const:nn { math_char_num } { 17 }
367 \__unravel_tex_const:nn { mark } { 18 }
368 \__unravel_tex_const:nn { xray } { 19 }
369 \__unravel_tex_const:nn { make_box } { 20 }
370 \__unravel_tex_const:nn { hmove } { 21 }
371 \__unravel_tex_const:nn { vmove } { 22 }
372 \__unravel_tex_const:nn { un_hbox } { 23 }
373 \__unravel_tex_const:nn { un_vbox } { 24 }
374 \__unravel_tex_const:nn { remove_item } { 25 }
375 \__unravel_tex_const:nn { hskip } { 26 }
376 \__unravel_tex_const:nn { vskip } { 27 }
377 \__unravel_tex_const:nn { mskip } { 28 }

```

```

378 \__unravel_tex_const:nn { kern } { 29 }
379 \__unravel_tex_const:nn { mkern } { 30 }
380 \__unravel_tex_const:nn { leader_ship } { 31 }
381 \__unravel_tex_const:nn { halign } { 32 }
382 \__unravel_tex_const:nn { valign } { 33 }
383 \__unravel_tex_const:nn { no_align } { 34 }
384 \__unravel_tex_const:nn { vrule } { 35 }
385 \__unravel_tex_const:nn { hrule } { 36 }
386 \__unravel_tex_const:nn { insert } { 37 }
387 \__unravel_tex_const:nn { vadjust } { 38 }
388 \__unravel_tex_const:nn { ignore_spaces } { 39 }
389 \__unravel_tex_const:nn { after_assignment } { 40 }
390 \__unravel_tex_const:nn { after_group } { 41 }
391 \__unravel_tex_const:nn { break_penalty } { 42 }
392 \__unravel_tex_const:nn { start_par } { 43 }
393 \__unravel_tex_const:nn { ital_corr } { 44 }
394 \__unravel_tex_const:nn { accent } { 45 }
395 \__unravel_tex_const:nn { math_accent } { 46 }
396 \__unravel_tex_const:nn { discretionary } { 47 }
397 \__unravel_tex_const:nn { eq_no } { 48 }
398 \__unravel_tex_const:nn { left_right } { 49 }
399 \__unravel_tex_const:nn { math_comp } { 50 }
400 \__unravel_tex_const:nn { limit_switch } { 51 }
401 \__unravel_tex_const:nn { above } { 52 }
402 \__unravel_tex_const:nn { math_style } { 53 }
403 \__unravel_tex_const:nn { math_choice } { 54 }
404 \__unravel_tex_const:nn { non_script } { 55 }
405 \__unravel_tex_const:nn { vcenter } { 56 }
406 \__unravel_tex_const:nn { case_shift } { 57 }
407 \__unravel_tex_const:nn { message } { 58 }
408 \__unravel_tex_const:nn { extension } { 59 }
409 \__unravel_tex_const:nn { in_stream } { 60 }
410 \__unravel_tex_const:nn { begin_group } { 61 }
411 \__unravel_tex_const:nn { end_group } { 62 }
412 \__unravel_tex_const:nn { omit } { 63 }
413 \__unravel_tex_const:nn { ex_space } { 64 }
414 \__unravel_tex_const:nn { no_boundary } { 65 }
415 \__unravel_tex_const:nn { radical } { 66 }
416 \__unravel_tex_const:nn { end_cs_name } { 67 }
417 \__unravel_tex_const:nn { min_internal } { 68 }
418 \__unravel_tex_const:nn { char_given } { 68 }
419 \__unravel_tex_const:nn { math_given } { 69 }
420 \__unravel_tex_const:nn { last_item } { 70 }
421 \__unravel_tex_const:nn { max_non_prefixed_command } { 70 }
422 \__unravel_tex_const:nn { toks_register } { 71 }
423 \__unravel_tex_const:nn { assign_toks } { 72 }
424 \__unravel_tex_const:nn { assign_int } { 73 }
425 \__unravel_tex_const:nn { assign_dimen } { 74 }
426 \__unravel_tex_const:nn { assign_glue } { 75 }
427 \__unravel_tex_const:nn { assign_mu_glue } { 76 }

```

```

428 \__unravel_tex_const:nn { assign_font_dimen } { 77 }
429 \__unravel_tex_const:nn { assign_font_int } { 78 }
430 \__unravel_tex_const:nn { set_aux } { 79 }
431 \__unravel_tex_const:nn { set_prev_graf } { 80 }
432 \__unravel_tex_const:nn { set_page_dimen } { 81 }
433 \__unravel_tex_const:nn { set_page_int } { 82 }
434 \__unravel_tex_const:nn { set_box_dimen } { 83 }
435 \__unravel_tex_const:nn { set_shape } { 84 }
436 \__unravel_tex_const:nn { def_code } { 85 }
437 \__unravel_tex_const:nn { def_family } { 86 }
438 \__unravel_tex_const:nn { set_font } { 87 }
439 \__unravel_tex_const:nn { def_font } { 88 }
440 \__unravel_tex_const:nn { register } { 89 }
441 \__unravel_tex_const:nn { max_internal } { 89 }
442 \__unravel_tex_const:nn { advance } { 90 }
443 \__unravel_tex_const:nn { multiply } { 91 }
444 \__unravel_tex_const:nn { divide } { 92 }
445 \__unravel_tex_const:nn { prefix } { 93 }
446 \__unravel_tex_const:nn { let } { 94 }
447 \__unravel_tex_const:nn { shorthand_def } { 95 }
448 \__unravel_tex_const:nn { read_to_cs } { 96 }
449 \__unravel_tex_const:nn { def } { 97 }
450 \__unravel_tex_const:nn { set_box } { 98 }
451 \__unravel_tex_const:nn { hyph_data } { 99 }
452 \__unravel_tex_const:nn { set_interaction } { 100 }
453 \__unravel_tex_const:nn { letterspace_font } { 101 }
454 \__unravel_tex_const:nn { pdf_copy_font } { 102 }
455 \__unravel_tex_const:nn { max_command } { 102 }
456 \__unravel_tex_const:nn { undefined_cs } { 103 }
457 \__unravel_tex_const:nn { expand_after } { 104 }
458 \__unravel_tex_const:nn { no_expand } { 105 }
459 \__unravel_tex_const:nn { input } { 106 }
460 \__unravel_tex_const:nn { if_test } { 107 }
461 \__unravel_tex_const:nn { fi_or_else } { 108 }
462 \__unravel_tex_const:nn { cs_name } { 109 }
463 \__unravel_tex_const:nn { convert } { 110 }
464 \__unravel_tex_const:nn { the } { 111 }
465 \__unravel_tex_const:nn { top_bot_mark } { 112 }
466 \__unravel_tex_const:nn { call } { 113 }
467 \__unravel_tex_const:nn { end_template } { 117 }

```

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

A few minor differences with pdfTEX'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 ε -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`.

```

468 \__unravel_tex_primitive:nnn { relax          } { relax      } { 256 }
469 \__unravel_tex_primitive:nnn { span           } { tab_mark   } { 256 }
470 \__unravel_tex_primitive:nnn { cr             } { car_ret     } { 257 }
471 \__unravel_tex_primitive:nnn { crcr          } { car_ret     } { 258 }
472 \__unravel_tex_primitive:nnn { par            } { par_end     } { 256 }
473 \__unravel_tex_primitive:nnn { end            } { stop        } { 0  }
474 \__unravel_tex_primitive:nnn { dump           } { stop        } { 1  }
475 \__unravel_tex_primitive:nnn { delimiter      } { delim_num   } { 0  }
476 \__unravel_tex_primitive:nnn { char           } { char_num    } { 0  }
477 \__unravel_tex_primitive:nnn { mathchar       } { math_char_num } { 0  }
478 \__unravel_tex_primitive:nnn { mark           } { mark        } { 0  }
479 \__unravel_tex_primitive:nnn { marks          } { mark        } { 5  }
480 \__unravel_tex_primitive:nnn { show           } { xray        } { 0  }
481 \__unravel_tex_primitive:nnn { showbox        } { xray        } { 1  }
482 \__unravel_tex_primitive:nnn { showthe        } { xray        } { 2  }
483 \__unravel_tex_primitive:nnn { showlists      } { xray        } { 3  }
484 \__unravel_tex_primitive:nnn { showgroups     } { xray        } { 4  }
485 \__unravel_tex_primitive:nnn { showtokens     } { xray        } { 5  }
486 \__unravel_tex_primitive:nnn { showifs        } { xray        } { 6  }
487 \__unravel_tex_primitive:nnn { box            } { make_box    } { 0  }
488 \__unravel_tex_primitive:nnn { copy           } { make_box    } { 1  }
489 \__unravel_tex_primitive:nnn { lastbox        } { make_box    } { 2  }
490 \__unravel_tex_primitive:nnn { vsplit        } { make_box    } { 3  }
491 \__unravel_tex_primitive:nnn { vtop           } { make_box    } { 4  }
492 \__unravel_tex_primitive:nnn { vbox          } { make_box    } { 5  }
493 \__unravel_tex_primitive:nnn { hbox          } { make_box    } { 106 }
494 \__unravel_tex_primitive:nnn { moveright      } { hmove       } { 0  }
495 \__unravel_tex_primitive:nnn { moveleft      } { hmove       } { 1  }
496 \__unravel_tex_primitive:nnn { lower         } { vmove       } { 0  }
497 \__unravel_tex_primitive:nnn { raise         } { vmove       } { 1  }
498 \__unravel_tex_primitive:nnn { unhbox        } { un_hbox     } { 0  }
499 \__unravel_tex_primitive:nnn { unhcopy       } { un_hbox     } { 1  }
500 \__unravel_tex_primitive:nnn { unvbox        } { un_vbox     } { 0  }
501 \__unravel_tex_primitive:nnn { unvcopy       } { un_vbox     } { 1  }
502 \__unravel_tex_primitive:nnn { pagediscards   } { un_vbox     } { 2  }
503 \__unravel_tex_primitive:nnn { splitdiscards } { un_vbox     } { 3  }
504 \__unravel_tex_primitive:nnn { unpenalty     } { remove_item } { 12 }
505 \__unravel_tex_primitive:nnn { unkern        } { remove_item } { 11 }
506 \__unravel_tex_primitive:nnn { unskip       } { remove_item } { 10 }
507 \__unravel_tex_primitive:nnn { hfil         } { hskip       } { 0  }
508 \__unravel_tex_primitive:nnn { hfill        } { hskip       } { 1  }
509 \__unravel_tex_primitive:nnn { hss          } { hskip       } { 2  }
510 \__unravel_tex_primitive:nnn { hfilneg      } { hskip       } { 3  }

```

```

511 \__unravel_tex_primitive:nnn { hskip } { hskip } { 4 }
512 \__unravel_tex_primitive:nnn { vfil } { vskip } { 0 }
513 \__unravel_tex_primitive:nnn { vfill } { vskip } { 1 }
514 \__unravel_tex_primitive:nnn { vss } { vskip } { 2 }
515 \__unravel_tex_primitive:nnn { vfilneg } { vskip } { 3 }
516 \__unravel_tex_primitive:nnn { vskip } { vskip } { 4 }
517 \__unravel_tex_primitive:nnn { mskip } { mskip } { 5 }
518 \__unravel_tex_primitive:nnn { kern } { kern } { 1 }
519 \__unravel_tex_primitive:nnn { mkern } { mkern } { 99 }
520 \__unravel_tex_primitive:nnn { shipout } { leader_ship } { 99 }
521 \__unravel_tex_primitive:nnn { leaders } { leader_ship } { 100 }
522 \__unravel_tex_primitive:nnn { cleaders } { leader_ship } { 101 }
523 \__unravel_tex_primitive:nnn { xleaders } { leader_ship } { 102 }
524 \__unravel_tex_primitive:nnn { halign } { halign } { 0 }
525 \__unravel_tex_primitive:nnn { valign } { valign } { 0 }
526 \__unravel_tex_primitive:nnn { beginL } { valign } { 4 }
527 \__unravel_tex_primitive:nnn { endL } { valign } { 5 }
528 \__unravel_tex_primitive:nnn { beginR } { valign } { 8 }
529 \__unravel_tex_primitive:nnn { endR } { valign } { 9 }
530 \__unravel_tex_primitive:nnn { noalign } { no_align } { 0 }
531 \__unravel_tex_primitive:nnn { vrule } { vrule } { 0 }
532 \__unravel_tex_primitive:nnn { hrule } { hrule } { 0 }
533 \__unravel_tex_primitive:nnn { insert } { insert } { 0 }
534 \__unravel_tex_primitive:nnn { vadjust } { vadjust } { 0 }
535 \__unravel_tex_primitive:nnn { ignorespaces } { ignore_spaces } { 0 }
536 \__unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
537 \__unravel_tex_primitive:nnn { aftergroup } { after_group } { 0 }
538 \__unravel_tex_primitive:nnn { penalty } { break_penalty } { 0 }
539 \__unravel_tex_primitive:nnn { indent } { start_par } { 1 }
540 \__unravel_tex_primitive:nnn { noindent } { start_par } { 0 }
541 \__unravel_tex_primitive:nnn { quitvmode } { start_par } { 2 }
542 \__unravel_tex_primitive:nnn { / } { ital_corr } { 0 }
543 \__unravel_tex_primitive:nnn { accent } { accent } { 0 }
544 \__unravel_tex_primitive:nnn { mathaccent } { math_accent } { 0 }
545 \__unravel_tex_primitive:nnn { - } { discretionary } { 1 }
546 \__unravel_tex_primitive:nnn { discretionary } { discretionary } { 0 }
547 \__unravel_tex_primitive:nnn { eqno } { eq_no } { 0 }
548 \__unravel_tex_primitive:nnn { leqno } { eq_no } { 1 }
549 \__unravel_tex_primitive:nnn { left } { left_right } { 30 }
550 \__unravel_tex_primitive:nnn { right } { left_right } { 31 }
551 \__unravel_tex_primitive:nnn { middle } { left_right } { 17 }
552 \__unravel_tex_primitive:nnn { mathord } { math_comp } { 16 }
553 \__unravel_tex_primitive:nnn { mathop } { math_comp } { 17 }
554 \__unravel_tex_primitive:nnn { mathbin } { math_comp } { 18 }
555 \__unravel_tex_primitive:nnn { mathrel } { math_comp } { 19 }
556 \__unravel_tex_primitive:nnn { mathopen } { math_comp } { 20 }
557 \__unravel_tex_primitive:nnn { mathclose } { math_comp } { 21 }
558 \__unravel_tex_primitive:nnn { mathpunct } { math_comp } { 22 }
559 \__unravel_tex_primitive:nnn { mathinner } { math_comp } { 23 }
560 \__unravel_tex_primitive:nnn { underline } { math_comp } { 26 }

```

```

561 \__unravel_tex_primitive:nnn { overline          } { math_comp } { 27 }
562 \__unravel_tex_primitive:nnn { displaylimits     } { limit_switch } { 0 }
563 \__unravel_tex_primitive:nnn { limits            } { limit_switch } { 1 }
564 \__unravel_tex_primitive:nnn { nolimits          } { limit_switch } { 2 }
565 \__unravel_tex_primitive:nnn { above             } { above } { 0 }
566 \__unravel_tex_primitive:nnn { over              } { above } { 1 }
567 \__unravel_tex_primitive:nnn { atop              } { above } { 2 }
568 \__unravel_tex_primitive:nnn { abovewithdelims    } { above } { 3 }
569 \__unravel_tex_primitive:nnn { overwithdelims     } { above } { 4 }
570 \__unravel_tex_primitive:nnn { atopwithdelims     } { above } { 5 }
571 \__unravel_tex_primitive:nnn { displaystyle       } { math_style } { 0 }
572 \__unravel_tex_primitive:nnn { textstyle         } { math_style } { 2 }
573 \__unravel_tex_primitive:nnn { scriptstyle       } { math_style } { 4 }
574 \__unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
575 \__unravel_tex_primitive:nnn { mathchoice        } { math_choice } { 0 }
576 \__unravel_tex_primitive:nnn { nonscript         } { non_script } { 0 }
577 \__unravel_tex_primitive:nnn { vcenter           } { vcenter } { 0 }
578 \__unravel_tex_primitive:nnn { lowercase         } { case_shift } { 256 }
579 \__unravel_tex_primitive:nnn { uppercase         } { case_shift } { 512 }
580 \__unravel_tex_primitive:nnn { message           } { message } { 0 }
581 \__unravel_tex_primitive:nnn { errmessage        } { message } { 1 }
582 \__unravel_tex_primitive:nnn { openout           } { extension } { 0 }
583 \__unravel_tex_primitive:nnn { write             } { extension } { 1 }
584 \__unravel_tex_primitive:nnn { closeout          } { extension } { 2 }
585 \__unravel_tex_primitive:nnn { special           } { extension } { 3 }
586 \__unravel_tex_primitive:nnn { immediate         } { extension } { 4 }
587 \__unravel_tex_primitive:nnn { setlanguage       } { extension } { 5 }
588 \__unravel_tex_primitive:nnn { pdfliteral        } { extension } { 6 }
589 \__unravel_tex_primitive:nnn { pdfobj            } { extension } { 7 }
590 \__unravel_tex_primitive:nnn { pdfrefobj         } { extension } { 8 }
591 \__unravel_tex_primitive:nnn { pdfxform          } { extension } { 9 }
592 \__unravel_tex_primitive:nnn { pdfrefxform       } { extension } { 10 }
593 \__unravel_tex_primitive:nnn { pdfximage         } { extension } { 11 }
594 \__unravel_tex_primitive:nnn { pdfrefximage      } { extension } { 12 }
595 \__unravel_tex_primitive:nnn { pdfannot          } { extension } { 13 }
596 \__unravel_tex_primitive:nnn { pdfstartlink      } { extension } { 14 }
597 \__unravel_tex_primitive:nnn { pdfendlink        } { extension } { 15 }
598 \__unravel_tex_primitive:nnn { pdfoutline        } { extension } { 16 }
599 \__unravel_tex_primitive:nnn { pdfdest           } { extension } { 17 }
600 \__unravel_tex_primitive:nnn { pdfthread         } { extension } { 18 }
601 \__unravel_tex_primitive:nnn { pdfstartthread    } { extension } { 19 }
602 \__unravel_tex_primitive:nnn { pdfendthread      } { extension } { 20 }
603 \__unravel_tex_primitive:nnn { pdfsavepos        } { extension } { 21 }
604 \__unravel_tex_primitive:nnn { pdfinfo           } { extension } { 22 }
605 \__unravel_tex_primitive:nnn { pdfcatalog        } { extension } { 23 }
606 \__unravel_tex_primitive:nnn { pdfnames          } { extension } { 24 }
607 \__unravel_tex_primitive:nnn { pdffontattr      } { extension } { 25 }
608 \__unravel_tex_primitive:nnn { pdfincludechars   } { extension } { 26 }
609 \__unravel_tex_primitive:nnn { pdfmapfile        } { extension } { 27 }
610 \__unravel_tex_primitive:nnn { pdfmapline       } { extension } { 28 }

```



```

611 \__unravel_tex_primitive:nnn { pdftrailer      } { extension } { 29 }
612 \__unravel_tex_primitive:nnn { pdfresettimer   } { extension } { 30 }
613 \__unravel_tex_primitive:nnn { pdffontexpand  } { extension } { 31 }
614 \__unravel_tex_primitive:nnn { pdfsetrandomseed } { extension } { 32 }
615 \__unravel_tex_primitive:nnn { pdfsnaprefpoint } { extension } { 33 }
616 \__unravel_tex_primitive:nnn { pdfsnapy      } { extension } { 34 }
617 \__unravel_tex_primitive:nnn { pdfsnapycomp   } { extension } { 35 }
618 \__unravel_tex_primitive:nnn { pdfglyphtounicode } { extension } { 36 }
619 \__unravel_tex_primitive:nnn { pdfcolorstack  } { extension } { 37 }
620 \__unravel_tex_primitive:nnn { pdfsetmatrix    } { extension } { 38 }
621 \__unravel_tex_primitive:nnn { pdfsave        } { extension } { 39 }
622 \__unravel_tex_primitive:nnn { pdfrestore      } { extension } { 40 }
623 \__unravel_tex_primitive:nnn { pdfnobluiltintounicode } { extension } { 41 }
624 \__unravel_tex_primitive:nnn { openin          } { in_stream } { 1 }
625 \__unravel_tex_primitive:nnn { closein         } { in_stream } { 0 }
626 \__unravel_tex_primitive:nnn { begingroup      } { begin_group } { 0 }
627 \__unravel_tex_primitive:nnn { endgroup        } { end_group } { 0 }
628 \__unravel_tex_primitive:nnn { omit            } { omit } { 0 }
629 \__unravel_tex_primitive:nnn { ~              } { ex_space } { 0 }
630 \__unravel_tex_primitive:nnn { noboundary      } { no_boundary } { 0 }
631 \__unravel_tex_primitive:nnn { radical         } { radical } { 0 }
632 \__unravel_tex_primitive:nnn { endcsname       } { end_cs_name } { 0 }
633 \__unravel_tex_primitive:nnn { lastpenalty     } { last_item } { 0 }
634 \__unravel_tex_primitive:nnn { lastkern        } { last_item } { 1 }
635 \__unravel_tex_primitive:nnn { lastskip        } { last_item } { 2 }
636 \__unravel_tex_primitive:nnn { lastnodetype     } { last_item } { 3 }
637 \__unravel_tex_primitive:nnn { inputlineno     } { last_item } { 4 }
638 \__unravel_tex_primitive:nnn { badness         } { last_item } { 5 }
639 \__unravel_tex_primitive:nnn { pdftexversion   } { last_item } { 6 }
640 \__unravel_tex_primitive:nnn { pdflastobj       } { last_item } { 7 }
641 \__unravel_tex_primitive:nnn { pdflastxform     } { last_item } { 8 }
642 \__unravel_tex_primitive:nnn { pdflastximage    } { last_item } { 9 }
643 \__unravel_tex_primitive:nnn { pdflastximagepages } { last_item } { 10 }
644 \__unravel_tex_primitive:nnn { pdflastannot     } { last_item } { 11 }
645 \__unravel_tex_primitive:nnn { pdflastxpos      } { last_item } { 12 }
646 \__unravel_tex_primitive:nnn { pdflastypos      } { last_item } { 13 }
647 \__unravel_tex_primitive:nnn { pdfretval        } { last_item } { 14 }
648 \__unravel_tex_primitive:nnn { pdflastximagecolordepth } { last_item } { 15 }
649 \__unravel_tex_primitive:nnn { pdfelapsedtime   } { last_item } { 16 }
650 \__unravel_tex_primitive:nnn { pdfshellescape   } { last_item } { 17 }
651 \__unravel_tex_primitive:nnn { pdfrandomseed    } { last_item } { 18 }
652 \__unravel_tex_primitive:nnn { pdflastlink      } { last_item } { 19 }
653 \__unravel_tex_primitive:nnn { eTeXversion      } { last_item } { 20 }
654 \__unravel_tex_primitive:nnn { currentgrouplevel } { last_item } { 21 }
655 \__unravel_tex_primitive:nnn { currentgrouptype } { last_item } { 22 }
656 \__unravel_tex_primitive:nnn { currentiflevel   } { last_item } { 23 }
657 \__unravel_tex_primitive:nnn { currentiftypel   } { last_item } { 24 }
658 \__unravel_tex_primitive:nnn { currentifbranch  } { last_item } { 25 }
659 \__unravel_tex_primitive:nnn { gluestretchorder } { last_item } { 26 }
660 \__unravel_tex_primitive:nnn { glueshrinkorder  } { last_item } { 27 }

```

```

661 \__unravel_tex_primitive:nnn { fontcharwd } { last_item } { 28 }
662 \__unravel_tex_primitive:nnn { fontcharht } { last_item } { 29 }
663 \__unravel_tex_primitive:nnn { fontchardp } { last_item } { 30 }
664 \__unravel_tex_primitive:nnn { fontcharic } { last_item } { 31 }
665 \__unravel_tex_primitive:nnn { parshapelength } { last_item } { 32 }
666 \__unravel_tex_primitive:nnn { parshapeindent } { last_item } { 33 }
667 \__unravel_tex_primitive:nnn { parshapedimen } { last_item } { 34 }
668 \__unravel_tex_primitive:nnn { gluestretch } { last_item } { 35 }
669 \__unravel_tex_primitive:nnn { glueshrink } { last_item } { 36 }
670 \__unravel_tex_primitive:nnn { mutoglu } { last_item } { 37 }
671 \__unravel_tex_primitive:nnn { glueto } { last_item } { 38 }
672 \__unravel_tex_primitive:nnn { numexpr } { last_item } { 39 }
673 \__unravel_tex_primitive:nnn { dimexpr } { last_item } { 40 }
674 \__unravel_tex_primitive:nnn { glueexpr } { last_item } { 41 }
675 \__unravel_tex_primitive:nnn { muexpr } { last_item } { 42 }
676 \__unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
677 \__unravel_tex_primitive:nnn { output } { assign_toks } { 1 }
678 \__unravel_tex_primitive:nnn { everypar } { assign_toks } { 2 }
679 \__unravel_tex_primitive:nnn { everymath } { assign_toks } { 3 }
680 \__unravel_tex_primitive:nnn { everydisplay } { assign_toks } { 4 }
681 \__unravel_tex_primitive:nnn { everyhbox } { assign_toks } { 5 }
682 \__unravel_tex_primitive:nnn { everyvbox } { assign_toks } { 6 }
683 \__unravel_tex_primitive:nnn { everyjob } { assign_toks } { 7 }
684 \__unravel_tex_primitive:nnn { everycr } { assign_toks } { 8 }
685 \__unravel_tex_primitive:nnn { errhelp } { assign_toks } { 9 }
686 \__unravel_tex_primitive:nnn { pdfpagesattr } { assign_toks } { 10 }
687 \__unravel_tex_primitive:nnn { pdfpageattr } { assign_toks } { 11 }
688 \__unravel_tex_primitive:nnn { pdfpageresources } { assign_toks } { 12 }
689 \__unravel_tex_primitive:nnn { pdfpkmode } { assign_toks } { 13 }
690 \__unravel_tex_primitive:nnn { everyeof } { assign_toks } { 14 }
691 \__unravel_tex_primitive:nnn { pretolerance } { assign_int } { 0 }
692 \__unravel_tex_primitive:nnn { tolerance } { assign_int } { 1 }
693 \__unravel_tex_primitive:nnn { linepenalty } { assign_int } { 2 }
694 \__unravel_tex_primitive:nnn { hyphenpenalty } { assign_int } { 3 }
695 \__unravel_tex_primitive:nnn { exhyphenpenalty } { assign_int } { 4 }
696 \__unravel_tex_primitive:nnn { clubpenalty } { assign_int } { 5 }
697 \__unravel_tex_primitive:nnn { widowpenalty } { assign_int } { 6 }
698 \__unravel_tex_primitive:nnn { displaywidowpenalty } { assign_int } { 7 }
699 \__unravel_tex_primitive:nnn { brokenpenalty } { assign_int } { 8 }
700 \__unravel_tex_primitive:nnn { binoppenalty } { assign_int } { 9 }
701 \__unravel_tex_primitive:nnn { relpenalty } { assign_int } { 10 }
702 \__unravel_tex_primitive:nnn { predisplayspace } { assign_int } { 11 }
703 \__unravel_tex_primitive:nnn { postdisplayspace } { assign_int } { 12 }
704 \__unravel_tex_primitive:nnn { interlinepenalty } { assign_int } { 13 }
705 \__unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int } { 14 }
706 \__unravel_tex_primitive:nnn { finalhyphendemerits } { assign_int } { 15 }
707 \__unravel_tex_primitive:nnn { adjdemerits } { assign_int } { 16 }
708 \__unravel_tex_primitive:nnn { mag } { assign_int } { 17 }
709 \__unravel_tex_primitive:nnn { delimiterfactor } { assign_int } { 18 }
710 \__unravel_tex_primitive:nnn { looseness } { assign_int } { 19 }

```

```

711 \__unravel_tex_primitive:nnn { time } { assign_int } { 20 }
712 \__unravel_tex_primitive:nnn { day } { assign_int } { 21 }
713 \__unravel_tex_primitive:nnn { month } { assign_int } { 22 }
714 \__unravel_tex_primitive:nnn { year } { assign_int } { 23 }
715 \__unravel_tex_primitive:nnn { showboxbreadth } { assign_int } { 24 }
716 \__unravel_tex_primitive:nnn { showboxdepth } { assign_int } { 25 }
717 \__unravel_tex_primitive:nnn { hbadness } { assign_int } { 26 }
718 \__unravel_tex_primitive:nnn { vbadness } { assign_int } { 27 }
719 \__unravel_tex_primitive:nnn { pausing } { assign_int } { 28 }
720 \__unravel_tex_primitive:nnn { tracingonline } { assign_int } { 29 }
721 \__unravel_tex_primitive:nnn { tracingmacros } { assign_int } { 30 }
722 \__unravel_tex_primitive:nnn { tracingstats } { assign_int } { 31 }
723 \__unravel_tex_primitive:nnn { tracingparagraphs } { assign_int } { 32 }
724 \__unravel_tex_primitive:nnn { tracingpages } { assign_int } { 33 }
725 \__unravel_tex_primitive:nnn { tracingoutput } { assign_int } { 34 }
726 \__unravel_tex_primitive:nnn { tracinglostchars } { assign_int } { 35 }
727 \__unravel_tex_primitive:nnn { tracingcommands } { assign_int } { 36 }
728 \__unravel_tex_primitive:nnn { tracingrestores } { assign_int } { 37 }
729 \__unravel_tex_primitive:nnn { uchyph } { assign_int } { 38 }
730 \__unravel_tex_primitive:nnn { outputpenalty } { assign_int } { 39 }
731 \__unravel_tex_primitive:nnn { maxdeadcycles } { assign_int } { 40 }
732 \__unravel_tex_primitive:nnn { hangafter } { assign_int } { 41 }
733 \__unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
734 \__unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
735 \__unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
736 \__unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
737 \__unravel_tex_primitive:nnn { defaultthyphenchar } { assign_int } { 46 }
738 \__unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
739 \__unravel_tex_primitive:nnn { endlinechar } { assign_int } { 48 }
740 \__unravel_tex_primitive:nnn { newlinechar } { assign_int } { 49 }
741 \__unravel_tex_primitive:nnn { language } { assign_int } { 50 }
742 \__unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
743 \__unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
744 \__unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
745 \__unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
746 \__unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
747 \__unravel_tex_primitive:nnn { pdfcompresslevel } { assign_int } { 56 }
748 \__unravel_tex_primitive:nnn { pdfdecimaldigits } { assign_int } { 57 }
749 \__unravel_tex_primitive:nnn { pdfmovechars } { assign_int } { 58 }
750 \__unravel_tex_primitive:nnn { pdfimageresolution } { assign_int } { 59 }
751 \__unravel_tex_primitive:nnn { pdfpkresolution } { assign_int } { 60 }
752 \__unravel_tex_primitive:nnn { pdfuniqueresname } { assign_int } { 61 }
753 \__unravel_tex_primitive:nnn
754 { pdfoptionalwaysusepdfpagebox } { assign_int } { 62 }
755 \__unravel_tex_primitive:nnn
756 { pdfoptionpdfinclusionerrorlevel } { assign_int } { 63 }
757 \__unravel_tex_primitive:nnn
758 { pdfoptionpdfminorversion } { assign_int } { 64 }
759 \__unravel_tex_primitive:nnn { pdfminorversion } { assign_int } { 64 }
760 \__unravel_tex_primitive:nnn { pdfforcepagebox } { assign_int } { 65 }

```

```

761 \_ unravel\_tex\_primitive:nnn { pdfpagebox } { assign\_int } { 66 }
762 \_ unravel\_tex\_primitive:nnn
763 { pdfinclusionerrorlevel } { assign\_int } { 67 }
764 \_ unravel\_tex\_primitive:nnn { pdfgamma } { assign\_int } { 68 }
765 \_ unravel\_tex\_primitive:nnn { pdfimagegamma } { assign\_int } { 69 }
766 \_ unravel\_tex\_primitive:nnn { pdfimagehicolor } { assign\_int } { 70 }
767 \_ unravel\_tex\_primitive:nnn { pdfimageapplygamma } { assign\_int } { 71 }
768 \_ unravel\_tex\_primitive:nnn { pdfadjustspacing } { assign\_int } { 72 }
769 \_ unravel\_tex\_primitive:nnn { pdfprotrudechars } { assign\_int } { 73 }
770 \_ unravel\_tex\_primitive:nnn { pdftracingfonts } { assign\_int } { 74 }
771 \_ unravel\_tex\_primitive:nnn { pdfobjcompresslevel } { assign\_int } { 75 }
772 \_ unravel\_tex\_primitive:nnn
773 { pdfadjustinterwordglue } { assign\_int } { 76 }
774 \_ unravel\_tex\_primitive:nnn { pdfprependkern } { assign\_int } { 77 }
775 \_ unravel\_tex\_primitive:nnn { pdfappendkern } { assign\_int } { 78 }
776 \_ unravel\_tex\_primitive:nnn { pdfgentounicode } { assign\_int } { 79 }
777 \_ unravel\_tex\_primitive:nnn { pdfdraftmode } { assign\_int } { 80 }
778 \_ unravel\_tex\_primitive:nnn { pdfinclusioncopyfonts } { assign\_int } { 81 }
779 \_ unravel\_tex\_primitive:nnn { tracingassigns } { assign\_int } { 82 }
780 \_ unravel\_tex\_primitive:nnn { tracinggroups } { assign\_int } { 83 }
781 \_ unravel\_tex\_primitive:nnn { tracingifs } { assign\_int } { 84 }
782 \_ unravel\_tex\_primitive:nnn { tracingscantokens } { assign\_int } { 85 }
783 \_ unravel\_tex\_primitive:nnn { tracingnesting } { assign\_int } { 86 }
784 \_ unravel\_tex\_primitive:nnn { predisdisplaydirection } { assign\_int } { 87 }
785 \_ unravel\_tex\_primitive:nnn { lastlinefit } { assign\_int } { 88 }
786 \_ unravel\_tex\_primitive:nnn { savingvdiscards } { assign\_int } { 89 }
787 \_ unravel\_tex\_primitive:nnn { savinghyphcodes } { assign\_int } { 90 }
788 \_ unravel\_tex\_primitive:nnn { TeXxETstate } { assign\_int } { 91 }
789 \_ unravel\_tex\_primitive:nnn { parindent } { assign\_dimen } { 0 }
790 \_ unravel\_tex\_primitive:nnn { mathsurround } { assign\_dimen } { 1 }
791 \_ unravel\_tex\_primitive:nnn { lineskiplimit } { assign\_dimen } { 2 }
792 \_ unravel\_tex\_primitive:nnn { hsize } { assign\_dimen } { 3 }
793 \_ unravel\_tex\_primitive:nnn { vsize } { assign\_dimen } { 4 }
794 \_ unravel\_tex\_primitive:nnn { maxdepth } { assign\_dimen } { 5 }
795 \_ unravel\_tex\_primitive:nnn { splitmaxdepth } { assign\_dimen } { 6 }
796 \_ unravel\_tex\_primitive:nnn { boxmaxdepth } { assign\_dimen } { 7 }
797 \_ unravel\_tex\_primitive:nnn { hfuzz } { assign\_dimen } { 8 }
798 \_ unravel\_tex\_primitive:nnn { vfuzz } { assign\_dimen } { 9 }
799 \_ unravel\_tex\_primitive:nnn { delimitershortfall } { assign\_dimen } { 10 }
800 \_ unravel\_tex\_primitive:nnn { nulldelimiterspace } { assign\_dimen } { 11 }
801 \_ unravel\_tex\_primitive:nnn { scriptspace } { assign\_dimen } { 12 }
802 \_ unravel\_tex\_primitive:nnn { predisplaysize } { assign\_dimen } { 13 }
803 \_ unravel\_tex\_primitive:nnn { displaywidth } { assign\_dimen } { 14 }
804 \_ unravel\_tex\_primitive:nnn { displayindent } { assign\_dimen } { 15 }
805 \_ unravel\_tex\_primitive:nnn { overfullrule } { assign\_dimen } { 16 }
806 \_ unravel\_tex\_primitive:nnn { hangindent } { assign\_dimen } { 17 }
807 \_ unravel\_tex\_primitive:nnn { hoffset } { assign\_dimen } { 18 }
808 \_ unravel\_tex\_primitive:nnn { voffset } { assign\_dimen } { 19 }
809 \_ unravel\_tex\_primitive:nnn { emergencystretch } { assign\_dimen } { 20 }
810 \_ unravel\_tex\_primitive:nnn { pdfhorigin } { assign\_dimen } { 21 }

```

```

811 \_unravel_tex_primitive:nnn { pdfvorigin          } { assign_dimen } { 22 }
812 \_unravel_tex_primitive:nnn { pdfpagewidth       } { assign_dimen } { 23 }
813 \_unravel_tex_primitive:nnn { pdfpageheight     } { assign_dimen } { 24 }
814 \_unravel_tex_primitive:nnn { pdflinkmargin      } { assign_dimen } { 25 }
815 \_unravel_tex_primitive:nnn { pdfdestmargin     } { assign_dimen } { 26 }
816 \_unravel_tex_primitive:nnn { pdfthreadmargin   } { assign_dimen } { 27 }
817 \_unravel_tex_primitive:nnn { pdffirstlineheight } { assign_dimen } { 28 }
818 \_unravel_tex_primitive:nnn { pdflastlinedepth  } { assign_dimen } { 29 }
819 \_unravel_tex_primitive:nnn { pdfeachlineheight } { assign_dimen } { 30 }
820 \_unravel_tex_primitive:nnn { pdfeachlinedepth  } { assign_dimen } { 31 }
821 \_unravel_tex_primitive:nnn { pdfignoreddimen  } { assign_dimen } { 32 }
822 \_unravel_tex_primitive:nnn { pdfpxdimen       } { assign_dimen } { 33 }
823 \_unravel_tex_primitive:nnn { lineskip         } { assign_glue } { 0 }
824 \_unravel_tex_primitive:nnn { baselineskip      } { assign_glue } { 1 }
825 \_unravel_tex_primitive:nnn { parskip          } { assign_glue } { 2 }
826 \_unravel_tex_primitive:nnn { abovedisplayskip  } { assign_glue } { 3 }
827 \_unravel_tex_primitive:nnn { belowdisplayskip  } { assign_glue } { 4 }
828 \_unravel_tex_primitive:nnn { abovedisplayskip  } { assign_glue } { 5 }
829 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
830 \_unravel_tex_primitive:nnn { leftskip        } { assign_glue } { 7 }
831 \_unravel_tex_primitive:nnn { rightskip       } { assign_glue } { 8 }
832 \_unravel_tex_primitive:nnn { topskip         } { assign_glue } { 9 }
833 \_unravel_tex_primitive:nnn { splittopskip    } { assign_glue } { 10 }
834 \_unravel_tex_primitive:nnn { tabskip         } { assign_glue } { 11 }
835 \_unravel_tex_primitive:nnn { spaceskip       } { assign_glue } { 12 }
836 \_unravel_tex_primitive:nnn { xspaceskip      } { assign_glue } { 13 }
837 \_unravel_tex_primitive:nnn { parfillskip     } { assign_glue } { 14 }
838 \_unravel_tex_primitive:nnn { thinmuskip      } { assign_mu_glue } { 15 }
839 \_unravel_tex_primitive:nnn { medmuskip       } { assign_mu_glue } { 16 }
840 \_unravel_tex_primitive:nnn { thickmuskip     } { assign_mu_glue } { 17 }
841 \_unravel_tex_primitive:nnn { fontdimen       } { assign_font_dimen } { 0 }
842 \_unravel_tex_primitive:nnn { hyphenchar      } { assign_font_int } { 0 }
843 \_unravel_tex_primitive:nnn { skewchar        } { assign_font_int } { 1 }
844 \_unravel_tex_primitive:nnn { lpcode          } { assign_font_int } { 2 }
845 \_unravel_tex_primitive:nnn { rpcode          } { assign_font_int } { 3 }
846 \_unravel_tex_primitive:nnn { efcode          } { assign_font_int } { 4 }
847 \_unravel_tex_primitive:nnn { tagcode         } { assign_font_int } { 5 }
848 \_unravel_tex_primitive:nnn { pdfnoligatures  } { assign_font_int } { 6 }
849 \_unravel_tex_primitive:nnn { knbscode       } { assign_font_int } { 7 }
850 \_unravel_tex_primitive:nnn { stbscode       } { assign_font_int } { 8 }
851 \_unravel_tex_primitive:nnn { shbscode       } { assign_font_int } { 9 }
852 \_unravel_tex_primitive:nnn { knbccode      } { assign_font_int } { 10 }
853 \_unravel_tex_primitive:nnn { knaccode      } { assign_font_int } { 11 }
854 \_unravel_tex_primitive:nnn { spacefactor   } { set_aux } { 102 }
855 \_unravel_tex_primitive:nnn { prevdepth     } { set_aux } { 1 }
856 \_unravel_tex_primitive:nnn { prevgraf     } { set_prev_graf } { 0 }
857 \_unravel_tex_primitive:nnn { pagegoal     } { set_page_dimen } { 0 }
858 \_unravel_tex_primitive:nnn { pagetotal    } { set_page_dimen } { 1 }
859 \_unravel_tex_primitive:nnn { pagestretch  } { set_page_dimen } { 2 }
860 \_unravel_tex_primitive:nnn { pagefilstretch } { set_page_dimen } { 3 }

```

```

861 \__unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 4 }
862 \__unravel_tex_primitive:nnn { pagefilllstretch } { set_page_dimen } { 5 }
863 \__unravel_tex_primitive:nnn { pageshrink } { set_page_dimen } { 6 }
864 \__unravel_tex_primitive:nnn { pagedepth } { set_page_dimen } { 7 }
865 \__unravel_tex_primitive:nnn { deadcycles } { set_page_int } { 0 }
866 \__unravel_tex_primitive:nnn { insertpenalties } { set_page_int } { 1 }
867 \__unravel_tex_primitive:nnn { interactionmode } { set_page_int } { 2 }
868 \__unravel_tex_primitive:nnn { wd } { set_box_dimen } { 1 }
869 \__unravel_tex_primitive:nnn { dp } { set_box_dimen } { 2 }
870 \__unravel_tex_primitive:nnn { ht } { set_box_dimen } { 3 }
871 \__unravel_tex_primitive:nnn { parshape } { set_shape } { 0 }
872 \__unravel_tex_primitive:nnn { interlinepenalties } { set_shape } { 1 }
873 \__unravel_tex_primitive:nnn { clubpenalties } { set_shape } { 2 }
874 \__unravel_tex_primitive:nnn { widowpenalties } { set_shape } { 3 }
875 \__unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
876 \__unravel_tex_primitive:nnn { catcode } { def_code } { 0 }
877 \__unravel_tex_primitive:nnn { lccode } { def_code } { 256 }
878 \__unravel_tex_primitive:nnn { uccode } { def_code } { 512 }
879 \__unravel_tex_primitive:nnn { sfcode } { def_code } { 768 }
880 \__unravel_tex_primitive:nnn { mathcode } { def_code } { 1024 }
881 \__unravel_tex_primitive:nnn { delcode } { def_code } { 1591 }
882 \__unravel_tex_primitive:nnn { textfont } { def_family } { -48 }
883 \__unravel_tex_primitive:nnn { scriptfont } { def_family } { -32 }
884 \__unravel_tex_primitive:nnn { scriptscriptfont } { def_family } { -16 }
885 \__unravel_tex_primitive:nnn { nullfont } { set_font } { 0 }
886 \__unravel_tex_primitive:nnn { font } { def_font } { 0 }
887 \__unravel_tex_primitive:nnn { count } { register } { 1 000 000 }
888 \__unravel_tex_primitive:nnn { dimen } { register } { 2 000 000 }
889 \__unravel_tex_primitive:nnn { skip } { register } { 3 000 000 }
890 \__unravel_tex_primitive:nnn { muskip } { register } { 4 000 000 }
891 \__unravel_tex_primitive:nnn { advance } { advance } { 0 }
892 \__unravel_tex_primitive:nnn { multiply } { multiply } { 0 }
893 \__unravel_tex_primitive:nnn { divide } { divide } { 0 }
894 \__unravel_tex_primitive:nnn { long } { prefix } { 1 }
895 \__unravel_tex_primitive:nnn { outer } { prefix } { 2 }
896 \__unravel_tex_primitive:nnn { global } { prefix } { 4 }
897 \__unravel_tex_primitive:nnn { protected } { prefix } { 8 }
898 \__unravel_tex_primitive:nnn { let } { let } { 0 }
899 \__unravel_tex_primitive:nnn { futurelet } { let } { 1 }
900 \__unravel_tex_primitive:nnn { chardef } { shorthand_def } { 0 }
901 \__unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
902 \__unravel_tex_primitive:nnn { countdef } { shorthand_def } { 2 }
903 \__unravel_tex_primitive:nnn { dimendef } { shorthand_def } { 3 }
904 \__unravel_tex_primitive:nnn { skipdef } { shorthand_def } { 4 }
905 \__unravel_tex_primitive:nnn { muskipdef } { shorthand_def } { 5 }
906 \__unravel_tex_primitive:nnn { toksdef } { shorthand_def } { 6 }
907 \__unravel_tex_primitive:nnn { read } { read_to_cs } { 0 }
908 \__unravel_tex_primitive:nnn { readline } { read_to_cs } { 1 }
909 \__unravel_tex_primitive:nnn { def } { def } { 0 }
910 \__unravel_tex_primitive:nnn { gdef } { def } { 1 }

```

```

911 \__unravel_tex_primitive:nnn { edef          } { def } { 2 }
912 \__unravel_tex_primitive:nnn { xdef          } { def } { 3 }
913 \__unravel_tex_primitive:nnn { setbox        } { set_box } { 0 }
914 \__unravel_tex_primitive:nnn { hyphenation   } { hyph_data } { 0 }
915 \__unravel_tex_primitive:nnn { patterns      } { hyph_data } { 1 }
916 \__unravel_tex_primitive:nnn { batchmode     } { set_interaction } { 0 }
917 \__unravel_tex_primitive:nnn { nonstopmode   } { set_interaction } { 1 }
918 \__unravel_tex_primitive:nnn { scrollmode     } { set_interaction } { 2 }
919 \__unravel_tex_primitive:nnn { errorstopmode  } { set_interaction } { 3 }
920 \__unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
921 \__unravel_tex_primitive:nnn { pdfcopyfont    } { pdf_copy_font } { 0 }
922 \__unravel_tex_primitive:nnn { undefined      } { undefined_cs } { 0 }
923 \__unravel_tex_primitive:nnn { ndefined       } { undefined_cs } { 0 }
924 \__unravel_tex_primitive:nnn { expandafter    } { expand_after } { 0 }
925 \__unravel_tex_primitive:nnn { unless         } { expand_after } { 1 }
926 \__unravel_tex_primitive:nnn { pdfprimitive   } { no_expand } { 1 }
927 \__unravel_tex_primitive:nnn { noexpand       } { no_expand } { 0 }
928 \__unravel_tex_primitive:nnn { input          } { input } { 0 }
929 \__unravel_tex_primitive:nnn { endinput       } { input } { 1 }
930 \__unravel_tex_primitive:nnn { scantokens     } { input } { 2 }
931 \__unravel_tex_primitive:nnn { if             } { if_test } { 0 }
932 \__unravel_tex_primitive:nnn { ifcat          } { if_test } { 1 }
933 \__unravel_tex_primitive:nnn { ifnum          } { if_test } { 2 }
934 \__unravel_tex_primitive:nnn { ifdim          } { if_test } { 3 }
935 \__unravel_tex_primitive:nnn { ifodd          } { if_test } { 4 }
936 \__unravel_tex_primitive:nnn { ifvmode        } { if_test } { 5 }
937 \__unravel_tex_primitive:nnn { ifhmode        } { if_test } { 6 }
938 \__unravel_tex_primitive:nnn { ifmmode        } { if_test } { 7 }
939 \__unravel_tex_primitive:nnn { ifinner        } { if_test } { 8 }
940 \__unravel_tex_primitive:nnn { ifvoid         } { if_test } { 9 }
941 \__unravel_tex_primitive:nnn { ifhbox         } { if_test } { 10 }
942 \__unravel_tex_primitive:nnn { ifvbox         } { if_test } { 11 }
943 \__unravel_tex_primitive:nnn { ifx            } { if_test } { 12 }
944 \__unravel_tex_primitive:nnn { ifeof          } { if_test } { 13 }
945 \__unravel_tex_primitive:nnn { iftrue         } { if_test } { 14 }
946 \__unravel_tex_primitive:nnn { iffalse        } { if_test } { 15 }
947 \__unravel_tex_primitive:nnn { ifcase         } { if_test } { 16 }
948 \__unravel_tex_primitive:nnn { ifdefined      } { if_test } { 17 }
949 \__unravel_tex_primitive:nnn { ifcsname       } { if_test } { 18 }
950 \__unravel_tex_primitive:nnn { iffontchar     } { if_test } { 19 }
951 \__unravel_tex_primitive:nnn { ifincsname     } { if_test } { 20 }
952 \__unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
953 \__unravel_tex_primitive:nnn { ifpdfabsnum    } { if_test } { 22 }
954 \__unravel_tex_primitive:nnn { ifpdfabsdim    } { if_test } { 23 }
955 \__unravel_tex_primitive:nnn { fi            } { fi_or_else } { 2 }
956 \__unravel_tex_primitive:nnn { else          } { fi_or_else } { 3 }
957 \__unravel_tex_primitive:nnn { or            } { fi_or_else } { 4 }
958 \__unravel_tex_primitive:nnn { csname        } { cs_name } { 0 }
959 \__unravel_tex_primitive:nnn { number        } { convert } { 0 }
960 \__unravel_tex_primitive:nnn { romannumeral   } { convert } { 1 }

```

```

961 \__unravel_tex_primitive:nnn { string          } { convert } { 2 }
962 \__unravel_tex_primitive:nnn { meaning         } { convert } { 3 }
963 \__unravel_tex_primitive:nnn { fontname        } { convert } { 4 }
964 \__unravel_tex_primitive:nnn { eTeXrevision    } { convert } { 5 }
965 \__unravel_tex_primitive:nnn { pdftexrevision  } { convert } { 6 }
966 \__unravel_tex_primitive:nnn { pdftexbanner    } { convert } { 7 }
967 \__unravel_tex_primitive:nnn { pdffontname     } { convert } { 8 }
968 \__unravel_tex_primitive:nnn { pdffontobjnum   } { convert } { 9 }
969 \__unravel_tex_primitive:nnn { pdffontsize     } { convert } { 10 }
970 \__unravel_tex_primitive:nnn { pdfpageref      } { convert } { 11 }
971 \__unravel_tex_primitive:nnn { pdfxformname    } { convert } { 12 }
972 \__unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
973 \__unravel_tex_primitive:nnn { pdfescapename   } { convert } { 14 }
974 \__unravel_tex_primitive:nnn { leftmarginkern  } { convert } { 15 }
975 \__unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
976 \__unravel_tex_primitive:nnn { pdfstrcmp       } { convert } { 17 }
977 \__unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
978 \__unravel_tex_primitive:nnn { pdfescapehex    } { convert } { 19 }
979 \__unravel_tex_primitive:nnn { pdfunescapehex   } { convert } { 20 }
980 \__unravel_tex_primitive:nnn { pdfcreationdate  } { convert } { 21 }
981 \__unravel_tex_primitive:nnn { pdffilemoddate   } { convert } { 22 }
982 \__unravel_tex_primitive:nnn { pdffilesize     } { convert } { 23 }
983 \__unravel_tex_primitive:nnn { pdfmdfivesum    } { convert } { 24 }
984 \__unravel_tex_primitive:nnn { pdffiledump     } { convert } { 25 }
985 \__unravel_tex_primitive:nnn { pdfmatch        } { convert } { 26 }
986 \__unravel_tex_primitive:nnn { pdflastmatch    } { convert } { 27 }
987 \__unravel_tex_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
988 \__unravel_tex_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
989 \__unravel_tex_primitive:nnn { pdfinsertht     } { convert } { 30 }
990 \__unravel_tex_primitive:nnn { pdfximagebbox    } { convert } { 31 }
991 \__unravel_tex_primitive:nnn { jobname         } { convert } { 32 }
992 \__unravel_tex_primitive:nnn { the             } { the } { 0 }
993 \__unravel_tex_primitive:nnn { unexpanded      } { the } { 1 }
994 \__unravel_tex_primitive:nnn { detokenize      } { the } { 5 }
995 \__unravel_tex_primitive:nnn { topmark         } { top_bot_mark } { 0 }
996 \__unravel_tex_primitive:nnn { firstmark       } { top_bot_mark } { 1 }
997 \__unravel_tex_primitive:nnn { botmark        } { top_bot_mark } { 2 }
998 \__unravel_tex_primitive:nnn { splitfirstmark  } { top_bot_mark } { 3 }
999 \__unravel_tex_primitive:nnn { splitbotmark    } { top_bot_mark } { 4 }
1000 \__unravel_tex_primitive:nnn { topmarks       } { top_bot_mark } { 5 }
1001 \__unravel_tex_primitive:nnn { firstmarks     } { top_bot_mark } { 6 }
1002 \__unravel_tex_primitive:nnn { botmarks       } { top_bot_mark } { 7 }
1003 \__unravel_tex_primitive:nnn { splitfirstmarks } { top_bot_mark } { 8 }
1004 \__unravel_tex_primitive:nnn { splitbotmarks  } { top_bot_mark } { 9 }

```

2.4 Get next token

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

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

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

`__unravel_get_next`: If the input is empty, forcefully exit. Otherwise, remove the first token in the input, and store it in `\l__unravel_head_gtl`. Set `\l__unravel_head_token` equal in meaning to that first token. Then set `\l__unravel_head_tl` to contain the token, unless it is a begin-group or end-group character, in which case this token list is emptied.

```

1005 \cs_new_protected_nopar:Npn \__unravel_get_next:
1006 {
1007   \__unravel_input_if_empty:TF
1008   { \__unravel_exit:w }
1009   {
1010     \__unravel_input_gpop:N \l__unravel_head_gtl
1011     \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1012     \gtl_if_tl:NTF \l__unravel_head_gtl
1013     {
1014       \tl_set:Nx \l__unravel_head_tl
1015       { \gtl_head:N \l__unravel_head_gtl }
1016     }
1017     { \tl_clear:N \l__unravel_head_tl }
1018   }
1019 }
1020 \cs_new_protected_nopar:Npn \__unravel_get_next_aux:w
1021 { \cs_set_eq:NN \l__unravel_head_token }

```

(End definition for `__unravel_get_next`.)

`__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`.

```

1022 \cs_new_protected_nopar:Npn \__unravel_get_token:
1023 {
1024   \__unravel_get_next:
1025   \__unravel_set_cmd:
1026 }

```

(End definition for `__unravel_get_token`.)

`__unravel_set_cmd`: After the call to `__unravel_get_next`:, we find the command code `\l__unravel_head_cmd_int` and the character code `\l__unravel_head_char_int`, based only on `\l__unravel_head_token`. First set `\l__unravel_head_meaning_tl` from the `\meaning` of the first token. If the corresponding primitive exists, use the information to set the two integers. If the token is expandable, it can either be a macro or be a primitive that

we somehow do not know (*e.g.*, an expandable X_YTeX or LuaTeX primitive perhaps). Otherwise, it can be a control sequence or a character.

```

1027 \cs_new_protected_nopar:Npn \__unravel_set_cmd:
1028 {
1029   \__unravel_set_cmd_aux_meaning:
1030   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1031   { }
1032   {
1033     \__unravel_token_if_expandable:NTF \l__unravel_head_token
1034     {
1035       \token_if_macro:NTF \l__unravel_head_token
1036       { \__unravel_set_cmd_aux_macro: }
1037       { \__unravel_set_cmd_aux_unknown: }
1038     }
1039     {
1040       \token_if_cs:NTF \l__unravel_head_token
1041       { \__unravel_set_cmd_aux_cs: }
1042       { \__unravel_set_cmd_aux_char: }
1043     }
1044   }
1045 }

```

(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.

```

1046 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_meaning:
1047 {
1048   \tl_set:Nx \l__unravel_head_meaning_tl
1049   {
1050     \exp_after:wN \__unravel_strip_escape:w
1051     \token_to_meaning:N \l__unravel_head_token
1052     \tl_to_str:n { : }
1053   }
1054   \tl_set:Nx \l__unravel_head_meaning_tl
1055   {
1056     \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1057     \l__unravel_head_meaning_tl \q_stop
1058   }
1059 }
1060 \use:x
1061 {
1062   \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1063   ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1064 }

```

(End definition for __unravel_set_cmd_aux_meaning:.)

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

```

1065 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nTF #1#2
1066 {
1067   \cs_if_exist:cTF { c\_unravel_tex_#1_tl }
1068   {
1069     \exp_last_unbraced:Nv \_unravel_set_cmd_aux_primitive:nn
1070     { c\_unravel_tex_#1_tl }
1071     #2
1072   }
1073 }
1074 \cs_generate_variant:Nn \_unravel_set_cmd_aux_primitive:nTF { o }
1075 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nn #1#2
1076 {
1077   \int_set:Nn \l\_unravel_head_cmd_int {#1}
1078   \int_set:Nn \l\_unravel_head_char_int {#2}
1079 }

```

(End definition for _unravel_set_cmd_aux_primitive:nTF and _unravel_set_cmd_aux_primitive:oTF.)

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

```

1080 \cs_new_protected_nopar:Npn \_unravel_set_cmd_aux_macro:
1081 {
1082   \int_set:Nn \l\_unravel_head_cmd_int { \_unravel_tex_use:n { call } }
1083   \int_zero:N \l\_unravel_head_char_int
1084 }

```

(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`.

```

1085 \cs_new_protected_nopar:Npn \_unravel_set_cmd_aux_unknown:
1086 {
1087   \exp_last_unbraced:Nv \_unravel_set_cmd_aux_primitive:nn
1088   \c\_unravel_tex_relax_tl
1089   \msg_error:nnx { unravel } { unknown-primitive }
1090   { \l\_unravel_head_meaning_tl }
1091 }

```

(End definition for _unravel_set_cmd_aux_unknown:.)

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

```

1092 \cs_new_protected_nopar:Npn \_unravel_set_cmd_aux_cs:
1093 {
1094   \tl_if_in:NoTF \l\_unravel_head_meaning_tl
1095   { \tl_to_str:n { elect~font } }
1096   {

```

```

1097         \exp_last_unbraced:N \__unravel_set_cmd_aux_primitive:nn
1098         \c__unravel_tex_nullfont_tl
1099     }
1100     { \__unravel_set_cmd_aux_numeric: }
1101 }

```

(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 one of count, dimen, skip, muskip, or toks. In the first two 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.

```

1102 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_numeric:
1103 {
1104     \tl_set:Nx \l__unravel_tmpa_tl
1105     {
1106         \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1107         \l__unravel_head_meaning_tl + 0
1108     }
1109     \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1110     \l__unravel_tmpa_tl \q_stop
1111 }
1112 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1113 {
1114     \if_int_compare:w '#1 < 'A \exp_stop_f:
1115     \exp_not:N \q_mark
1116     \exp_after:wN \use_i:nn
1117     \fi:
1118     #1 \__unravel_set_cmd_aux_numeric:N
1119 }
1120 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1121 {
1122     \str_case:nnF {#1}
1123     {
1124         { char }      { \__unravel_set_cmd_aux_given:n { char_given } }
1125         { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1126     }
1127     {
1128         \__unravel_set_cmd_aux_primitive:nTF {#1}
1129         { }
1130         { \__unravel_set_cmd_aux_unknown: }
1131         \int_add:Nn \l__unravel_head_char_int { 100 000 }
1132     }
1133     \int_add:Nn \l__unravel_head_char_int {#2}
1134 }

```

```

1135 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1136 {
1137     \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1138     \int_zero:N \l__unravel_head_char_int
1139 }

```

(End definition for __unravel_set_cmd_aux_numeric:, __unravel_set_cmd_aux_numeric:w, and __unravel_set_cmd_aux_given:n.)

__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 ‘.

```

1140 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_char:
1141 {
1142     \tl_set:Nx \l__unravel_head_meaning_tl
1143     { \token_to_meaning:N \l__unravel_head_token }
1144     \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1145     { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1146     \exp_after:wN \__unravel_set_cmd_aux_char:w
1147     \l__unravel_head_meaning_tl \q_stop
1148     \exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1149     { ‘ \__unravel_token_to_char:N \l__unravel_head_token }
1150 }
1151 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1152 {
1153     \int_set:Nn \l__unravel_head_cmd_int
1154     { \__unravel_tex_use:n { #1_char } }
1155 }

```

(End definition for __unravel_set_cmd_aux_char:.)

2.5 Manipulating the input

2.5.1 Elementary operations

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

```

1156 \cs_new_nopar:Npn \__unravel_input_to_str:
1157 {
1158     \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1159     \__unravel_input_to_str_aux:n
1160 }
1161 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1162 { \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.

```

1163 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1164 {
1165   \int_compare:nNnTF \g__unravel_input_int = \c_zero
1166     { \use_i:nn }
1167     {
1168       \gtl_if_empty:cTF
1169       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1170       {
1171         \int_gdecr:N \g__unravel_input_int
1172         \__unravel_input_if_empty:TF
1173       }
1174       {
1175         \__unravel_input_split:
1176         \use_ii:nn
1177       }
1178     }
1179 }

```

(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

```

1180 \cs_new_protected_nopar:Npn \__unravel_input_split:
1181 {
1182   \int_compare:nNnT \g__unravel_input_int = \c_one
1183     {
1184       \exp_args:Nc \__unravel_input_split_aux:N
1185       { g__unravel_input_1_gtl }
1186     }
1187 }
1188 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1189 {
1190   \gtl_if_tl:NT #1
1191   {
1192     \gtl_if_head_is_N_type:NT #1
1193     {
1194       \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1195       \exp_last_unbraced:Nx \__unravel_input_split_auxii:N
1196       { \tl_head:N \l__unravel_input_tmpa_tl }
1197     }
1198   }
1199 }
1200 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1201 {
1202   \token_if_parameter:NF #1
1203   {

```

```

1204 \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1205 { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1206 \group_begin:
1207 \cs_set:Npn \__unravel_input_split_auxiii:w
1208 ##1 \__unravel_input_split_end: { + 1 }
1209 \int_gset:Nn \g__unravel_input_int
1210 { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1211 \group_end:
1212 \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1213 \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1214 }
1215 }
1216 \cs_new_nopar:Npn \__unravel_input_split_end: { }
1217 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1218 #1 \__unravel_input_split_end:
1219 {
1220 \gtl_gclear_new:c
1221 { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1222 \gtl_gset:cn
1223 { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1224 \int_gdecr:N \g__unravel_input_tmpa_int
1225 }

```

(End definition for __unravel_input_split:.)

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

```

1226 \cs_new_protected_nopar:Npn \__unravel_input_gset:n
1227 {
1228 \int_gzero:N \g__unravel_input_int
1229 \__unravel_back_input:n
1230 }

```

(End definition for __unravel_input_gset:n.)

__unravel_input_get:N

```

1231 \cs_new_protected:Npn \__unravel_input_get:N #1
1232 {
1233 \__unravel_input_if_empty:TF
1234 { \gtl_set:Nn #1 { \q_no_value } }
1235 {
1236 \gtl_get_left:cN
1237 { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1238 }
1239 }

```

(End definition for __unravel_input_get:N.)

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

```

1240 \cs_new_protected:Npn \__unravel_input_gpop:N #1

```

```

1241 {
1242   \__unravel_input_if_empty:TF
1243   { \gtl_set:Nn #1 { \q_no_value } }
1244   {
1245     \gtl_gpop_left:cN
1246     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1247   }
1248 }

```

(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.

```

1249 \cs_new_protected_nopar:Npn \__unravel_input_merge:
1250 {
1251   \int_gdecr:N \g__unravel_input_int
1252   \gtl_gconcat:ccc
1253   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1254   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1255   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1256   \gtl_gclear:c
1257   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1258 }

```

(End definition for __unravel_input_merge:.)

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

__unravel_input_gpop_item_aux:NN

```

1259 \prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
1260 {
1261   \int_compare:nNnTF \g__unravel_input_int = \c_zero
1262   { \prg_return_false: }
1263   {
1264     \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1265     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1266   }
1267 }
1268 \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
1269 {
1270   \gtl_gpop_left_item:NNTF #1#2
1271   { \prg_return_true: }
1272   {
1273     \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero
1274     { \prg_return_false: }
1275     {
1276       \int_compare:nNnTF \g__unravel_input_int = \c_one
1277       { \prg_return_false: }

```



```

1278         {
1279             \_unravel_input_merge:
1280             \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1281             {
1282                 g\_unravel_input_
1283                 \int_use:N \g\_unravel_input_int _gtl
1284             }
1285             #2
1286         }
1287     }
1288 }
1289 }

```

(End definition for _unravel_input_gpop_item:NTF.)

_unravel_input_gpop_tl:N

```

1290 \cs_new_protected:Npn \_unravel_input_gpop_tl:N #1
1291 { \tl_clear:N #1 \_unravel_input_gpop_tl_aux:N #1 }
1292 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:N #1
1293 {
1294     \int_compare:nNnF \g\_unravel_input_int = \c_zero
1295     {
1296         \exp_args:Nc \_unravel_input_gpop_tl_aux:NN
1297         { g\_unravel_input_ \int_use:N \g\_unravel_input_int _gtl } #1
1298     }
1299 }
1300 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:NN #1#2
1301 {
1302     \gtl_if_tl:NTF #1
1303     {
1304         \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1305         \gtl_gclear:N #1
1306         \int_gdecr:N \g\_unravel_input_int
1307         \_unravel_input_gpop_tl_aux:N #2
1308     }
1309     {
1310         \int_compare:nNnTF \g\_unravel_input_int > \c_one
1311         { \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero }
1312         { \use_i:nn }
1313         {
1314             \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1315             \gtl_gpop_left_tl:N #1
1316         }
1317         {
1318             \_unravel_input_merge:
1319             \_unravel_input_gpop_tl_aux:N #2
1320         }
1321     }
1322 }

```

(End definition for _unravel_input_gpop_tl:N.)

`__unravel_back_input:n` Insert a token list back into the input. Use `\gtl_gclear_new:c` to define the gtl variable if necessary: this happens whenever a new largest value of `\g__unravel_input_int` is reached.

```

1323 \cs_new_protected_nopar:Npn \__unravel_back_input:n
1324 {
1325   \int_gincr:N \g__unravel_input_int
1326   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1327   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1328 }
1329 \cs_generate_variant:Nn \__unravel_back_input:n { x , V , o }

```

(End definition for __unravel_back_input:n and __unravel_back_input:x.)

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

```

1330 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1331 {
1332   \gtl_if_tl:NTF #1
1333   { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1334   {
1335     \gtl_gconcat:cNc
1336     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1337     #1
1338     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1339   }
1340 }

```

(End definition for __unravel_back_input_gtl:N.)

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

```

1341 \cs_new_protected_nopar:Npn \__unravel_back_input:
1342 { \__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.

```

1343 \cs_new_protected_nopar:Npn \__unravel_back_input_tl_o:
1344 {
1345   \tl_set:Nx \l__unravel_tmpa_tl
1346   { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1347   \__unravel_back_input:V \l__unravel_tmpa_tl
1348   \__unravel_print_done:x
1349   { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1350 }

```

(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.

```

1351 \cs_new_protected_nopar:Npn \__unravel_insert_relax:
1352 {
1353   \__unravel_back_input:
1354   \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1355   \__unravel_back_input:
1356   \__unravel_print_action:
1357 }

```

(End definition for __unravel_insert_relax:.)

`__unravel_insert_group_begin_error:`

```

1358 \cs_new_protected_nopar:Npn \__unravel_insert_group_begin_error:
1359 {
1360   \msg_error:nn { unravel } { missing-lbrace }
1361   \__unravel_back_input:
1362   \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1363   \__unravel_back_input:
1364   \__unravel_print_action:
1365 }

```

(End definition for __unravel_insert_group_begin_error:.)

`__unravel_insert_dollar_error:`

```

1366 \cs_new_protected_nopar:Npn \__unravel_insert_dollar_error:
1367 {
1368   \__unravel_back_input:
1369   \__unravel_back_input:n { $ } % $
1370   \msg_error:nn { unravel } { missing-dollar }
1371   \__unravel_print_action:
1372 }

```

(End definition for __unravel_insert_dollar_error:.)

2.5.3 Macro calls

`__unravel_macro_prefix:N`
`__unravel_macro_parameter:N`
`__unravel_macro_replacement:N`

```

1373 \use:x
1374 {
1375   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
1376   {
1377     \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
1378     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1379     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:n }
1380     \exp_not:N \q_stop
1381   }

```

```

1382 \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
1383 \exp_not:n {#1} \tl_to_str:n { : } \exp_not:n { #2 -> }
1384 \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1385 { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
1386 }
1387 \cs_new:Npn \__unravel_macro_prefix:N #1
1388 { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1389 \cs_new:Npn \__unravel_macro_parameter:N #1
1390 { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1391 \cs_new:Npn \__unravel_macro_replacement:N #1
1392 { \__unravel_macro_split_do:NN #1 \use_iii:nnn }

```

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

__unravel_macro_call: Macros are simply expanded once. We cannot determine precisely which tokens a macro will need for its parameters, but we know that it must form a balanced token list. Thus __unravel_macro_call_safe: we can be safe by extracting the longest balanced prefix in the input and working with __unravel_macro_call_quick: that.

__unravel_macro_call_quick_loop:NNN

__unravel_macro_call_quick_runaway:Nw

```

1393 \cs_new_protected_nopar:Npn \__unravel_macro_call:
1394 {
1395   \bool_if:NTF \g__unravel_speedup_macros_bool
1396   {
1397     \tl_set:Nx \l__unravel_tmpa_tl
1398     { ^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1399     \tl_if_in:NVTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1400     { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1401   }
1402   { \__unravel_macro_call_safe: }
1403   \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1404   \__unravel_print_done:x { \g__unravel_action_text_str }
1405 }
1406 \cs_new_protected_nopar:Npn \__unravel_macro_call_safe:
1407 {
1408   \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1409   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1410 }
1411 \cs_new_protected_nopar:Npn \__unravel_macro_call_quick:
1412 {
1413   \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1414   { ? \use_none_delimit_by_q_stop:w } \q_stop
1415 }
1416 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1417 {
1418   \use_none:n #2
1419   \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1420   { \__unravel_macro_call_quick_runaway:Nw #3 }
1421   \tl_put_right:Nx \l__unravel_head_tl
1422   { { \exp_not:V \l__unravel_tmpa_tl } }
1423   \__unravel_macro_call_quick_loop:NNN

```

```

1424     #3
1425   }
1426 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1427 {
1428   \msg_error:nnxx { unravel } { runaway-macro-parameter }
1429   { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} }
1430 }

```

(End definition for __unravel_macro_call:.)

2.6 Expand next token

`__unravel_expand:` This is similar to `__unravel_do_step:`, but operates on expandable tokens rather than (non-expandable) commands. We mimic T_EX's structure, distinguishing macros from other commands (not quite sure why).

```

1431 \cs_new_protected_nopar:Npn \__unravel_expand:
1432 {
1433   \__unravel_set_action_text:
1434   \bool_if:NT \g__unravel_internal_debug_bool
1435   {
1436     \__unravel_set_cmd:
1437     \iow_term:x { Exp:~\int_use:N \l__unravel_head_cmd_int }
1438   }
1439   \token_if_macro:NTF \l__unravel_head_token
1440   { \__unravel_macro_call: }
1441   { \__unravel_expand_nonmacro: }
1442 }

```

(End definition for __unravel_expand:.)

`__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. If we recognize the meaning but there is no corresponding function, then we probably have not implemented it yet, so dump it in the output as is.

```

1443 \cs_new_protected_nopar:Npn \__unravel_expand_nonmacro:
1444 {
1445   \__unravel_set_cmd_aux_meaning:
1446   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1447   {
1448     \cs_if_exist_use:cF
1449     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1450     { \msg_error:nnx { unravel } { internal } { expandable } }
1451   }
1452   {
1453     \msg_error:nnx { unravel } { unknown-primitive }
1454     { \l__unravel_head_meaning_tl }
1455     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
1456     \__unravel_print_action:
1457   }
1458 }

```

(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.

```
1459 \cs_new_protected_nopar:Npn \__unravel_get_x_next:
1460 {
1461   \__unravel_get_next:
1462   \__unravel_token_if_expandable:NT \l__unravel_head_token
1463   {
1464     \__unravel_expand:
1465     \__unravel_get_x_next:
1466   }
1467 }
```

(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.

```
1468 \cs_new_protected_nopar:Npn \__unravel_get_x_or_protected:
1469 {
1470   \__unravel_get_next:
1471   \__unravel_token_if_protected:NF \l__unravel_head_token
1472   {
1473     \__unravel_expand:
1474     \__unravel_get_x_or_protected:
1475   }
1476 }
```

(End definition for `__unravel_get_x_or_protected:.`)

2.7 Basic scanning subroutines

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

```
1477 \cs_new_protected_nopar:Npn \__unravel_get_x_non_blank:
1478 {
1479   \__unravel_get_x_next:
1480   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1481   { \__unravel_get_x_non_blank: }
1482 }
```

(End definition for `__unravel_get_x_non_blank:.`)

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

```
1483 \cs_new_protected_nopar:Npn \__unravel_get_x_non_relax:
1484 {
1485   \__unravel_get_x_next:
1486   \token_if_eq_meaning:NNT \l__unravel_head_token \scan_stop:
1487   { \__unravel_get_x_non_relax: }
1488   {
1489     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
```

```

1490         { \_unravel_get_x_non_relax: }
1491     }
1492 }

```

(End definition for _unravel_get_x_non_relax:.)

_unravel_skip_optional_space:

```

1493 \cs_new_protected_nopar:Npn \_unravel_skip_optional_space:
1494 {
1495     \_unravel_get_x_next:
1496     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1497     { \_unravel_back_input: }
1498 }

```

(End definition for _unravel_skip_optional_space:.)

_unravel_scan_optional_equals:

See T_EX'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 \g__unravel_prev_input_seq.

```

1499 \cs_new_protected_nopar:Npn \_unravel_scan_optional_equals:
1500 {
1501     \_unravel_get_x_non_blank:
1502     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1503     { \_unravel_prev_input:n { = } }
1504     {
1505         \_unravel_prev_input_silent:n { = }
1506         \_unravel_back_input:
1507     }
1508 }

```

(End definition for _unravel_scan_optional_equals:.)

_unravel_scan_left_brace:

The presence of \relax is allowed before a begin-group character.

```

1509 \cs_new_protected_nopar:Npn \_unravel_scan_left_brace:
1510 {
1511     \_unravel_get_x_non_relax:
1512     \token_if_eq_catcode:NMF \l__unravel_head_token \c_group_begin_token
1513     { \_unravel_insert_group_begin_error: }
1514 }

```

(End definition for _unravel_scan_left_brace:.)

_unravel_scan_keyword:n

_unravel_scan_keyword:nTF

_unravel_scan_keyword_loop:NNN

_unravel_scan_keyword_test:NNTF

_unravel_scan_keyword_true:

_unravel_scan_keyword_false:w

The details of how T_EX looks for keywords are quite tricky to get right, in particular with respect to expansion, case-insensitivity, and spaces. We get rid of the case issue by requiring the keyword to be given in both cases, intertwined: for instance, _unravel_scan_keyword:n { pPtT }. Then loop through pairs of letters (which should be matching lowercase and uppercase letters). The looping auxiliary takes three arguments, the first of which is a boolean, true if spaces are allowed (no letter of the keyword has been found yet). At each iteration, get a token, with expansion, and test whether it is a non-active character equal (in character code) to either letter of the pair: this happens if the token is

not “definable” (neither a control sequence nor an active character) and it has the right string representation... well, it could also be doubled (macro parameter character), hence we look at the first character only; spaces become an empty string, but this works out because no keyword contains a space. So, at each iteration, if the token is the correct non-active character, add it to `\g__unravel_prev_input_seq` (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

1515 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1516   { \__unravel_scan_keyword:nTF {#1} { } { } }
1517 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1518   { T , F , TF }
1519   {
1520     \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl
1521     \__unravel_scan_keyword_loop:NNN \c_true_bool
1522     #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1523   }
1524 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1525   {
1526     \quark_if_recursion_tail_stop_do:nn {#2}
1527     { \__unravel_scan_keyword_true: }
1528     \quark_if_recursion_tail_stop_do:nn {#3}
1529     { \msg_error:nnx { unravel } { internal } { odd-keyword-length } }
1530     \__unravel_get_x_next:
1531     \__unravel_scan_keyword_test:NNTF #2#3
1532     {
1533       \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1534       \__unravel_scan_keyword_loop:NNN \c_false_bool
1535     }
1536     {
1537       \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
1538       { \__unravel_scan_keyword_false:w }
1539       \bool_if:NF #1
1540       { \__unravel_scan_keyword_false:w }
1541       \__unravel_scan_keyword_loop:NNN #1#2#3
1542     }
1543   }
1544 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1545   { TF }
1546   {

```



```

1547 \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
1548 { \prg_return_false: }
1549 {
1550   \str_if_eq_x:nnTF
1551   { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1552   { \prg_return_true: }
1553   {
1554     \str_if_eq_x:nnTF
1555     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
1556     { \prg_return_true: }
1557     { \prg_return_false: }
1558   }
1559 }
1560 }
1561 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_true:
1562 {
1563   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
1564   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1565   \prg_return_true:
1566 }
1567 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_false:w
1568 #1 \q_recursion_stop
1569 {
1570   \__unravel_back_input:
1571   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
1572   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1573   \prg_return_false:
1574 }

```

(End definition for __unravel_scan_keyword:n.)

__unravel_scan_font_ident: Find a font identifier.

```

1575 \cs_new_protected_nopar:Npn \__unravel_scan_font_ident:
1576 {
1577   \__unravel_get_x_non_blank:
1578   \__unravel_set_cmd:
1579   \int_case:nnF \l__unravel_head_cmd_int
1580   {
1581     { \__unravel_tex_use:n { def_font } }
1582     { \__unravel_prev_input:V \l__unravel_head_tl }
1583     { \__unravel_tex_use:n { letterspace_font } }
1584     { \__unravel_prev_input:V \l__unravel_head_tl }
1585     { \__unravel_tex_use:n { pdf_copy_font } }
1586     { \__unravel_prev_input:V \l__unravel_head_tl }
1587     { \__unravel_tex_use:n { set_font } }
1588     { \__unravel_prev_input:V \l__unravel_head_tl }
1589     { \__unravel_tex_use:n { def_family } }
1590     {
1591       \__unravel_prev_input:V \l__unravel_head_tl
1592       \__unravel_scan_int:

```

```

1593     }
1594   }
1595   {
1596     \msg_error:nn { unravel } { missing-font-id }
1597     \__unravel_prev_input:n { \__unravel_nullfont: }
1598     \__unravel_back_error:
1599   }
1600 }

```

(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`).

```

1601 \cs_new_protected_nopar:Npn \__unravel_scan_font_int:
1602   {
1603     \int_case:nnF \l__unravel_head_char_int
1604     {
1605       { 0 } { \__unravel_scan_font_ident: }
1606       { 1 } { \__unravel_scan_font_ident: }
1607       { 6 } { \__unravel_scan_font_ident: }
1608     }
1609     { \__unravel_scan_font_ident: \__unravel_scan_int: }
1610   }

```

(End definition for __unravel_scan_font_int:.)

`__unravel_scan_font_dimen:` Find operands for \fontdimen.

```

1611 \cs_new_protected_nopar:Npn \__unravel_scan_font_dimen:
1612   {
1613     \__unravel_scan_int:
1614     \__unravel_scan_font_ident:
1615   }

```

(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.

Scans something internal, and places its value, converted to the given level, to the right of the last item of `\g__unravel_prev_input_seq`, then sets `\g__unravel_val_level_int` to the found level (level before conversion, so this may be higher than requested). Get in one go 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 is not between `min_internal=68` and `max_internal=89`, this step claims a level of 8. If the level that will be produced is 4 or 5, but the argument `#1` is not, or if the level is 8 (exercise: check that the conditional indeed checks for this case, given that ε -TeX rounds “to nearest, ties away from zero”), then complain. Otherwise, fetch arguments if there are any: the scanning is performed after placing the current token in a new level of `prev_input` and telling the user about it. Once done with this step, `\l__unravel_head_tl` contains the tokens found. Convert them to the wanted level with `__unravel_thing_use:nN` (at this stage, TeX may complain about a missing number or incompatible glue units), and place the result in `prev_input`. Finally, tell the user the tokens that have been found and their value (and, if there was a single token, its meaning). 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).

```

1616 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1617 {
1618   \__unravel_set_cmd:
1619   \__unravel_set_action_text:
1620   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1621   \exp_after:wN \__unravel_scan_something_aux:nwn
1622   \l__unravel_tmpa_tl \q_stop {#1}
1623 }
1624 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1625 {
1626   \int_compare:nNnTF
1627     { ( #1 + \c_two ) / \c_four } > { ( #3 + \c_two ) / \c_four }
1628     { \__unravel_back_input: }
1629     {
1630       \tl_if_empty:nF {#2}
1631       {
1632         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
1633         \__unravel_print_action:
1634         #2
1635         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1636       }
1637     }
1638   \tl_set:Nx \l__unravel_tmpa_tl
1639     { \__unravel_thing_use:nnN {#1} {#3} \l__unravel_head_tl }
1640   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
1641   \__unravel_set_action_text:
1642   \__unravel_set_action_text:x
1643     { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:N \l__unravel_tmpa_tl }
1644   \int_compare:nNnF {#3} > { 3 } { \__unravel_print_action: }

```

```

1645 \int_gset:Nn \g__unravel_val_level_int {#1}
1646 }

```

(End definition for __unravel_scan_something_internal:n.)

__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.

```

1647 \cs_new_nopar:Npn \__unravel_thing_case:
1648 {
1649   \int_case:nnF \l__unravel_head_cmd_int
1650   {
1651     { 68 } { 0 } % char_given
1652     { 69 } { 0 } % math_given
1653     { 70 } { \__unravel_thing_last_item: } % last_item
1654     { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
1655     { 72 } { 5 } % assign_toks
1656     { 73 } { 0 } % assign_int
1657     { 74 } { 1 } % assign_dimen
1658     { 75 } { 2 } % assign_glue
1659     { 76 } { 3 } % assign_mu_glue
1660     { 77 } { 1 \__unravel_scan_font_dimen: } % assign_font_dimen
1661     { 78 } { 0 \__unravel_scan_font_int: } % assign_font_int
1662     { 79 } { \__unravel_thing_set_aux: } % set_aux
1663     { 80 } { 0 } % set_prev_graf
1664     { 81 } { 1 } % set_page_dimen
1665     { 82 } { 0 } % set_page_int
1666     { 83 } { 1 \__unravel_scan_int: } % set_box_dimen
1667     { 84 } { 0 \__unravel_scan_int: } % set_shape
1668     { 85 } { 0 \__unravel_scan_int: } % def_code
1669     { 86 } { 4 \__unravel_scan_font_ident: } % def_family
1670     { 87 } { 4 \__unravel_scan_font_ident: } % set_font
1671     { 88 } { 4 \__unravel_scan_font_ident: } % def_font
1672     { 89 } { \__unravel_thing_register: } % register
1673   }
1674   { 8 }
1675 }
1676 \cs_new_nopar:Npn \__unravel_thing_set_aux:
1677 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
1678 \cs_new_nopar:Npn \__unravel_thing_last_item:
1679 {
1680   \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
1681   {
1682     \int_case:nnF \l__unravel_head_char_int
1683     {
1684       { 1 } { 1 } % lastkern
1685       { 2 } { 2 } % lastskip

```

```

1686     }
1687     { 0 } % other integer parameters
1688 }
1689 {
1690     \int_case:nnF \l__unravel_head_char_int
1691     {
1692         { 26 } { 0 \__unravel_scan_normal_glue: } % gluestretchorder
1693         { 27 } { 0 \__unravel_scan_normal_glue: } % glueshrinkorder
1694         { 28 } % fontcharwd
1695         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
1696         { 29 } % fontcharht
1697         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
1698         { 30 } % fontcharpd
1699         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
1700         { 31 } % fontcharic
1701         { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
1702         { 32 } { 1 \__unravel_scan_int: } % parshapelength
1703         { 33 } { 1 \__unravel_scan_int: } % parshapeindent
1704         { 34 } { 1 \__unravel_scan_int: } % parshapedimen
1705         { 35 } { 1 \__unravel_scan_normal_glue: } % gluestretch
1706         { 36 } { 1 \__unravel_scan_normal_glue: } % glueshrink
1707         { 37 } { 2 \__unravel_scan_mu_glue: } % mutoglu
1708         { 38 } { 3 \__unravel_scan_normal_glue: } % gluetomu
1709         { 39 } % numexpr
1710         { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
1711         { 40 } % dimexpr
1712         { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
1713         { 41 } % glueexpr
1714         { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
1715         { 42 } % muexpr
1716         { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
1717     }
1718     { }
1719 }
1720 }
1721 \cs_new_nopar:Npn \__unravel_thing_register:
1722 {
1723     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
1724     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = \c_zero
1725     { \__unravel_scan_int: }
1726 }

```

(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.

```

1727 \cs_new_protected:Npn \__unravel_scan_toks_register:
1728 {
1729     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
1730     { \__unravel_scan_int: }

```

1731 }

(End definition for _unravel_scan_toks_register:.)

_unravel_thing_use:nnN Given a level #1, call a function to convert the token list #2 to the correct level. This step may trigger T_EX errors, which should precisely match the expected ones.

```
1732 \cs_new:Npn \_unravel\_thing\_use:nnN #1#2
1733 {
1734   \int_case:nnF { \int_min:nn {#1} {#2} }
1735   {
1736     { 0 } \int_eval:n
1737     { 1 } \dim_eval:n
1738     { 2 } \skip_eval:n
1739     { 3 } \muskip_eval:n
1740   }
1741   { \_unravel\_the:w }
1742 }
```

(End definition for _unravel_thing_use:nnN.)

_unravel_scan_expr:N
_unravel_scan_expr_aux:NN
_unravel_scan_factor:N

```
1743 \cs_new_protected:Npn \_unravel\_scan\_expr:N #1
1744 { \_unravel\_scan\_expr\_aux:NN #1 \c_false_bool }
1745 \cs_new_protected:Npn \_unravel\_scan\_expr\_aux:NN #1#2
1746 {
1747   \_unravel\_get\_x\_non\_blank:
1748   \_unravel\_scan\_factor:N #1
1749   \_unravel\_scan\_expr\_op:NN #1#2
1750 }
1751 \cs_new_protected:Npn \_unravel\_scan\_expr\_op:NN #1#2
1752 {
1753   \_unravel\_get\_x\_non\_blank:
1754   \tl_case:NnF \l\_unravel\_head\_tl
1755   {
1756     \c\_unravel\_plus\_tl
1757     {
1758       \_unravel\_prev\_input:V \l\_unravel\_head\_tl
1759       \_unravel\_scan\_expr\_aux:NN #1#2
1760     }
1761     \c\_unravel\_minus\_tl
1762     {
1763       \_unravel\_prev\_input:V \l\_unravel\_head\_tl
1764       \_unravel\_scan\_expr\_aux:NN #1#2
1765     }
1766     \c\_unravel\_times\_tl
1767     {
1768       \_unravel\_prev\_input:V \l\_unravel\_head\_tl
1769       \_unravel\_get\_x\_non\_blank:
1770       \_unravel\_scan\_factor:N \_unravel\_scan\_int:
1771       \_unravel\_scan\_expr\_op:NN #1#2

```

```

1772     }
1773     \c__unravel_over_t1
1774     {
1775         \__unravel_prev_input:V \l__unravel_head_t1
1776         \__unravel_get_x_non_blank:
1777         \__unravel_scan_factor:N \__unravel_scan_int:
1778         \__unravel_scan_expr_op:NN #1#2
1779     }
1780     \c__unravel_rp_t1
1781     {
1782         \bool_if:NTF #2
1783         { \__unravel_prev_input:V \l__unravel_head_t1 }
1784         { \__unravel_back_input: }
1785     }
1786 }
1787 {
1788     \bool_if:NTF #2
1789     {
1790         \msg_error:nn { unravel } { missing-rparen }
1791         \__unravel_back_input:
1792         \__unravel_prev_input:V \c__unravel_rp_t1
1793     }
1794     {
1795         \token_if_eq_meaning:NMF \l__unravel_head_token \scan_stop:
1796         { \__unravel_back_input: }
1797     }
1798 }
1799 }
1800 \cs_new_protected:Npn \__unravel_scan_factor:N #1
1801 {
1802     \tl_if_eq:NNTF \l__unravel_head_t1 \c__unravel_lp_t1
1803     {
1804         \__unravel_prev_input:V \l__unravel_head_t1
1805         \__unravel_scan_expr_aux:NN #1 \c_true_bool
1806     }
1807     {
1808         \__unravel_back_input:
1809         #1
1810     }
1811 }

```

(End definition for __unravel_scan_expr:N.)

__unravel_scan_signs: Skips blanks, scans signs, and places them to the right of the last item of **__unravel_prev_input:n**.

```

1812 \cs_new_protected_nopar:Npn \__unravel_scan_signs:
1813 {
1814     \__unravel_get_x_non_blank:
1815     \tl_if_eq:NNTF \l__unravel_head_t1 \c__unravel_plus_t1
1816     {

```

```

1817     \_unravel_prev_input:V \l\_unravel_head_tl
1818     \_unravel_scan_signs:
1819   }
1820   {
1821     \tl_if_eq:NNT \l\_unravel_head_tl \c\_unravel_minus_tl
1822     {
1823       \_unravel_prev_input:V \l\_unravel_head_tl
1824       \_unravel_scan_signs:
1825     }
1826   }
1827 }

```

(End definition for _unravel_scan_signs:.)

```

\_unravel_scan_int:
\_unravel_scan_int_char:
\_unravel_scan_int_lq:
\_unravel_scan_int_explicit:n
1828 \cs_new_protected_nopar:Npn \_unravel_scan_int:
1829 {
1830   \_unravel_scan_signs:
1831   \_unravel_set_cmd:
1832   \_unravel_cmd_if_internal:TF
1833   { \_unravel_scan_something_internal:n { 0 } }
1834   { \_unravel_scan_int_char: }
1835 }
1836 \cs_new_protected_nopar:Npn \_unravel_scan_int_char:
1837 {
1838   \tl_case:NnF \l\_unravel_head_tl
1839   {
1840     \c\_unravel_lq_tl { \_unravel_scan_int_lq: }
1841     \c\_unravel_rq_tl
1842     {
1843       \_unravel_prev_input:V \l\_unravel_head_tl
1844       \_unravel_get_x_next:
1845       \_unravel_scan_int_explicit:n { ' }
1846     }
1847     \c\_unravel_dq_tl
1848     {
1849       \_unravel_prev_input:V \l\_unravel_head_tl
1850       \_unravel_get_x_next:
1851       \_unravel_scan_int_explicit:n { " }
1852     }
1853   }
1854   { \_unravel_scan_int_explicit:n { } }
1855 }
1856 \cs_new_protected_nopar:Npn \_unravel_scan_int_lq:
1857 {
1858   \_unravel_get_next:
1859   \_unravel_gtl_if_head_is_definable:NF \l\_unravel_head_gtl
1860   {
1861     \tl_set:Nx \l\_unravel_head_tl
1862     { \_unravel_token_to_char:N \l\_unravel_head_token }

```



```

1863     }
1864     \tl_set:Nx \l__unravel_tmpa_tl
1865     { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
1866     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
1867     \__unravel_print_action:x
1868     { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
1869     \__unravel_skip_optional_space:
1870   }
1871   \cs_new_protected:Npn \__unravel_scan_int_explicit:n #1
1872   {
1873     \if_int_compare:w \c_one
1874     < #1 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
1875     \exp_after:wN \use_i:nn
1876     \else:
1877     \exp_after:wN \use_ii:nn
1878     \fi:
1879     {
1880       \__unravel_prev_input:V \l__unravel_head_tl
1881       \__unravel_get_x_next:
1882       \__unravel_scan_int_explicit:n {#1}
1883     }
1884     {
1885       \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1886       { \__unravel_back_input: }
1887     }
1888   }

```

(End definition for __unravel_scan_int:.)

__unravel_scan_normal_dimen:

```

1889 \cs_new_protected_nopar:Npn \__unravel_scan_normal_dimen:
1890 { \__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_dimen_unit:nN`.

```

1891 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
1892 {
1893   \__unravel_scan_signs:
1894   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
1895   \__unravel_set_cmd:
1896   \__unravel_cmd_if_internal:TF
1897   {
1898     \int_compare:nNnTF {#1} = { 3 }
1899     {

```

```

1900         \__unravel_scan_something_internal:n { 3 }
1901         \int_case:nnF \g__unravel_val_level_int
1902         {
1903             { 0 } { \__unravel_scan_dim_unit:nN {#1} #2 }
1904             { 3 } { }
1905         }
1906         {
1907             \msg_error:nn { unravel } { incompatible-units }
1908             % ^^A todo: error recovery
1909         }
1910     }
1911     {
1912         \__unravel_scan_something_internal:n { 1 }
1913         \int_case:nnF \g__unravel_val_level_int
1914         {
1915             { 0 } { \__unravel_scan_dim_unit:nN {#1} #2 }
1916             { 3 } % ^^A todo: error recovery
1917             { \msg_error:nn { unravel } { incompatible-units } }
1918         }
1919         { }
1920     }
1921 }
1922 { \__unravel_scan_dimen_char:nN {#1} #2 }
1923 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1924 \__unravel_prev_input_silent:V \l__unravel_head_tl
1925 }
1926 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
1927 {
1928     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
1929     { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
1930     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
1931     {
1932         \__unravel_prev_input:n { . }
1933         \__unravel_scan_decimal_loop:
1934     }
1935     {
1936         \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl
1937         {
1938             \__unravel_back_input:
1939             \__unravel_scan_int:
1940             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
1941             { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
1942             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
1943             {
1944                 \__unravel_input_gpop:N \l__unravel_tmpb_gtl
1945                 \__unravel_prev_input:n { . }
1946                 \__unravel_scan_decimal_loop:
1947             }
1948         }
1949     }

```

```

1950         \__unravel_back_input:
1951         \__unravel_scan_int:
1952     }
1953 }
1954 \__unravel_scan_dim_unit:nN {#1} #2
1955 }
1956 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
1957 {
1958     \bool_if:NT #2
1959     {
1960         \__unravel_scan_keyword:nT { fF iI lL }
1961         {
1962             \__unravel_scan_inf_unit_loop:
1963             \__unravel_break:w
1964         }
1965     }
1966     \__unravel_get_x_non_blank:
1967     \__unravel_set_cmd:
1968     \__unravel_cmd_if_internal:TF
1969     {
1970         \seq_gput_right:Nn \g__unravel_prev_input_seq { }
1971         \__unravel_scan_something_internal:n {#1}
1972         \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
1973         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_tmpa_tl
1974         \__unravel_break:w
1975     }
1976     { \__unravel_back_input: }
1977     \int_compare:nNnT {#1} = { 3 }
1978     {
1979         \__unravel_scan_keyword:nT { mM uU } { \__unravel_break:w }
1980         \msg_error:nn { unravel } { missing-mudim }
1981         \__unravel_break:w
1982     }
1983     \__unravel_scan_keyword:nT { eE mM } { \__unravel_break:w }
1984     \__unravel_scan_keyword:nT { eE xX } { \__unravel_break:w }
1985     \__unravel_scan_keyword:nT { pP xX } { \__unravel_break:w }
1986     \__unravel_scan_keyword:n { tT rR uU eE }
1987     \__unravel_scan_keyword:nT { pP tT } { \__unravel_break:w }
1988     \__unravel_scan_keyword:nT { iI nN } { \__unravel_break:w }
1989     \__unravel_scan_keyword:nT { pP cC } { \__unravel_break:w }
1990     \__unravel_scan_keyword:nT { cC mM } { \__unravel_break:w }
1991     \__unravel_scan_keyword:nT { mM mM } { \__unravel_break:w }
1992     \__unravel_scan_keyword:nT { bB pP } { \__unravel_break:w }
1993     \__unravel_scan_keyword:nT { dD dD } { \__unravel_break:w }
1994     \__unravel_scan_keyword:nT { cC cC } { \__unravel_break:w }
1995     \__unravel_scan_keyword:nT { nN dD } { \__unravel_break:w }
1996     \__unravel_scan_keyword:nT { nN cC } { \__unravel_break:w }
1997     \__unravel_scan_keyword:nT { sS pP } { \__unravel_break:w }
1998     \__unravel_break_point:
1999 }

```

```

2000 \cs_new_protected_nopar:Npn \__unravel_scan_inf_unit_loop:
2001 { \__unravel_scan_keyword:nT { 1L } { \__unravel_scan_inf_unit_loop: } }
2002 \cs_new_protected_nopar:Npn \__unravel_scan_decimal_loop:
2003 {
2004   \__unravel_get_x_next:
2005   \tl_if_empty:NTF \l__unravel_head_tl
2006     { \use_ii:nn }
2007     { \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl }
2008     {
2009       \__unravel_prev_input:V \l__unravel_head_tl
2010       \__unravel_scan_decimal_loop:
2011     }
2012     {
2013       \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2014       { \__unravel_back_input: }
2015       \__unravel_prev_input_silent:n { ~ }
2016     }
2017 }

```

(End definition for __unravel_scan_dimen:nN.)

__unravel_scan_normal_glue:
 __unravel_scan_mu_glue:

```

2018 \cs_new_protected_nopar:Npn \__unravel_scan_normal_glue:
2019 { \__unravel_scan_glue:n { 2 } }
2020 \cs_new_protected_nopar:Npn \__unravel_scan_mu_glue:
2021 { \__unravel_scan_glue:n { 3 } }

```

(End definition for __unravel_scan_normal_glue: and __unravel_scan_mu_glue:.)

__unravel_scan_glue:n

```

2022 \cs_new_protected:Npn \__unravel_scan_glue:n #1
2023 {
2024   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2025   \__unravel_scan_signs:
2026   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2027   \__unravel_set_cmd:
2028   \__unravel_cmd_if_internal:TF
2029   {
2030     \__unravel_scan_something_internal:n {#1}
2031     \int_case:nnF \g__unravel_val_level_int
2032       {
2033         { 0 } { \__unravel_scan_dimen:nN {#1} \c_false_bool }
2034         {#1} { \__unravel_break:w }
2035       }
2036       {
2037         \int_compare:nNnF { \g__unravel_val_level_int + #1 } = 3
2038         { \msg_error:nn { unravel } { incompatible-units } }
2039       }
2040   }
2041   { \__unravel_back_input: \__unravel_scan_dimen:nN {#1} \c_false_bool }

```

```

2042 \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2043 \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2044 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2045 \__unravel_scan_keyword:nT { pP lL uU sS }
2046 { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2047 \__unravel_scan_keyword:nT { mM iI nN uU sS }
2048 { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2049 \__unravel_break_point:
2050 \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2051 \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2052 }

```

(End definition for __unravel_scan_glue:n.)

__unravel_scan_file_name:

```

2053 \cs_new_protected_nopar:Npn \__unravel_scan_file_name:
2054 {
2055   \bool_gset_true:N \g__unravel_name_in_progress_bool
2056   \__unravel_get_x_non_blank:
2057   \__unravel_scan_file_name_loop:
2058   \bool_gset_false:N \g__unravel_name_in_progress_bool
2059   \__unravel_prev_input_silent:n { ~ }
2060 }
2061 \cs_new_protected_nopar:Npn \__unravel_scan_file_name_loop:
2062 {
2063   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2064   { \__unravel_back_input: }
2065   {
2066     \tl_set:Nx \l__unravel_tmpa_tl
2067     { \__unravel_token_to_char:N \l__unravel_head_token }
2068     \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2069     {
2070       \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2071       \__unravel_get_x_next:
2072       \__unravel_scan_file_name_loop:
2073     }
2074   }
2075 }

```

(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.

```

2076 \cs_new_protected_nopar:Npn \__unravel_scan_r_token:
2077 {
2078   \bool_do_while:nn
2079   { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2080   { \__unravel_get_next: }
2081   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2082   {

```

```

2083         \msg_error:nn { unravel } { missing-cs }
2084         \__unravel_back_input:
2085         \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2086     }
2087     \__unravel_prev_input_silent:V \l__unravel_head_tl
2088     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2089 }

```

(End definition for __unravel_scan_r_token:.)

__unravel_scan_toks_to_str:

```

2090 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2091 {
2092     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2093     \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2094     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2095     \__unravel_prev_input_silent:x
2096     { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2097 }

```

(End definition for __unravel_scan_toks_to_str:.)

__unravel_scan_toks:NN

```

2098 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2099 {
2100     \bool_if:NT #1 { \__unravel_scan_param: }
2101     \__unravel_scan_left_brace:
2102     \bool_if:NTF #2
2103     { \__unravel_scan_group_x:N #1 }
2104     { \__unravel_scan_group_n:N #1 }
2105 }

```

(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.

```

2106 \cs_new_protected_nopar:Npn \__unravel_scan_param:
2107 {
2108     \tl_clear:N \l__unravel_tmpa_tl
2109     \__unravel_scan_param_aux:
2110     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2111     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2112 }
2113 \cs_new_protected_nopar:Npn \__unravel_scan_param_aux:
2114 {
2115     \__unravel_get_next:
2116     \tl_concat:NNN \l__unravel_tmpa_tl
2117     \l__unravel_tmpa_tl \l__unravel_head_tl
2118     \tl_if_empty:NTF \l__unravel_head_tl

```

```

2119     { \_unravel_back_input: } { \_unravel_scan_param_aux: }
2120 }

```

(End definition for _unravel_scan_param:.)

_unravel_scan_group_n:N

```

2121 \cs_new_protected:Npn \_unravel_scan_group_n:N #1
2122 {
2123   \_unravel_back_input:
2124   \_unravel_input_gpop_item:NF \l__unravel_head_tl
2125   {
2126     \msg_error:nn { unravel } { runaway-text }
2127     \_unravel_exit:w
2128   }
2129   \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2130   \bool_if:NT #1
2131     { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2132   \_unravel_prev_input_silent:V \l__unravel_head_tl
2133 }

```

(End definition for _unravel_scan_group_n:N.)

_unravel_scan_group_x:N

```

2134 \cs_new_protected:Npn \_unravel_scan_group_x:N #1
2135 {
2136   \_unravel_input_gpop_tl:N \l__unravel_head_tl
2137   \_unravel_back_input:V \l__unravel_head_tl
2138   \bool_if:NTF #1
2139   {
2140     \_unravel_prev_input_silent:V \c_left_brace_str
2141     \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2142     \_unravel_scan_group_xdef:n { 1 }
2143   }
2144   {
2145     \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl
2146     \_unravel_prev_input_gtl:N \l__unravel_head_gtl
2147     \_unravel_scan_group_x:n { 1 }
2148     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
2149     \_unravel_prev_input_silent:x
2150     { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2151   }
2152 }

```

(End definition for _unravel_scan_group_x:N.)

_unravel_scan_group_xdef:n

```

2153 \cs_new_protected:Npn \_unravel_scan_group_xdef:n #1
2154 {
2155   \_unravel_get_token_x:N \c_true_bool
2156   \tl_if_empty:NTF \l__unravel_head_tl

```

```

2157 {
2158   \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2159   {
2160     \__unravel_prev_input_silent:V \c_left_brace_str
2161     \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2162     \__unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2163   }
2164   {
2165     \__unravel_prev_input_silent:V \c_right_brace_str
2166     \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2167     \int_compare:nNnF {#1} = \c_one
2168     { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2169   }
2170 }
2171 {
2172   \__unravel_prev_input_silent:V \l__unravel_head_tl
2173   \tl_put_right:Nx \l__unravel_defining_tl
2174   { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2175   \__unravel_scan_group_xdef:n {#1}
2176 }
2177 }
2178 \cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }

```

(End definition for __unravel_scan_group_xdef:n.)

__unravel_scan_group_x:n

```

2179 \cs_new_protected:Npn \__unravel_scan_group_x:n #1
2180 {
2181   \__unravel_get_token_x:N \c_false_bool
2182   \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2183   \tl_if_empty:NTF \l__unravel_head_tl
2184   {
2185     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2186     { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2187     {
2188       \int_compare:nNnF {#1} = \c_one
2189       { \__unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2190     }
2191   }
2192   { \__unravel_scan_group_x:n {#1} }
2193 }
2194 \cs_generate_variant:Nn \__unravel_scan_group_x:n { f }

```

(End definition for __unravel_scan_group_x:n.)

__unravel_get_token_x:N

```

2195 \cs_new_protected:Npn \__unravel_get_token_x:N #1
2196 {
2197   \__unravel_get_next:
2198   \__unravel_token_if_protected:NF \l__unravel_head_token

```



```

2199     {
2200         \__unravel_set_cmd:
2201         \int_compare:nNnTF
2202             \l__unravel_head_cmd_int = { \__unravel_tex_use:n { the } }
2203         {
2204             \__unravel_get_the:
2205             \bool_if:NTF #1
2206             {
2207                 \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
2208                 \__unravel_prev_input:V \l__unravel_head_tl
2209             }
2210             {
2211                 \gtl_set:Nx \l__unravel_tmpb_gtl { \l__unravel_head_tl }
2212                 \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
2213                 \__unravel_print_action:
2214             }
2215         }
2216         { \__unravel_expand: }
2217         \__unravel_get_token_x:N #1
2218     }
2219 }

```

(End definition for __unravel_get_token_x:N.)

__unravel_scan_alt_rule:

```

2220 \cs_new_protected_nopar:Npn \__unravel_scan_alt_rule:
2221 {
2222     \__unravel_scan_keyword:nTF { wWiIdDtThH }
2223     {
2224         \__unravel_scan_normal_dimen:
2225         \__unravel_scan_alt_rule:
2226     }
2227     {
2228         \__unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2229         {
2230             \__unravel_scan_normal_dimen:
2231             \__unravel_scan_alt_rule:
2232         }
2233         {
2234             \__unravel_scan_keyword:nT { dDeEpPtThH }
2235             {
2236                 \__unravel_scan_normal_dimen:
2237                 \__unravel_scan_alt_rule:
2238             }
2239         }
2240     }
2241 }

```

(End definition for __unravel_scan_alt_rule:.)

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

```

2242 \cs_new_protected_nopar:Npn \__unravel_scan_spec:
2243 {
2244   \__unravel_scan_keyword:nTF { tT oO } { \__unravel_scan_normal_dimen: }
2245   {
2246     \__unravel_scan_keyword:nT { sS pP rR eE aA dD }
2247     { \__unravel_scan_normal_dimen: }
2248   }
2249   \__unravel_scan_left_brace:
2250 }

```

(End definition for `__unravel_scan_spec:`.)

2.8 Working with boxes

`__unravel_do_box:N` When this procedure is called, the last item in `\g__unravel_prev_input_seq` 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.

```

2251 \cs_new_protected:Npn \__unravel_do_box:N #1
2252 {
2253   \__unravel_get_x_non_relax:
2254   \__unravel_set_cmd:
2255   \int_compare:nNnTF
2256     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2257     { \__unravel_do_begin_box:N #1 }
2258     {
2259       \bool_if:NTF #1
2260       {
2261         \int_case:nnTF \l__unravel_head_cmd_int
2262         {
2263           { \__unravel_tex_use:n { hrule } } { }
2264           { \__unravel_tex_use:n { vrule } } { }
2265         }
2266         { \__unravel_do_leaders_rule: }
2267         { \__unravel_do_box_error: }
2268       }
2269       { \__unravel_do_box_error: }
2270     }
2271 }

```

(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 \g_unravel_prev_input_seq). For some reason (this appears to be what T_EX does), there is no need to remove the after assignment token here.

```

2272 \cs_new_protected_nopar:Npn \_unravel\_do\_box\_error:
2273 {
2274   \_unravel\_back\_input:
2275   \msg_error:nn { unravel } { missing-box }
2276   \seq_gpop_right:NN \g\_unravel\_prev\_input\_seq \l\_unravel\_head\_tl
2277   \_unravel\_print\_action:x { \tl\_to\_str:N \l\_unravel\_head\_tl }
2278 }

```

(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 \g_unravel_prev_input_seq. 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.

```

2279 \cs_new_protected:Npn \_unravel\_do\_begin\_box:N #1
2280 {
2281   \_unravel\_prev\_input:V \l\_unravel\_head\_tl
2282   \int_case:nnTF \l\_unravel\_head\_char\_int
2283   {
2284     { 0 } { \_unravel\_scan\_int: } % box
2285     { 1 } { \_unravel\_scan\_int: } % copy
2286     { 2 } { } % lastbox
2287     { 3 } % vsplit
2288     {
2289       \_unravel\_scan\_int:
2290       \_unravel\_scan\_keyword:nF { tT oO }
2291       {
2292         \msg_error:nn { unravel } { missing-to }
2293         \_unravel\_prev\_input:n { to }
2294       }
2295       \_unravel\_scan\_normal\_dimen:
2296     }
2297   }
2298   { \_unravel\_do\_simple\_box:N #1 }
2299   { \_unravel\_do\_box\_explicit:N #1 }
2300 }

```

(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.

```

2301 \cs_new_protected:Npn \_unravel\_do\_simple\_box:N #1
2302 {

```

```

2303 \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2304 {
2305   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2306   \tl_use:N \l__unravel_head_tl \scan_stop:
2307   \gtl_put_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2308   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2309 }
2310 }

```

(End definition for __unravel_do_simple_box:N.)

__unravel_do_leaders_fetch_skip:

```

2311 \cs_new_protected_nopar:Npn \__unravel_do_leaders_fetch_skip:
2312 {
2313   \__unravel_get_x_non_relax:
2314   \__unravel_set_cmd:
2315   \int_compare:nNnTF \l__unravel_head_cmd_int
2316   = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2317   {
2318     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2319     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2320     \__unravel_do_append_glue:
2321   }
2322   {
2323     \__unravel_back_input:
2324     \msg_error:nn { unravel } { improper-leaders }
2325     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2326     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2327   }
2328 }

```

(End definition for __unravel_do_leaders_fetch_skip:.)

__unravel_do_box_explicit:N

At this point, the last item in \g__unravel_prev_input_seq is typically \setbox0\hbox or \raise 3pt\hbox. Scan for keywords to and spread and a left brace. Install a hook in \everyhbox or \everyvbox (whichever T_EX is going to insert in the box). We then retrieve all the material that led to the current box into \l__unravel_head_tl in order to print it, then let T_EX perform the box operation (here we need to provide the begin-group token, as it was scanned but not placed in \g__unravel_prev_input_seq). T_EX inserts \everyhbox or \everyvbox just after the begin-group token, and the hook we did is such that all that material is collected and put into the input that we will study. We must remember to find a glue for leaders, and for this we use a stack of booleans: the top is true if the innermost box is part of leaders.

```

2329 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2330 {
2331   \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2332   { \__unravel_box_hook:N \tex_everyhbox:D }
2333   { \__unravel_box_hook:N \tex_everyvbox:D }
2334   % ^^A todo: TeX calls |normal_paragraph| here.

```

```

2335 \__unravel_scan_spec:
2336 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2337 \__unravel_set_action_text:x
2338 { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2339 \seq_push:Nx \l__unravel_leaders_box_seq
2340 { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2341 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2342 \gtl_gconcat:NNN \g__unravel_output_gtl
2343 \g__unravel_output_gtl \c_group_begin_gtl
2344 \tl_use:N \l__unravel_head_tl
2345 \c_group_begin_token \__unravel_box_hook_end:
2346 }

```

(End definition for __unravel_do_box_explicit:N.)

```

\__unravel_box_hook:N
\__unravel_box_hook:w
\__unravel_box_hook_end:
2347 \cs_new_protected:Npn \__unravel_box_hook:N #1
2348 {
2349   \tl_set:NV \l__unravel_tmpa_tl #1
2350   \str_if_eq_x:nnF
2351     { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2352     {
2353       \exp_args:Nx #1
2354       {
2355         \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2356         \exp_not:V #1
2357       }
2358     }
2359   \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2360   {
2361     \exp_args:No #1 {##1}
2362     \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2363     \__unravel_print_action:
2364     \__unravel_back_input:o {##1}
2365     \__unravel_set_action_text:x
2366     { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2367     \tl_if_empty:oF {##1} { \__unravel_print_action: }
2368   }
2369 }
2370 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2371 \cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:

```

(End definition for __unravel_box_hook:N.)

__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.

```

2372 \cs_new_protected_nopar:Npn \__unravel_do_leaders_rule:
2373 {
2374   \__unravel_prev_input:V \l__unravel_head_tl

```

```

2375 \__unravel_scan_alt_rule:
2376 \__unravel_do_leaders_fetch_skip:
2377 }

```

(End definition for __unravel_do_leaders_rule:.)

2.9 Paragraphs

__unravel_charcode_if_safe:nTF

```

2378 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2379 {
2380   \bool_if:nTF
2381   {
2382     \int_compare_p:n { #1 = '!' }
2383     || \int_compare_p:n { ' ' <= #1 <= '[' }
2384     || \int_compare_p:n { #1 = ']' }
2385     || \int_compare_p:n { ' ' <= #1 <= 'z' }
2386   }
2387   { \prg_return_true: }
2388   { \prg_return_false: }
2389 }

```

(End definition for __unravel_charcode_if_safe:nTF.)

__unravel_char:n
__unravel_char:V
__unravel_char:x

```

2390 \cs_new_protected:Npn \__unravel_char:n #1
2391 {
2392   \tex_char:D #1 \scan_stop:
2393   \__unravel_charcode_if_safe:nTF {#1}
2394   { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2395   {
2396     \tl_set:Nx \l__unravel_tmpa_tl
2397     { \exp_not:N \char \int_eval:n {#1} ~ }
2398   }
2399   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2400   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2401 }
2402 \cs_generate_variant:Nn \__unravel_char:n { V , x }

```

(End definition for __unravel_char:n, __unravel_char:V, and __unravel_char:x.)

__unravel_char_in_mmode:n
__unravel_char_in_mmode:V
__unravel_char_in_mmode:x

```

2403 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2404 {
2405   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2406   { % math active
2407     \gtl_set:Nx \l__unravel_head_gtl
2408     { \char_generate:nn {#1} { 12 } }
2409     \__unravel_back_input:
2410   }

```

```

2411     { \_unravel_char:n {#1} }
2412   }
2413   \cs_generate_variant:Nn \_unravel_char_in_mmode:n { V , x }

(End definition for \_unravel_char_in_mmode:n, \_unravel_char_in_mmode:V, and \_unravel-
char_in_mmode:x.)

```

```

\_unravel_mathchar:n
\_unravel_mathchar:x
2414 \cs_new_protected:Npn \_unravel_mathchar:n #1
2415   {
2416     \tex_mathchar:D #1 \scan_stop:
2417     \tl_set:Nx \l__unravel_tmpa_tl
2418       { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2419     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2420     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2421   }
2422   \cs_generate_variant:Nn \_unravel_mathchar:n { x }

(End definition for \_unravel_mathchar:n and \_unravel_mathchar:x.)

```

`_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.

```

2423 \cs_new_protected:Npn \_unravel_new_graf:N #1
2424   {
2425     \tl_set:NV \l__unravel_tmpa_tl \_unravel_everypar:w
2426     \__unravel_everypar:w { }
2427     \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2428     \exp_args:NV \_unravel_everypar:w \l__unravel_tmpa_tl
2429     \_unravel_back_input:V \l__unravel_tmpa_tl
2430     \__unravel_print_action:x
2431     {
2432       \g__unravel_action_text_str \c_space_tl : ~
2433       \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2434     }
2435   }

(End definition for \_unravel_new_graf:N.)

```

```

\_unravel_end_graf:
2436 \cs_new_protected_nopar:Npn \_unravel_end_graf:
2437   { \mode_if_horizontal:T { \_unravel_normal_paragraph: } }

(End definition for \_unravel_end_graf:.)

```

```

\_unravel_normal_paragraph:
2438 \cs_new_protected_nopar:Npn \_unravel_normal_paragraph:
2439   {
2440     \tex_par:D

```

```

2441 \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2442 \__unravel_print_action:x { Paragraph~end. }
2443 }

```

(End definition for __unravel_normal_paragraph:.)

__unravel_build_page:

```

2444 \cs_new_protected_nopar:Npn \__unravel_build_page:
2445 {
2446 }

```

(End definition for __unravel_build_page:.)

2.10 Groups

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

```

2447 \cs_new_protected_nopar:Npn \_unravel_handle_left_brace:
2448 {
2449   \int_case:nnF \_unravel_currentgroupstype:
2450   {
2451     { 1 } { \_unravel_end_simple_group: } % simple
2452     { 2 } { \_unravel_end_box_group: } % hbox
2453     { 3 } { \_unravel_end_box_group: } % adjusted_hbox
2454     { 4 } { \_unravel_end_graf: \_unravel_end_box_group: } % vbox
2455     { 5 } { \_unravel_end_graf: \_unravel_end_box_group: } % vtop
2456     { 6 } { \_unravel_end_align_group: } % align
2457     { 7 } { \_unravel_end_no_align_group: } % no_align
2458     { 8 } { \_unravel_end_output_group: } % output
2459     { 9 } { \_unravel_end_simple_group: } % math
2460     { 10 } { \_unravel_end_disc_group: } % disc
2461     { 11 } { \_unravel_end_graf: \_unravel_end_simple_group: } % insert
2462     { 12 } { \_unravel_end_graf: \_unravel_end_simple_group: } % vcenter
2463     { 13 } { \_unravel_end_math_choice_group: } % math_choice
2464   }
2465   { % bottom_level, semi_simple, math_shift, math_left
2466     \_unravel_back_input:
2467     \l__unravel_head_token
2468     \_unravel_print_action:
2469   }
2470 }

```

(End definition for _unravel_handle_left_brace:.)

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

```

2471 \cs_new_protected_nopar:Npn \_unravel_end_simple_group:
2472 {
2473   \l__unravel_head_token
2474   \gtl_gconcat:NNN \g__unravel_output_gtl
2475   \g__unravel_output_gtl \c_group_end_gtl

```



```

2476   \__unravel_print_action:
2477   }

(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.

```

2478 \cs_new_protected_nopar:Npn \__unravel_end_box_group:
2479 {
2480   \seq_pop:Nn \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2481   \str_if_eq_x:nnTF \l__unravel_tmpa_tl { Z }
2482   { \__unravel_end_simple_group: }
2483   {
2484     \__unravel_get_x_non_relax:
2485     \__unravel_set_cmd:
2486     \int_compare:nNnTF \l__unravel_head_cmd_int
2487       = { \__unravel_tex_use:n { \l__unravel_tmpa_tl skip } }
2488       {
2489         \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2490         \__unravel_do_append_glue:
2491       }
2492       {
2493         \__unravel_back_input:
2494         \c_group_end_token \group_begin: \group_end:
2495         \__unravel_print_action:
2496       }
2497   }
2498 }

```

(End definition for `__unravel_end_box_group:.`)

`__unravel_off_save:`

```

2499 \cs_new_protected_nopar:Npn \__unravel_off_save:
2500 {
2501   \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
2502   { % bottom-level
2503     \msg_error:nnx { unravel } { extra-close }
2504     { \token_to_meaning:N \l__unravel_head_token }
2505   }
2506   {
2507     \__unravel_back_input:
2508     \int_case:nnF \__unravel_currentgrouptype:
2509     {
2510       { 14 } % semi_simple_group
2511       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
2512       { 15 } % math_shift_group
2513       { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2514       { 16 } % math_left_group
2515       { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2516     }
2517   }

```

```

2517         { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2518     \__unravel_back_input:
2519     \msg_error:nxx { unravel } { off-save }
2520     { \gtl_to_str:N \l__unravel_head_gtl }
2521 }
2522 }

```

(End definition for __unravel_off_save:.)

2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
2523 \cs_new_protected:Npn \__unravel_mode_math:n #1
2524 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
2525 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
2526 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
2527 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
2528 {
2529     \mode_if_math:TF
2530     { \__unravel_insert_dollar_error: }
2531     { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
2532 }
2533 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
2534 {
2535     \mode_if_vertical:TF
2536     { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
2537     {#1}
2538 }

```

(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.

```

2539 \cs_new_protected_nopar:Npn \__unravel_head_for_vmode:
2540 {
2541     \mode_if_inner:TF
2542     {
2543         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hruler:D
2544         {
2545             \msg_error:nn { unravel } { hruler-bad-mode }
2546             \__unravel_print_action:
2547         }
2548         { \__unravel_off_save: }
2549     }
2550     {
2551         \__unravel_back_input:
2552         \gtl_set:Nn \l__unravel_head_gtl { \par }
2553         \__unravel_back_input:
2554     }
2555 }

```

(End definition for _unravel_head_for_vmode:.)

_unravel_goto_inner_math:

```

2556 \cs_new_protected_nopar:Npn \_unravel_goto_inner_math:
2557 {
2558   \_unravel_box_hook:N \tex_everymath:D
2559   $ % $
2560   \_unravel_box_hook_end:
2561 }

```

(End definition for _unravel_goto_inner_math:.)

_unravel_goto_display_math:

```

2562 \cs_new_protected_nopar:Npn \_unravel_goto_display_math:
2563 {
2564   \_unravel_box_hook:N \tex_everydisplay:D
2565   $ $
2566   \_unravel_box_hook_end:
2567 }

```

(End definition for _unravel_goto_display_math:.)

_unravel_after_math:

```

2568 \cs_new_protected_nopar:Npn \_unravel_after_math:
2569 {
2570   \mode_if_inner:TF
2571   {
2572     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2573     $ % $
2574   }
2575   {
2576     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2577     \_unravel_get_x_next:
2578     \token_if_eq_catcode:NNF
2579     \l__unravel_head_token \c_math_toggle_token
2580     {
2581       \_unravel_back_input:
2582       \tl_set:Nn \l__unravel_head_tl { $ } % $
2583       \msg_error:nn { unravel } { missing-dollar }
2584     }
2585     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2586     $ $
2587   }
2588   \_unravel_print_action:
2589 }

```

(End definition for _unravel_after_math:.)

2.12 One step

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

```
2590 \cs_new_protected_nopar:Npn \__unravel_do_step:
2591 {
2592   \__unravel_set_action_text:
2593   \bool_if:NT \g__unravel_internal_debug_bool
2594     { \iow_term:x { Cmd:~\int_use:N \l__unravel_head_cmd_int } }
2595   \cs_if_exist_use:cF
2596     { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2597     { \msg_error:nnx { unravel } { internal } { unknown-command } }
2598 }
```

(End definition for __unravel_do_step:.)

2.13 Commands

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

2.13.1 Characters: from 0 to 15

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

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

- `par_end=13` for `\par`.
- `stop=14` for `\end` and `\dump`.
- `delim_num=15` for `\delimiter`.

Not implemented at all: `endv`.

`\relax` does nothing.

```
2599 \__unravel_new_tex_cmd:nn { relax } % 0
2600 { \__unravel_print_action: }
```

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.

```
2601 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
2602 {
2603   \gtl_gconcat:NNN \g__unravel_output_gtl
2604   \g__unravel_output_gtl \c_group_begin_gtl
2605   \__unravel_print_action:
2606   \l__unravel_head_token
2607 }
2608 \__unravel_new_tex_cmd:nn { end-group_char } % 2
2609 { \__unravel_handle_left_brace: }
```

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

```
2610 \__unravel_new_tex_cmd:nn { math_char } % 3
2611 {
2612   \__unravel_mode_non_vertical:n
2613   {
2614     \mode_if_math:TF
2615     {
2616       \int_compare:nNnTF
2617       \__unravel_currentgrouptype: = { 15 } % math_shift_group
2618       { \__unravel_after_math: }
2619       { \__unravel_off_save: }
2620     }
2621     {
2622       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2623       \__unravel_get_next:
2624       \token_if_eq_catcode:NNTF
2625       \l__unravel_head_token \c_math_toggle_token
2626       {
2627         \mode_if_inner:TF
2628         { \__unravel_back_input: \__unravel_goto_inner_math: }
2629         {
2630           \gtl_gput_right:NV
2631           \g__unravel_output_gtl \l__unravel_head_tl
2632           \__unravel_goto_display_math:
2633         }
2634       }
2635       { \__unravel_back_input: \__unravel_goto_inner_math: }
```

```

2636     }
2637   }
2638 }

```

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.

```

2639 \__unravel_new_tex_cmd:nn { alignment_char } % 4
2640 { \l__unravel_head_token \__unravel_print_action: }
2641 \__unravel_new_tex_cmd:nn { car_ret } % 5
2642 { \l__unravel_head_token \__unravel_print_action: }
2643 \__unravel_new_tex_cmd:nn { macro_char } % 6
2644 { \l__unravel_head_token \__unravel_print_action: }
2645 \__unravel_new_tex_cmd:nn { superscript_char } % 7
2646 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2647 \__unravel_new_tex_cmd:nn { subscript_char } % 8
2648 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2649 \cs_new_protected_nopar:Npn \__unravel_sub_sup:
2650 {
2651   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2652   \__unravel_print_action:
2653   \__unravel_get_x_non_relax:
2654   \__unravel_set_cmd:
2655   \int_case:nnTF \l__unravel_head_cmd_int
2656   {
2657     { \__unravel_tex_use:n { the_char } }
2658     { \__unravel_prev_input:V \l__unravel_head_tl }
2659     { \__unravel_tex_use:n { other_char } }
2660     { \__unravel_prev_input:V \l__unravel_head_tl }
2661     { \__unravel_tex_use:n { char_given } }
2662     { \__unravel_prev_input:V \l__unravel_head_tl }
2663     { \__unravel_tex_use:n { char_num } }
2664     {
2665       \__unravel_prev_input:V \l__unravel_head_tl
2666       \__unravel_scan_int:
2667     }
2668     { \__unravel_tex_use:n { math_char_num } }
2669     {
2670       \__unravel_prev_input:V \l__unravel_head_tl
2671       \__unravel_scan_int:
2672     }
2673     { \__unravel_tex_use:n { math_given } }
2674     { \__unravel_prev_input:V \l__unravel_head_tl }
2675     { \__unravel_tex_use:n { delim_num } }
2676     { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
2677   }
2678   {
2679     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2680     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2681     \tl_use:N \l__unravel_head_tl \scan_stop:

```

```

2682     }
2683     {
2684         \__unravel_back_input:
2685         \__unravel_scan_left_brace:
2686         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2687         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2688         \gtl_gconcat:NNN \g__unravel_output_gtl
2689         \g__unravel_output_gtl \c_group_begin_gtl
2690         \tl_use:N \l__unravel_head_tl \c_group_begin_token
2691     }
2692     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2693 }
2694 \__unravel_new_tex_cmd:nn { endv } % 9
2695 { \msg_error:nn { unravel } { not-implemented } { 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.

```

2696 \__unravel_new_tex_cmd:nn { blank_char } % 10
2697 {
2698     \mode_if_horizontal:T
2699     {
2700         \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
2701         \l__unravel_head_token
2702     }
2703     \__unravel_print_action:
2704 }

```

Letters and other characters leave vertical mode.

```

2705 \__unravel_new_tex_cmd:nn { the_char } % 11
2706 {
2707     \__unravel_mode_non_vertical:n
2708     {
2709         \tl_set:Nx \l__unravel_tmpa_tl
2710         { ' \__unravel_token_to_char:N \l__unravel_head_token }
2711         \mode_if_math:TF
2712         { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
2713         { \__unravel_char:V \l__unravel_tmpa_tl }
2714     }
2715 }
2716 \__unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12
2717 \__unravel_new_tex_cmd:nn { par_end } % 13
2718 {
2719     \__unravel_mode_non_math:n
2720     {
2721         \mode_if_vertical:TF
2722         { \__unravel_normal_paragraph: }
2723         {
2724             % if align_state<0 then off_save;
2725             \__unravel_end_graf:

```

```

2726         \mode_if_vertical:T
2727         { \mode_if_inner:F { \__unravel_build_page: } }
2728     }
2729 }
2730 }
2731 \__unravel_new_tex_cmd:nn { stop } % 14
2732 {
2733     \__unravel_mode_vertical:n
2734     {
2735         \mode_if_inner:TF
2736         { \__unravel_forbidden_case: }
2737         {
2738             % ^^A todo: unless its_all_over
2739             \int_gdecr:N \g__unravel_ends_int
2740             \int_compare:nNnTF \g__unravel_ends_int > \c_zero
2741             {
2742                 \__unravel_back_input:
2743                 \__unravel_back_input:n
2744                 {
2745                     \__unravel_hbox:w to \tex_hsize:D { }
2746                     \tex_vfill:D
2747                     \tex_penalty:D - '10000000000 ~
2748                 }
2749                 \__unravel_build_page:
2750                 \__unravel_print_action:x { End~everything! }
2751             }
2752             {
2753                 \__unravel_print_outcome:
2754                 \l__unravel_head_token
2755             }
2756         }
2757     }
2758 }
2759 \__unravel_new_tex_cmd:nn { delim_num } % 15
2760 {
2761     \__unravel_mode_math:n
2762     {
2763         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2764         \__unravel_print_action:
2765         \__unravel_scan_int:
2766         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2767         \tl_use:N \l__unravel_head_tl \scan_stop:
2768         \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2769     }
2770 }

```

2.13.2 Boxes: from 16 to 31

- char_num=16 for \char

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

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

```

2771 \__unravel_new_tex_cmd:nn { char_num } % 16
2772 {
2773   \__unravel_mode_non_vertical:n
2774   {
2775     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2776     \__unravel_print_action:
2777     \__unravel_scan_int:
2778     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2779     \mode_if_math:TF
2780     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
2781     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
2782   }
2783 }
```

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.

```

2784 \__unravel_new_tex_cmd:nn { math_char_num } % 17
```

```

2785 {
2786   \__unravel_mode_math:n
2787   {
2788     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2789     \__unravel_print_action:
2790     \__unravel_scan_int:
2791     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2792     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
2793   }
2794 }

2795 \__unravel_new_tex_cmd:nn { mark } % 18
2796 {
2797   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2798   \__unravel_print_action:
2799   \int_compare:nNnF \l__unravel_head_char_int = \c_zero
2800   { \__unravel_scan_int: }
2801   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2802   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2803   \seq_gpop_right:Nn \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2804   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2805   \__unravel_print_action:x
2806   { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
2807   \tl_put_right:Nx \l__unravel_head_tl
2808   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2809   \tl_use:N \l__unravel_head_tl
2810 }

```

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 `\g__unravel_prev_input_seq` in general. Since no expansion can occur, simply grab the token and show it.

```

2811 \__unravel_new_tex_cmd:nn { xray } % 19
2812 {
2813   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2814   \__unravel_print_action:
2815   \int_case:nnF \l__unravel_head_char_int
2816   {
2817     { 0 }
2818     { % show
2819       \__unravel_get_next:
2820       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2821       \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl
2822     }
2823     { 2 }
2824     { % showthe
2825       \__unravel_get_x_next:
2826       \__unravel_scan_something_internal:n { 5 }

```

```

2827         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2828         \exp_args:Nx \etex_showtokens:D
2829         { \tl_tail:N \l__unravel_head_tl }
2830     }
2831 }
2832 { % no operand for showlists, showgroups, showifs
2833   \int_compare:nNnT \l__unravel_head_char_int = \c_one % showbox
2834   { \__unravel_scan_int: }
2835   \int_compare:nNnT \l__unravel_head_char_int = \c_five % showtokens
2836   { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
2837   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2838   \tl_use:N \l__unravel_head_tl \scan_stop:
2839 }
2840 }
2841 \__unravel_new_tex_cmd:nn { make_box } % 20
2842 {
2843   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2844   \__unravel_back_input:
2845   \__unravel_do_box:N \c_false_bool
2846 }

```

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

```

2847 \cs_new_protected_nopar:Npn \__unravel_do_move:
2848 {
2849   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2850   \__unravel_print_action:
2851   \__unravel_scan_normal_dimen:
2852   \__unravel_do_box:N \c_false_bool
2853 }

```

(End definition for __unravel_do_move:.)

hmove=21 for \moveright and \moveleft.

```

2854 \__unravel_new_tex_cmd:nn { hmove } % 21
2855 {
2856   \mode_if_vertical:TF
2857   { \__unravel_do_move: } { \__unravel_forbidden_case: }
2858 }

```

vmove=22 for \lower and \raise.

```

2859 \__unravel_new_tex_cmd:nn { vmove } % 22
2860 {
2861   \mode_if_vertical:TF
2862   { \__unravel_forbidden_case: } { \__unravel_do_move: }
2863 }

```

`__unravel_do_unpackage:`

```

2864 \cs_new_protected_nopar:Npn \__unravel_do_unpackage:
2865 {

```

```

2866 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2867 \__unravel_print_action:
2868 \__unravel_scan_int:
2869 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2870 \tl_use:N \l__unravel_head_tl \scan_stop:
2871 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2872 }

```

(End definition for __unravel_do_unpackage:.)

un_hbox=23 for \unhbox and \unhcopy.

```

2873 \__unravel_new_tex_cmd:nn { un_hbox } % 23
2874 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

```

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

```

2875 \__unravel_new_tex_cmd:nn { un_vbox } % 24
2876 {
2877   \__unravel_mode_vertical:n
2878   {
2879     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
2880     { \l__unravel_head_token \__unravel_print_action: }
2881     { \__unravel_do_unpackage: }
2882   }
2883 }

```

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

```

2884 \__unravel_new_tex_cmd:nn { remove_item } % 25
2885 { \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.

```

2886 \cs_new_protected_nopar:Npn \__unravel_do_append_glue:
2887 {
2888   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
2889   { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
2890   {
2891     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2892     \__unravel_print_action:
2893     \exp_args:Nf \__unravel_scan_glue:n
2894     { \int_eval:n { \l__unravel_head_char_int - 2 } }
2895     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2896     \tl_use:N \l__unravel_head_tl \scan_stop:
2897     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2898   }
2899 }

```

```

(End definition for \_unravel\_do\_append\_glue:.)
hskip=26 for \hfil, \hfill, \hss, \hfilneg, \hskip.
2900 \_unravel\_new\_tex\_cmd:nn { hskip } % 26
2901 { \_unravel\_mode\_non\_vertical:n { \_unravel\_do\_append\_glue: } }
vskip=27 for \vfil, \vfill, \vss, \vfilneg, \vskip.
2902 \_unravel\_new\_tex\_cmd:nn { vskip } % 27
2903 { \_unravel\_mode\_vertical:n { \_unravel\_do\_append\_glue: } }
mskip=28 for \mskip (5).
2904 \_unravel\_new\_tex\_cmd:nn { mskip } % 28
2905 { \_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.

```

2906 \cs\_new\_protected\_nopar:Npn \_unravel\_do\_append\_kern:
2907 {
2908   \seq\_gput\_right:NV \g\_unravel\_prev\_input\_seq \l\_unravel\_head\_tl
2909   \_unravel\_print\_action:
2910   \token\_if\_eq\_meaning:NNTF \l\_unravel\_head\_token \tex\_kern:D
2911     { \_unravel\_scan\_dimen:nN { 2 } \c\_false\_bool }
2912     { \_unravel\_scan\_dimen:nN { 3 } \c\_false\_bool }
2913   \seq\_gpop\_right:NN \g\_unravel\_prev\_input\_seq \l\_unravel\_head\_tl
2914   \tl\_use:N \l\_unravel\_head\_tl \scan\_stop:
2915   \_unravel\_print\_action:x { \tl\_to\_str:N \l\_unravel\_head\_tl }
2916 }

(End definition for \_unravel\_do\_append\_kern:.)
kern=29 for \kern (1).
2917 \_unravel\_new\_tex\_cmd:nn { kern } % 29
2918 { \_unravel\_do\_append\_kern: }
mkern=30 for \mkern (99).
2919 \_unravel\_new\_tex\_cmd:nn { mkern } % 30
2920 { \_unravel\_mode\_math:n { \_unravel\_do\_append\_kern: } }
leader\_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).
2921 \_unravel\_new\_tex\_cmd:nn { leader\_ship } % 31
2922 {
2923   \seq\_gput\_right:NV \g\_unravel\_prev\_input\_seq \l\_unravel\_head\_tl
2924   \_unravel\_print\_action:
2925   \_unravel\_do\_box:N \c\_true\_bool
2926 }

```

2.13.3 From 32 to 47

- halign=32
- valign=33
- no_align=34
- vrule=35
- hrule=36
- insert=37
- vadjust=38
- ignore_spaces=39
- after_assignment=40
- after_group=41
- break_penalty=42
- start_par=43
- ital_corr=44
- accent=45
- math_accent=46
- discretionary=47

```

2927 \__unravel_new_tex_cmd:nn { halign } % 32
2928 { \msg_fatal:nxx { unravel } { not-implemented } { halign } }
2929 \__unravel_new_tex_cmd:nn { valign } % 33
2930 { \msg_fatal:nxx { unravel } { not-implemented } { valign } }
2931 \__unravel_new_tex_cmd:nn { no_align } % 34
2932 { \msg_fatal:nxx { unravel } { not-implemented } { noalign } }

2933 \__unravel_new_tex_cmd:nn { vrule } % 35
2934 { \__unravel_mode_non_vertical:n { \__unravel_do_rule: } }
2935 \__unravel_new_tex_cmd:nn { hrule } % 36
2936 { \__unravel_mode_vertical:n { \__unravel_do_rule: } }
2937 \cs_new_protected_nopar:Npn \__unravel_do_rule:
2938 {
2939   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2940   \__unravel_print_action:
2941   \__unravel_scan_alt_rule:
2942   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2943   \tl_use:N \l__unravel_head_tl \scan_stop:
2944   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2945 }

```

```

2946 \__unravel_new_tex_cmd:nn { insert } % 37
2947 { \__unravel_begin_insert_or_adjust: }
2948 \__unravel_new_tex_cmd:nn { vadjust } % 38
2949 {
2950   \mode_if_vertical:TF
2951   { \__unravel_forbidden_case: } { \__unravel_begin_insert_or_adjust: }
2952 }
2953 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
2954 {
2955   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
2956   {
2957     \__unravel_get_x_non_blank:
2958     \__unravel_set_cmd:
2959     \__unravel_do_step:
2960   }
2961   { \msg_error:nn { unravel } { not-implemented } { pdfprimitive } }
2962 }
2963 \__unravel_new_tex_cmd:nn { after_assignment } % 40
2964 {
2965   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
2966   \__unravel_get_next:
2967   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
2968   \__unravel_print_action:x
2969   {
2970     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
2971     \gtl_to_str:N \l__unravel_head_gtl
2972   }
2973 }
2974 \__unravel_new_tex_cmd:nn { after_group } % 41
2975 { \msg_error:nxx { unravel } { not-implemented } { aftergroup } }
2976 See \__unravel_do_append_glue:.
2977 \__unravel_new_tex_cmd:nn { break_penalty } % 42
2978 {
2979   \seq_gput_right:N \g__unravel_prev_input_seq \l__unravel_head_tl
2980   \__unravel_print_action:
2981   \__unravel_scan_int:
2982   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2983   \tl_use:N \l__unravel_head_tl \scan_stop:
2984   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2985 }
2986 \__unravel_new_tex_cmd:nn { start_par } % 43
2987 {
2988   \mode_if_vertical:TF
2989   {
2990     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
2991     { \__unravel_new_graf:N \c_false_bool }
2992     { \__unravel_new_graf:N \c_true_bool }

```

```

2992     }
2993     {
2994         \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
2995         {
2996             \__unravel_hbox:w width \tex_parindent:D { }
2997             \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2998         }
2999         \__unravel_print_action:
3000     }
3001 }
3002 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3003 {
3004     \mode_if_vertical:TF { \__unravel_forbidden_case: }
3005     { \l__unravel_head_token \__unravel_print_action: }
3006 }

```

__unravel_do_accent:

```

3007 \cs_new_protected_nopar:Npn \__unravel_do_accent:
3008 {
3009     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3010     \__unravel_print_action:
3011     \__unravel_scan_int:
3012     \__unravel_do_assignments:
3013     \bool_if:nTF
3014     {
3015         \token_if_eq_catcode_p:NN
3016         \l__unravel_head_token \c_catcode_letter_token
3017         ||
3018         \token_if_eq_catcode_p:NN
3019         \l__unravel_head_token \c_catcode_other_token
3020         ||
3021         \int_compare_p:nNn
3022         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3023     }
3024     { \__unravel_prev_input:V \l__unravel_head_tl }
3025     {
3026         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3027         {
3028             \__unravel_prev_input:V \l__unravel_head_tl
3029             \__unravel_scan_int:
3030         }
3031         { \__unravel_break:w }
3032     }
3033     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3034     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3035     \tl_use:N \l__unravel_head_tl \scan_stop:
3036     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3037     \__unravel_break_point:
3038 }

```


(End definition for `__unravel_do_accent:`)

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

```

3039 \cs_new_protected_nopar:Npn \__unravel_do_math_accent:
3040 {
3041   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3042   \__unravel_print_action:
3043   \__unravel_scan_int:
3044   \__unravel_scan_math:
3045   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3046   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3047   \tl_use:N \l__unravel_head_tl \scan_stop:
3048   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3049 }

```

(End definition for `__unravel_do_math_accent:`)

```

3050 \__unravel_new_tex_cmd:nn { accent } % 45
3051 {
3052   \__unravel_mode_non_vertical:n
3053   {
3054     \mode_if_math:TF
3055     { \__unravel_do_math_accent: } { \__unravel_do_accent: }
3056   }
3057 }
3058 \__unravel_new_tex_cmd:nn { math_accent } % 46
3059 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
3060 \__unravel_new_tex_cmd:nn { discretionary } % 47
3061 { \msg_error:nxx { unravel } { not-implemented } { discretionary } }

```

2.13.4 Maths: from 48 to 56

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

```

3062 \_unravel_new_tex_cmd:nn { eq_no } % 48
3063 { \msg_error:nxx { unravel } { not-implemented } { eqno } }
3064 \_unravel_new_tex_cmd:nn { left_right } % 49
3065 { \msg_error:nxx { unravel } { not-implemented } { left/right } }
3066 \_unravel_new_tex_cmd:nn { math_comp } % 50
3067 { \msg_error:nxx { unravel } { not-implemented } { math-comp } }
3068 \_unravel_new_tex_cmd:nn { limit_switch } % 51
3069 { \msg_error:nxx { unravel } { not-implemented } { limits } }
3070 \_unravel_new_tex_cmd:nn { above } % 52
3071 { \msg_error:nxx { unravel } { not-implemented } { above } }
3072 \_unravel_new_tex_cmd:nn { math_style } % 53
3073 { \msg_error:nxx { unravel } { not-implemented } { math-style } }
3074 \_unravel_new_tex_cmd:nn { math_choice } % 54
3075 { \msg_error:nxx { unravel } { not-implemented } { math-choice } }
3076 \_unravel_new_tex_cmd:nn { non_script } % 55
3077 { \msg_error:nxx { unravel } { not-implemented } { non-script } }
3078 \_unravel_new_tex_cmd:nn { vcenter } % 56
3079 { \msg_error:nxx { unravel } { not-implemented } { vcenter } }

```

2.13.5 From 57 to 70

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

```

3080 \__unravel_new_tex_cmd:nn { case_shift } % 57
3081 {
3082   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3083   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3084   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3085   \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
3086 }
3087 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
3088 {
3089   #1 { \__unravel_back_input:n {#2} }
3090   \__unravel_print_action:x
3091   { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3092 }
3093 \__unravel_new_tex_cmd:nn { message } % 58
3094 {
3095   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3096   \__unravel_print_action:
3097   \__unravel_scan_toks_to_str:
3098   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3099   \tl_use:N \l__unravel_head_tl
3100   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3101 }

```

Extensions are implemented in a later section.

```

3102 \__unravel_new_tex_cmd:nn { extension } % 59
3103 {
3104   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3105   \__unravel_print_action:
3106   \__unravel_scan_extension_operands:
3107   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3108   \tl_use:N \l__unravel_head_tl \scan_stop:
3109   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3110 }
3111 \__unravel_new_tex_cmd:nn { in_stream } % 60
3112 {
3113   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3114   \__unravel_print_action:
3115   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3116   {
3117     \__unravel_scan_int:
3118     \__unravel_scan_optional_equals:
3119     \__unravel_scan_file_name:
3120   }
3121   { \__unravel_scan_int: }
3122   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3123   \tl_use:N \l__unravel_head_tl \scan_stop:
3124   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3125 }
3126 \__unravel_new_tex_cmd:nn { begin_group } % 61

```

```

3127 {
3128   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3129   \__unravel_print_action:
3130   \l__unravel_head_token
3131 }
3132 \__unravel_new_tex_cmd:nn { end_group } % 62
3133 {
3134   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3135   \__unravel_print_action:
3136   \l__unravel_head_token
3137 }
3138 \__unravel_new_tex_cmd:nn { omit } % 63
3139 { \msg_error:nn { unravel } { not-implemented } { omit } }
3140 \__unravel_new_tex_cmd:nn { ex_space } % 64
3141 {
3142   \__unravel_mode_non_vertical:n
3143   { \l__unravel_head_token \__unravel_print_action: }
3144 }
3145 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3146 {
3147   \__unravel_mode_non_vertical:n
3148   { \l__unravel_head_token \__unravel_print_action: }
3149 }
3150 \__unravel_new_tex_cmd:nn { radical } % 66
3151 {
3152   \__unravel_mode_math:n
3153   {
3154     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3155     \__unravel_print_action:
3156     \__unravel_scan_int:
3157     \__unravel_scan_math:
3158     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3159     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3160     \tl_use:N \l__unravel_head_tl \scan_stop:
3161     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3162   }
3163 }

```

Let \TeX cause the error.

```

3164 \__unravel_new_tex_cmd:nn { end_cs_name } % 67
3165 { \l__unravel_head_token \__unravel_print_action: }

```

See `the_char` and `other_char`.

```

3166 \__unravel_new_tex_cmd:nn { char_given } % 68
3167 {
3168   \__unravel_mode_non_vertical:n
3169   {
3170     \mode_if_math:TF
3171     { \__unravel_char_in_mmode:V \l__unravel_head_char_int }

```

```

3172         { \_unravel_char:V \l\_unravel_head_char_int }
3173     }
3174 }
    See math_char_num.
3175 \_unravel_new_tex_cmd:nn { math_given } % 69
3176 {
3177     \_unravel_mode_math:n
3178     { \_unravel_mathchar:x { \int_use:N \l\_unravel_head_char_int } }
3179 }
3180 \_unravel_new_tex_cmd:nn { last_item } % 70
3181 { \_unravel_forbidden_case: }

```

2.13.6 Extensions

_unravel_scan_extension_operands:

```

3182 \cs_new_protected_nopar:Npn \_unravel_scan_extension_operands:
3183 {
3184     \int_case:nnF \l\_unravel_head_char_int
3185     {
3186         { 0 } % openout
3187         {
3188             \_unravel_scan_int:
3189             \_unravel_scan_optional_equals:
3190             \_unravel_scan_file_name:
3191         }
3192         { 1 } % write
3193         {
3194             \_unravel_scan_int:
3195             \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3196         }
3197         { 2 } % closeout
3198         { \_unravel_scan_int: }
3199         { 3 } % special
3200         { \_unravel_scan_toks_to_str: }
3201         { 4 } % immediate
3202         { \_unravel_scan_immediate_operands: }
3203         { 5 } % setlanguage
3204         {
3205             \mode_if_horizontal:TF
3206             { \_unravel_scan_int: }
3207             { \msg_error:nn { unravel } { invalid-mode } }
3208         }
3209         { 6 } % pdfliteral
3210         {
3211             \_unravel_scan_keyword:nF { dD iI rR eE cC tT }
3212             { \_unravel_scan_keyword:n { pP aA gG eE } }
3213             \_unravel_scan_pdf_ext_toks:
3214         }

```

```

3215 { 7 } % pdfobj
3216 {
3217   \_unravel_scan_keyword:nTF
3218   { rR eE sS eE rR vV eE oO bB jJ nN uU mM }
3219   { \_unravel_skip_optional_space: }
3220   {
3221     \_unravel_scan_keyword:nF { uU sS eE oO bB jJ nN uU mM }
3222     { \_unravel_scan_int: }
3223     \_unravel_scan_keyword:nT { sS tT rR eE aA mM }
3224     {
3225       \_unravel_scan_keyword:nT { aA tT tT rR }
3226       { \_unravel_scan_pdf_ext_toks: }
3227     }
3228     \_unravel_scan_keyword:n { fF iI lL eE }
3229     \_unravel_scan_pdf_ext_toks:
3230   }
3231 }
3232 { 8 } % pdfrefobj
3233 { \_unravel_scan_int: }
3234 { 9 } % pdfxform
3235 {
3236   \_unravel_scan_keyword:nT { aA tT tT rR }
3237   { \_unravel_scan_pdf_ext_toks: }
3238   \_unravel_scan_keyword:nTF { rR eE sS oO uU rR cC eE sS }
3239   { \_unravel_scan_pdf_ext_toks: }
3240   \_unravel_scan_int:
3241 }
3242 { 10 } % pdfrefxform
3243 { \_unravel_scan_int: }
3244 { 11 } % pdfximage
3245 { \_unravel_scan_image: }
3246 { 12 } % pdfrefximage
3247 { \_unravel_scan_int: }
3248 { 13 } % pdfannot
3249 {
3250   \_unravel_scan_keyword:nTF
3251   { rR eE sS eE rR vV eE oO bB jJ nN uU mM }
3252   { \_unravel_scan_optional_space: }
3253   {
3254     \_unravel_scan_keyword:nT { uU sS eE oO bB jJ nN uU mM }
3255     { \_unravel_scan_int: }
3256     \_unravel_scan_alt_rule:
3257     \_unravel_scan_pdf_ext_toks:
3258   }
3259 }
3260 { 14 } % pdfstartlink
3261 {
3262   \mode_if_vertical:TF
3263   { \msg_error:nn { unravel } { invalid-mode } }
3264   {

```

```

3265         \_unravel\_scan\_rule\_attr:
3266         \_unravel\_scan\_action:
3267     }
3268 }
3269 { 15 } % pdfendlink
3270 {
3271     \mode\_if\_vertical:T
3272     { \msg\_error:nn { unravel } { invalid-mode } }
3273 }
3274 { 16 } % pdfoutline
3275 {
3276     \_unravel\_scan\_keyword:nT { aA tT tT rR }
3277     { \_unravel\_scan\_pdf\_ext\_toks: }
3278     \_unravel\_scan\_action:
3279     \_unravel\_scan\_keyword:nT { cC oO uU nN tT }
3280     { \_unravel\_scan\_int: }
3281     \_unravel\_scan\_pdf\_ext\_toks:
3282 }
3283 { 17 } % pdfdest
3284 { \_unravel\_scan\_pdfdest\_operands: }
3285 { 18 } % pdfthread
3286 { \_unravel\_scan\_rule\_attr: \_unravel\_scan\_thread\_id: }
3287 { 19 } % pdfstartthread
3288 { \_unravel\_scan\_rule\_attr: \_unravel\_scan\_thread\_id: }
3289 { 20 } % pdfendthread
3290 { }
3291 { 21 } % pdfsavepos
3292 { }
3293 { 22 } % pdfinfo
3294 { \_unravel\_scan\_pdf\_ext\_toks: }
3295 { 23 } % pdfcatalog
3296 {
3297     \_unravel\_scan\_pdf\_ext\_toks:
3298     \_unravel\_scan\_keyword:n { oO pP eE nN aA cC tT iI oO nN }
3299     { \_unravel\_scan\_action: }
3300 }
3301 { 24 } % pdfnames
3302 { \_unravel\_scan\_pdf\_ext\_toks: }
3303 { 25 } % pdffontattr
3304 {
3305     \_unravel\_scan\_font\_ident:
3306     \_unravel\_scan\_pdf\_ext\_toks:
3307 }
3308 { 26 } % pdfincludechars
3309 {
3310     \_unravel\_scan\_font\_ident:
3311     \_unravel\_scan\_pdf\_ext\_toks:
3312 }
3313 { 27 } % pdfmapfile
3314 { \_unravel\_scan\_pdf\_ext\_toks: }

```

```

3315 { 28 } % pdfmapline
3316 { \_unravel_scan_pdf_ext_toks: }
3317 { 29 } % pdftrailer
3318 { \_unravel_scan_pdf_ext_toks: }
3319 { 30 } % pdfresettimer
3320 { }
3321 { 31 } % pdffontexpand
3322 {
3323     \_unravel_scan_font_ident:
3324     \_unravel_scan_optional_equals:
3325     \_unravel_scan_int:
3326     \_unravel_scan_int:
3327     \_unravel_scan_int:
3328     \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3329     { \_unravel_skip_optional_space: }
3330 }
3331 { 32 } % pdfsetrandomseed
3332 { \_unravel_scan_int: }
3333 { 33 } % pdfsnaprefpoint
3334 { }
3335 { 34 } % pdfsnappy
3336 { \_unravel_scan_normal_glue: }
3337 { 35 } % pdfsnappycomp
3338 { \_unravel_scan_int: }
3339 { 36 } % pdfglyphtounicode
3340 {
3341     \_unravel_scan_pdf_ext_toks:
3342     \_unravel_scan_pdf_ext_toks:
3343 }
3344 { 37 } % pdfcolorstack
3345 { \_unravel_scan_pdfcolorstack_operands: }
3346 { 38 } % pdfsetmatrix
3347 { \_unravel_scan_pdf_ext_toks: }
3348 { 39 } % pdfsave
3349 { }
3350 { 40 } % pdfrestore
3351 { }
3352 { 41 } % pdfnobluiltintounicode
3353 { \_unravel_scan_font_ident: }
3354 }
3355 { } % no other cases.
3356 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```

3357 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstack_operands:
3358 {
3359     \_unravel_scan_int:
3360     \_unravel_scan_keyword:nF { sSeEtT }

```



```

3361 {
3362   \_unravel\_scan\_keyword:nF { pPuUsShH }
3363   {
3364     \_unravel\_scan\_keyword:nF { pPoOpP }
3365     {
3366       \_unravel\_scan\_keyword:nF { cCuUrRrReEnNtT }
3367       {
3368         \msg\_error:nn { unravel }
3369         { color-stack-action-missing }
3370       }
3371     }
3372   }
3373 }
3374 }

```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```

3375 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_rule\_attr:
3376 {
3377   \_unravel\_scan\_alt\_rule:
3378   \_unravel\_scan\_keyword:nT { aA tT tT rR }
3379   { \_unravel\_scan\_pdf\_ext\_toks: }
3380 }

```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```

3381 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_action:
3382 {
3383   \_unravel\_scan\_keyword:nTF { uUsSeErR }
3384   { \_unravel\_scan\_pdf\_ext\_toks: }
3385   {
3386     \_unravel\_scan\_keyword:nF { gGoOtToO }
3387     {
3388       \_unravel\_scan\_keyword:nF { tThHrReEaAdD }
3389       { \msg\_error:nn { unravel } { action-type-missing } }
3390     }
3391   }
3392   \_unravel\_scan\_keyword:nT { fFiIlLeE }
3393   { \_unravel\_scan\_pdf\_ext\_toks: }
3394   \_unravel\_scan\_keyword:nTF { pPaAgGeE }
3395   {
3396     \_unravel\_scan\_int:
3397     \_unravel\_scan\_pdf\_ext\_toks:
3398   }
3399   {
3400     \_unravel\_scan\_keyword:nTF { nNaAmMeE }
3401     { \_unravel\_scan\_pdf\_ext\_toks: }
3402     {

```

```

3403         \_unravel_scan_keyword:nTF { nNuUmM }
3404         { \_unravel_scan_int: }
3405         { \msg_error:nn { unravel } { identifier-type-missing } }
3406     }
3407 }
3408 \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3409 { \_unravel_skip_optional_space: }
3410 {
3411     \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3412     { \_unravel_skip_optional_space: }
3413 }
3414 }

```

(End definition for _unravel_scan_action:.)

_unravel_scan_image: Used by \pdfximage.

```

3415 \cs_new_protected_nopar:Npn \_unravel_scan_image:
3416 {
3417     \_unravel_scan_rule_attr:
3418     \_unravel_scan_keyword:nTF { nNaAmMeEdD }
3419     { \_unravel_scan_pdf_ext_toks: }
3420     {
3421         \_unravel_scan_keyword:nT { pPaAgGeE }
3422         { \_unravel_scan_int: }
3423     }
3424     \_unravel_scan_keyword:nT { cCoOlLoOrRsSpPaAcCeE }
3425     { \_unravel_scan_int: }
3426     \_unravel_scan_pdf_ext_toks:
3427 }

```

(End definition for _unravel_scan_image:.)

_unravel_scan_immediate_operands:

```

3428 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands:
3429 {
3430     \_unravel_get_x_next:
3431     \_unravel_set_cmd:
3432     \int_compare:nNnTF
3433     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { extension } }
3434     {
3435         \int_compare:nNnTF
3436         \l__unravel_head_char_int < { 3 } % openout, write, closeout
3437         { \_unravel_scan_immediate_operands_aux: }
3438         {
3439             \int_case:nnF \l__unravel_head_char_int
3440             {
3441                 { 7 } { \_unravel_scan_extension_operands_aux: } % pdfobj
3442                 { 9 } { \_unravel_scan_extension_operands_aux: } % pdfxform
3443                 { 11 } { \_unravel_scan_extension_operands_aux: } %pdfximage
3444             }
3445         }
3446     }
3447 }

```

```

3445         { \_unravel_scan_immediate_operands_bad: }
3446     }
3447 }
3448 { \_unravel_scan_immediate_operands_bad: }
3449 }
3450 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_aux:
3451 {
3452     \_unravel_prev_input:V \l_unravel_head_tl
3453     \_unravel_scan_extension_operands:
3454 }
3455 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_bad:
3456 {
3457     \_unravel_back_input:
3458     \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_head_tl
3459     \_unravel_print_action:x { \tl_to_str:N \l_unravel_head_tl ignored }
3460     \seq_gput_right:Nn \g_unravel_prev_input_seq { }
3461 }
3462

```

(End definition for _unravel_scan_immediate_operands:.)

_unravel_scan_pdfdest_operands:

```

3463 \cs_new_protected_nopar:Npn \_unravel_scan_pdfdest_operands:
3464 {
3465     \_unravel_scan_keyword:nTF { nNuUmM }
3466     { \_unravel_scan_int: }
3467     {
3468         \_unravel_scan_keyword:nTF { nNaAmMeE }
3469         { \_unravel_scan_pdf_ext_toks: }
3470         { \msg_error:nn { unravel } { identifier-type-missing } }
3471     }
3472     \_unravel_scan_keyword:nTF { xXyYzZ }
3473     {
3474         \_unravel_scan_keyword:nT { zZoOoOmM }
3475         { \_unravel_scan_int: }
3476     }
3477     {
3478         \_unravel_scan_keyword:nF { fFiItTbBhH }
3479         {
3480             \_unravel_scan_keyword:nF { fFiItTbBvV }
3481             {
3482                 \_unravel_scan_keyword:nF { fFiItTbB }
3483                 {
3484                     \_unravel_scan_keyword:nF { fFiItThHhH }
3485                     {
3486                         \_unravel_scan_keyword:nF { fFiItTvV }
3487                         {
3488                             \_unravel_scan_keyword:nTF
3489                             { fFiItTrR }
3490                             {

```

```

3491         \_unravel_skip_optional_space:
3492         \_unravel_scan_alt_rule:
3493         \use_none:n
3494     }
3495     {
3496         \_unravel_scan_keyword:nF
3497         { fFiItT }
3498         {
3499             \msg_error:nn { unravel }
3500             {
3501                 destination-type-missing
3502             }
3503         }
3504     }
3505 }
3506 }
3507 }
3508 }
3509 }
3510 }
3511 \_unravel_skip_optional_space:
3512 }

```

(End definition for _unravel_scan_pdfdest_operands:.)

2.13.7 Assignments

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

```

3513 \cs_set_protected_nopar:Npn \_unravel_tmp:w
3514 {
3515     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3516     \_unravel_prefixed_command:
3517 }
3518 \int_step_inline:nnnn
3519 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
3520 { 1 }
3521 { \_unravel_tex_use:n { max_command } }
3522 { \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 `\g__unravel_prev_input_seq`.

```

3523 \cs_new_protected_nopar:Npn \_unravel_prefixed_command:
3524 {
3525     \int_while_do:nNnn
3526         \l__unravel_head_cmd_int = { \_unravel_tex_use:n { prefix } }

```

```

3527 {
3528   \__unravel_prev_input:V \l__unravel_head_tl
3529   \__unravel_get_x_non_relax:
3530   \__unravel_set_cmd:
3531   \int_compare:nNnF \l__unravel_head_cmd_int
3532   > { \__unravel_tex_use:n { max_non_prefixed_command } }
3533   {
3534     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3535     \msg_error:nnxx { unravel } { erroneous-prefixes }
3536     { \tl_to_str:N \l__unravel_tmpa_tl }
3537     { \tl_to_str:N \l__unravel_head_tl }
3538     \__unravel_back_input:
3539     \__unravel_omit_after_assignment:w
3540   }
3541 }
3542 % ^^A todo: Discard non-\global prefixes if they are irrelevant
3543 % ^^A todo: Adjust for the setting of \globaldefs
3544 \cs_if_exist_use:cF
3545 { \__unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3546 {
3547   \msg_error:nnx { unravel } { internal } { prefixed }
3548   \__unravel_omit_after_assignment:w
3549 }
3550 \__unravel_after_assignment:
3551 }

```

(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
3552 \cs_new_protected_nopar:Npn \__unravel_after_assignment:
3553 {
3554   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3555   \gtl_gclear:N \g__unravel_after_assignment_gtl
3556 }
3557 \cs_new_protected_nopar:Npn \__unravel_omit_after_assignment:w
3558 #1 \__unravel_after_assignment: { }

```

(End definition for __unravel_after_assignment:.)

__unravel_prefixed_new:nn

```

3559 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3560 {
3561   \cs_new_protected_nopar:cpn
3562   { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3563 }

```

(End definition for __unravel_prefixed_new:nn.)

_unravel_assign_token:n

```
3564 \cs_new_protected:Npn \_unravel_assign_token:n #1
3565 {
3566   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3567   #1
3568   \tl_use:N \l__unravel_head_tl \scan_stop:
3569   \_unravel_print_assigned_token:
3570 }
```

(End definition for _unravel_assign_token:n.)

_unravel_assign_register:

```
3571 \cs_new_protected_nopar:Npn \_unravel_assign_register:
3572 {
3573   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3574   \tl_use:N \l__unravel_head_tl \scan_stop:
3575   \_unravel_print_assigned_register:
3576 }
```

(End definition for _unravel_assign_register:.)

_unravel_assign_value:nn

```
3577 \cs_new_protected:Npn \_unravel_assign_value:nn #1#2
3578 {
3579   \tl_if_empty:nF {#1}
3580   {
3581     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3582     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3583     #1
3584     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3585   }
3586   \_unravel_prev_input:V \l__unravel_head_tl
3587   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3588   \_unravel_scan_optional_equals:
3589   #2
3590   \_unravel_assign_register:
3591 }
```

(End definition for _unravel_assign_value:nn.)

_unravel_assign_toks:

```
3592 \_unravel_prefixed_new:nn { toks_register } % 71
3593 {
3594   \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3595   { % \toks
3596     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3597     \_unravel_print_action:
3598     \_unravel_scan_int:
3599     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3600   }
```

```

3601   \__unravel_assign_toks:
3602   }
3603   \__unravel_prefixed_new:nn { assign_toks } % 72
3604   { \__unravel_assign_toks: }
3605   \cs_new_protected_nopar:Npn \__unravel_assign_toks:
3606   {
3607     \__unravel_prev_input_silent:V \l__unravel_head_tl
3608     \__unravel_print_action:
3609     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3610     \__unravel_scan_optional_equals:
3611     \__unravel_get_x_non_relax:
3612     \__unravel_set_cmd:
3613     \int_compare:nNnTF
3614       \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
3615       {
3616         \__unravel_prev_input:V \l__unravel_head_tl
3617         \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3618         { \__unravel_scan_int: }
3619       }
3620       {
3621         \int_compare:nNnTF
3622           \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
3623           { \__unravel_prev_input:V \l__unravel_head_tl }
3624           {
3625             \__unravel_back_input:
3626             \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3627           }
3628       }
3629   \__unravel_assign_register:
3630   }

(End definition for \__unravel_assign_toks:.)

3631   \__unravel_prefixed_new:nn { assign_int } % 73
3632   { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3633   \__unravel_prefixed_new:nn { assign_dimen } % 74
3634   { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3635   \__unravel_prefixed_new:nn { assign_glue } % 75
3636   { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
3637   \__unravel_prefixed_new:nn { assign_mu_glue } % 76
3638   { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
3639   \__unravel_prefixed_new:nn { assign_font_dimen } % 77
3640   {
3641     \__unravel_assign_value:nn
3642     { \__unravel_scan_int: \__unravel_scan_font_ident: }
3643     { \__unravel_scan_normal_dimen: }
3644   }
3645   \__unravel_prefixed_new:nn { assign_font_int } % 78
3646   {
3647     \__unravel_assign_value:nn
3648     { \__unravel_scan_font_int: } { \__unravel_scan_int: }

```

```

3649 }
3650 \__unravel_prefixed_new:nn { set_aux } % 79
3651 { % prevdepth = 1, spacefactor = 102
3652   \int_compare:nNnTF \l__unravel_head_char_int = \c_one
3653     { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3654     { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3655   }
3656 \__unravel_prefixed_new:nn { set_prev_graf } % 80
3657 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3658 \__unravel_prefixed_new:nn { set_page_dimen } % 81
3659 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3660 \__unravel_prefixed_new:nn { set_page_int } % 82
3661 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3662 \__unravel_prefixed_new:nn { set_box_dimen } % 83
3663 {
3664   \__unravel_assign_value:nn
3665   { \__unravel_scan_int: } { \__unravel_scan_normal_dimen: }
3666 }
3667 \__unravel_prefixed_new:nn { set_shape } % 84
3668 {
3669   \__unravel_assign_value:nn { \__unravel_scan_int: }
3670   {
3671     \prg_replicate:nn
3672     {
3673       \tl_if_head_eq_meaning:VNT
3674       \l__unravel_defined_tl \tex_parshape:D { \c_two * }
3675       \tl_tail:N \l__unravel_defined_tl
3676     }
3677     { \__unravel_scan_int: }
3678   }
3679 }
3680 \__unravel_prefixed_new:nn { def_code } % 85
3681 {
3682   \__unravel_assign_value:nn
3683   { \__unravel_scan_int: } { \__unravel_scan_int: }
3684 }
3685 \__unravel_prefixed_new:nn { def_family } % 86
3686 {
3687   \__unravel_assign_value:nn
3688   { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
3689 }
3690 \__unravel_prefixed_new:nn { set_font } % 87
3691 {
3692   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3693   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3694   \tl_use:N \l__unravel_head_tl \scan_stop:
3695   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3696   \__unravel_print_action:
3697 }

```



```

3698 \__unravel_prefixed_new:nn { def_font } % 88
3699 {
3700   \__unravel_prev_input_silent:V \l__unravel_head_tl
3701   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3702   \__unravel_scan_r_token:
3703   \__unravel_print_action:x
3704   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
3705   \__unravel_scan_optional_equals:
3706   \__unravel_scan_file_name:
3707   \bool_gset_true:N \g__unravel_name_in_progress_bool
3708   \__unravel_scan_keyword:nTF { aAtT }
3709   { \__unravel_scan_normal_dimen: }
3710   {
3711     \__unravel_scan_keyword:nT { sS cC aA lL eE dD }
3712     { \__unravel_scan_int: }
3713   }
3714   \bool_gset_false:N \g__unravel_name_in_progress_bool
3715   \__unravel_assign_token:n { }
3716 }

```

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

let, futurelet

```

3717 \__unravel_prefixed_new:nn { let } % 94
3718 {
3719   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3720   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
3721   { % |let|
3722     \__unravel_scan_r_token:
3723     \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3724     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3725     \__unravel_get_next:
3726     \bool_while_do:nn
3727     { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
3728     { \__unravel_get_next: }
3729     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
3730     { \__unravel_get_next: }
3731     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
3732     { \__unravel_get_next: }
3733   }
3734   { % |futurelet|
3735     \__unravel_scan_r_token:
3736     \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3737     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3738     \__unravel_get_next:
3739     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
3740     \__unravel_get_next:
3741     \__unravel_back_input:
3742     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
3743     \__unravel_back_input:

```

```

3744     }
3745     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3746     \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
3747     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3748     \use:x
3749     {
3750         \exp_not:V \l__unravel_head_tl
3751         \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
3752     }
3753     \__unravel_print_assigned_token:
3754 }
3755 \__unravel_prefixed_new:nn { shorthand_def } % 95
3756 {
3757     \__unravel_prev_input_silent:V \l__unravel_head_tl
3758     \tl_set:Nx \l__unravel_prev_action_tl
3759     { \tl_to_str:N \l__unravel_head_tl }
3760     \__unravel_scan_r_token:
3761     \__unravel_print_action:x
3762     { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
3763     \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
3764     \__unravel_scan_optional_equals:
3765     \__unravel_scan_int:
3766     \__unravel_assign_token:n { }
3767 }
3768 \__unravel_prefixed_new:nn { read_to_cs } % 96
3769 {
3770     \__unravel_prev_input:V \l__unravel_head_tl
3771     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3772     \__unravel_scan_int:
3773     \__unravel_scan_keyword:nF { tTo0 }
3774     {
3775         \msg_error:nn { unravel } { missing-to }
3776         \__unravel_prev_input:n { to }
3777     }
3778     \__unravel_scan_r_token:
3779     \__unravel_assign_token:n { }
3780 }
3781 \__unravel_prefixed_new:nn { def } % 97
3782 {
3783     \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3784     \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
3785     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
3786     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3787     \int_compare:nNnTF \l__unravel_head_char_int < \c_two
3788     { % def/gdef
3789         \__unravel_scan_r_token:
3790         \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
3791         \__unravel_scan_toks:NN \c_true_bool \c_false_bool
3792     }

```

```

3793 { % edef/xdef
3794   \__unravel_scan_r_token:
3795     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
3796     \__unravel_scan_toks:NN \c_true_bool \c_true_bool
3797   }
3798   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3799   \__unravel_prev_input:V \l__unravel_head_tl
3800   \__unravel_assign_token:n
3801   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
3802 }

```

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

```

3803 \__unravel_prefixed_new:nn { set_box } % 98
3804 {
3805   \__unravel_prev_input:V \l__unravel_head_tl
3806   \__unravel_scan_int:
3807   \__unravel_scan_optional_equals:
3808   \bool_if:NTF \g__unravel_set_box_allowed_bool
3809   { \__unravel_do_box:N \c_false_bool }
3810   {
3811     \msg_error:nn { unravel } { improper-setbox }
3812     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3813     \__unravel_omit_after_assignment:w
3814   }
3815 }

```

\hyphenation and \patterns

```

3816 \__unravel_prefixed_new:nn { hyph_data } % 99
3817 {
3818   \__unravel_prev_input:V \l__unravel_head_tl
3819   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3820   \__unravel_assign_token:n { }
3821 }
3822 \__unravel_prefixed_new:nn { set_interaction } % 100
3823 {
3824   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3825   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3826   \tl_use:N \l__unravel_head_tl \scan_stop:
3827   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3828 }
3829 \__unravel_prefixed_new:nn { letterspace_font } % 101
3830 {
3831   \__unravel_prev_input_silent:V \l__unravel_head_tl
3832   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3833   \__unravel_scan_r_token:
3834   \__unravel_print_action:x
3835   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }

```

```

3836 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
3837 \__unravel_scan_optional_equals:
3838 \__unravel_scan_font_ident:
3839 \__unravel_scan_int:
3840 \__unravel_assign_token:n { }
3841 }
3842 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
3843 {
3844 \__unravel_prev_input_silent:V \l__unravel_head_tl
3845 \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3846 \__unravel_scan_r_token:
3847 \__unravel_print_action:x
3848 { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
3849 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
3850 \__unravel_scan_optional_equals:
3851 \__unravel_scan_font_ident:
3852 \__unravel_assign_token:n { }
3853 }

```

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

```

3854 \__unravel_prefixed_new:nn { register } % 89
3855 { \__unravel_do_register:N \c_zero }
3856 \__unravel_prefixed_new:nn { advance } % 90
3857 { \__unravel_do_operation:N \c_one }
3858 \__unravel_prefixed_new:nn { multiply } % 91
3859 { \__unravel_do_operation:N \c_two }
3860 \__unravel_prefixed_new:nn { divide } % 92
3861 { \__unravel_do_operation:N \c_three }

```

__unravel_do_operation:N

__unravel_do_operation_fail:w

```

3862 \cs_new_protected:Npn \__unravel_do_operation:N #1
3863 {
3864 \__unravel_prev_input_silent:V \l__unravel_head_tl
3865 \__unravel_print_action:
3866 \__unravel_get_x_next:
3867 \__unravel_set_cmd:
3868 \int_compare:nNnTF
3869 \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
3870 {
3871 \int_compare:nNnTF
3872 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
3873 { \__unravel_do_register:N #1 }
3874 { \__unravel_do_operation_fail:w }
3875 }
3876 {
3877 \int_compare:nNnTF
3878 \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
3879 { \__unravel_do_operation_fail:w }

```

```

3880     {
3881       \__unravel_prev_input:V \l__unravel_head_tl
3882       \exp_args:NNf \__unravel_do_register_set:Nn #1
3883       {
3884         \int_eval:n
3885         {
3886           \l__unravel_head_cmd_int
3887           - \__unravel_tex_use:n { assign_toks }
3888         }
3889       }
3890     }
3891   }
3892 }
3893 \cs_new_protected_nopar:Npn \__unravel_do_operation_fail:w
3894 {
3895   \msg_error:nn { unravel } { after-advance }
3896   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3897   \__unravel_omit_after_assignment:w
3898 }

```

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

__unravel_do_register:N

__unravel_do_register_aux:Nn

```

3899 \cs_new_protected:Npn \__unravel_do_register:N #1
3900 {
3901   \exp_args:NNV \__unravel_do_register_aux:Nn #1
3902   \l__unravel_head_char_int
3903 }
3904 \cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
3905 {
3906   \int_compare:nNnTF { \tl_tail:n {#2} } = \c_zero
3907   {
3908     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3909     \__unravel_print_action:
3910     \__unravel_scan_int:
3911     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3912     \__unravel_prev_input_silent:V \l__unravel_head_tl
3913   }
3914   {
3915     \__unravel_prev_input_silent:V \l__unravel_head_tl
3916     \__unravel_print_action:
3917   }
3918   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3919   \exp_args:NNf \__unravel_do_register_set:Nn #1
3920   { \int_eval:n { #2 / 1 000 000 } }
3921 }

```

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

__unravel_do_register_set:Nn

```

3922 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
3923 {
3924   \int_compare:nNnTF {#1} = \c_zero
3925   { % truly register command
3926     \__unravel_scan_optional_equals:
3927   }
3928   { % \advance, \multiply, \divide
3929     \__unravel_scan_keyword:nF { bByY }
3930     { \__unravel_prev_input_silent:n { by } }
3931   }
3932   \int_compare:nNnTF {#1} < \c_two
3933   {
3934     \int_case:nnF {#2}
3935     {
3936       { 1 } { \__unravel_scan_int: } % count
3937       { 2 } { \__unravel_scan_normal_dimen: } % dim
3938       { 3 } { \__unravel_scan_normal_glue: } % glue
3939       { 4 } { \__unravel_scan_mu_glue: } % muglue
3940     }
3941     { \msg_error:nnx { unravel } { internal } { do-reg=#2 } }
3942   }
3943   { \__unravel_scan_int: }
3944   \__unravel_assign_register:
3945 }

```

(End definition for __unravel_do_register_set:Nn.)

The following is used for instance when making accents.

```

3946 \cs_new_protected_nopar:Npn \__unravel_do_assignments:
3947 {
3948   \__unravel_get_x_non_relax:
3949   \__unravel_set_cmd:
3950   \int_compare:nNnT
3951   \l__unravel_head_cmd_int
3952   > { \__unravel_tex_use:n { max_non_prefixed_command } }
3953   {
3954     \bool_gset_false:N \g__unravel_set_box_allowed_bool
3955     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3956     \__unravel_prefixed_command:
3957     \bool_gset_true:N \g__unravel_set_box_allowed_bool
3958     \__unravel_do_assignments:
3959   }
3960 }

```

2.14 Expandable primitives

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

- `undefined_cs=103` for undefined control sequences (not quite a primitive).
- `expand_after=104` for `\expandafter` and `\unless`.

- no_expand=105 for \noexpand and \pdfprimitive.
- input=106 for \input, \endinput and \scantokens.
- if_test=107 for the conditionals, \if, \ifcat, \ifnum, \ifdim, \ifodd, \ifvmode, \ifhmode, \ifmmode, \ifinner, \ifvoid, \ifhbox, \ifvbox, \ifx, \ifeof, \iftrue, \iffalse, \ifcase, \ifdefined, \ifcsname, \iffontchar, \ifincsname, \ifpdfprimitive, \ifpdfabsnum, and \ifpdfabsdim.
- fi_or_else=108 for \fi, \else and \or.
- cs_name=109 for \csname.
- convert=110 for \number, \romannumeral, \string, \meaning, \fontname, \eTeXrevision, \pdftexrevision, \pdftexbanner, \pdffontname, \pdffontobjnum, \pdffontsize, \pdfpageref, \pdfxformname, \pdfescapestring, \pdfescapename, \leftmarginkern, \rightmarginkern, \pdfstrcmp, \pdfcolorstackinit, \pdfescapehex, \pdfunescapehex, \pdfcreationdate, \pdffilemoddate, \pdffilesize, \pdfmdfivesum, \pdffiledump, \pdfmatch, \pdflastmatch, \pdfuniformdeviate, \pdfnormaldeviate, \pdfinsertht, \pdfximagebbox, and \jobname.
- 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 T_EX's end template.

Let T_EX trigger an error.

```

3961 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
3962 { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }

\__unravel_expandafter:
  \__unravel_unless:
  \__unravel_unless_bad:
3963 \__unravel_new_tex_expandable:nn { expand_after } % 104
3964 {
3965   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
3966   { \__unravel_expandafter: } { \__unravel_unless: }
3967 }
3968 \cs_new_protected_nopar:Npn \__unravel_expandafter:
3969 {
3970   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
3971   \__unravel_get_next:
3972   \gtl_concat:NNN \l__unravel_head_gtl
3973   \l__unravel_tmpb_gtl \l__unravel_head_gtl
3974   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_gtl
3975   \__unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
3976   \__unravel_get_next:
3977   \__unravel_token_if_expandable:NTF \l__unravel_head_token

```

```

3978     { \_unravel_expand: }
3979     { \_unravel_back_input: }
3980     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_gtl
3981     \_unravel_set_action_text:x
3982     { back_input: ~ \gtl_to_str:N \l\_unravel_head_gtl }
3983     \gtl_pop_left:N \l\_unravel_head_gtl
3984     \_unravel_back_input:
3985     \_unravel_print_action:
3986   }
3987   \cs_new_protected_nopar:Npn \_unravel_unless:
3988   {
3989     \_unravel_get_token:
3990     \int_compare:nNnTF
3991       \l\_unravel_head_cmd_int = { \_unravel_tex_use:n { if_test } }
3992     {
3993       \token_if_eq_meaning:NNTF \l\_unravel_head_token \tex_ifcase:D
3994       { \_unravel_unless_bad: }
3995       {
3996         \tl_put_left:Nn \l\_unravel_head_tl { \reverse_if:N }
3997         % \int_add:Nn \l\_unravel_head_char_int { 32 }
3998         \_unravel_expand_nonmacro:
3999       }
4000     }
4001     { \_unravel_unless_bad: }
4002   }
4003   \cs_new_protected_nopar:Npn \_unravel_unless_bad:
4004   {
4005     \msg_error:nn { unravel } { bad-unless }
4006     \_unravel_back_input:
4007   }

```

(End definition for _unravel_expandafter:, _unravel_unless:, and _unravel_unless_bad:.)

```

\_unravel_noexpand:
\_unravel_pdfprimitive:
4008 \_unravel_new_tex_expandable:nn { no_expand } % 105
4009 {
4010   \token_if_eq_meaning:NNTF \l\_unravel_head_token \tex_noexpand:D
4011   { \_unravel_noexpand: }
4012   { \_unravel_pdfprimitive: }
4013 }
4014 \cs_new_protected_nopar:Npn \_unravel_noexpand:
4015 {
4016   \_unravel_get_token:
4017   \_unravel_back_input:
4018   \_unravel_token_if_expandable:NT \l\_unravel_head_token
4019   {
4020     \cs_gset_protected_nopar:Npx \_unravel_get_next:
4021     {
4022       \cs_gset_protected_nopar:Npn \_unravel_get_next:
4023       { \exp_not:o { \_unravel_get_next: } }

```



```

4024         \exp_not:o { \__unravel_get_next: }
4025         \exp_not:n { \cs_set_eq:NN \l__unravel_head_token \tex_relax:D }
4026     }
4027 }
4028 }
4029 \cs_new_protected_nopar:Npn \__unravel_pdfprimitive:
4030 { \msg_error:nnx { unravel } { not-implemented } { pdfprimitive } }

```

(End definition for __unravel_noexpand: and __unravel_pdfprimitive:.)

```

\__unravel_endinput:
\__unravel_scantokens:
\__unravel_input:
4031 \__unravel_new_tex_expandable:nn { input } % 106
4032 {
4033     \int_case:nnF \l__unravel_head_char_int
4034     {
4035         { 1 } { \__unravel_endinput: } % \endinput
4036         { 2 } { \__unravel_scantokens: } % \scantokens
4037     }
4038     { % 0=\input
4039         \bool_if:NTF \g__unravel_name_in_progress_bool
4040         { \__unravel_insert_relax: } { \__unravel_input: }
4041     }
4042 }
4043 \cs_new_protected_nopar:Npn \__unravel_endinput:
4044 {
4045     \msg_warning:nn { unravel } { endinput-ignored }
4046     \__unravel_print_action:
4047 }
4048 \cs_new_protected_nopar:Npn \__unravel_scantokens:
4049 {
4050     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
4051     \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4052     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4053     \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4054     \__unravel_back_input:V \l__unravel_head_tl
4055     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4056 }
4057 \cs_new_protected_nopar:Npn \__unravel_input:
4058 {
4059     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4060     \__unravel_scan_file_name:
4061     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4062     \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4063     \__unravel_tl_gset_input:Nno \g__unravel_tmpc_tl { } \l__unravel_tmpa_tl
4064     \__unravel_back_input:V \g__unravel_tmpc_tl
4065     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4066 }

```

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

_unravel_csname_loop:

```

4067 \_unravel_new_tex_expandable:nn { cs_name } % 109
4068 {
4069   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4070   \_unravel_print_action:
4071   \_unravel_csname_loop:
4072   \_unravel_prev_input:V \l__unravel_head_tl
4073   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4074   \_unravel_back_input_tl_o:
4075 }
4076 \cs_new_protected_nopar:Npn \_unravel_csname_loop:
4077 {
4078   \_unravel_get_x_next:
4079   \token_if_cs:NTF \l__unravel_head_token
4080   {
4081     \cs_if_eq:NMF \l__unravel_head_token \tex_endcsname:D
4082     {
4083       \msg_error:nn { unravel } { missing-endcsname }
4084       \_unravel_back_input:
4085       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4086     }
4087   }
4088   {
4089     \_unravel_prev_input_silent:x
4090     { \_unravel_token_to_char:N \l__unravel_head_token }
4091     \_unravel_csname_loop:
4092   }
4093 }

```

(End definition for _unravel_csname_loop:.)

```

4094 \_unravel_new_tex_expandable:nn { convert } % 110
4095 {
4096   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4097   \_unravel_print_action:
4098   \int_case:nn \l__unravel_head_char_int
4099   {
4100     0 \_unravel_scan_int:
4101     1 \_unravel_scan_int:
4102     2 { \_unravel_get_next: \_unravel_prev_input:V \l__unravel_head_tl }
4103     3 { \_unravel_get_next: \_unravel_prev_input:V \l__unravel_head_tl }
4104     4 \_unravel_scan_font_ident:
4105     8 \_unravel_scan_font_ident:
4106     9 \_unravel_scan_font_ident:
4107     { 10 } \_unravel_scan_font_ident:
4108     { 11 } \_unravel_scan_int:
4109     { 12 } \_unravel_scan_int:
4110     { 13 } \_unravel_scan_pdf_ext_toks:
4111     { 14 } \_unravel_scan_pdf_ext_toks:
4112     { 15 } \_unravel_scan_int:

```

```

4113     { 16 } \_unravel_scan_int:
4114     { 17 } \_unravel_scan_pdfstrcmp:
4115     { 18 } \_unravel_scan_pdfcolorstackinit:
4116     { 19 } \_unravel_scan_pdf_ext_toks:
4117     { 20 } \_unravel_scan_pdf_ext_toks:
4118     { 22 } \_unravel_scan_pdf_ext_toks:
4119     { 23 } \_unravel_scan_pdf_ext_toks:
4120     { 24 }
4121     {
4122         \_unravel_scan_keyword:n { fF iI lL eE }
4123         \_unravel_scan_pdf_ext_toks:
4124     }
4125     { 25 } \_unravel_scan_pdffiledump:
4126     { 26 } \_unravel_scan_pdfmatch:
4127     { 27 } \_unravel_scan_int:
4128     { 28 } \_unravel_scan_int:
4129     { 30 } \_unravel_scan_int:
4130     { 31 } \_unravel_scan_pdfximagebbox:
4131 }
4132 \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_head_tl
4133 \_unravel_back_input_tl_o:
4134 }
4135 \cs_new_protected_nopar:Npn \_unravel_scan_pdfstrcmp:
4136 {
4137     \_unravel_scan_toks_to_str:
4138     \_unravel_scan_toks_to_str:
4139 }
4140 \cs_new_protected_nopar:Npn \_unravel_scan_pdfximagebbox:
4141 { \_unravel_scan_int: \_unravel_scan_int: }
4142 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstackinit:
4143 {
4144     \_unravel_scan_keyword:nTF { pP aA gG eE }
4145     { \bool_set_true:N \l_unravel_tmpa_bool }
4146     { \bool_set_false:N \l_unravel_tmpb_bool }
4147     \_unravel_scan_keyword:nF { dD iI rR eE cC tT }
4148     { \_unravel_scan_keyword:n { pP aA gG eE } }
4149     \_unravel_scan_toks_to_str:
4150 }
4151 \cs_new_protected_nopar:Npn \_unravel_scan_pdffiledump:
4152 {
4153     \_unravel_scan_keyword:nT { oO fF fF sS eE tT } \_unravel_scan_int:
4154     \_unravel_scan_keyword:nT { lL eE nN gG tT hH } \_unravel_scan_int:
4155     \_unravel_scan_pdf_ext_toks:
4156 }
4157 \cs_new_protected_nopar:Npn \_unravel_scan_pdfmatch:
4158 {
4159     \_unravel_scan_keyword:n { iI cC aA sS eE }
4160     \_unravel_scan_keyword:nT { sS uU bB cC oO uU nN tT }
4161     { \_unravel_scan_int: }
4162     \_unravel_scan_pdf_ext_toks:

```

```

4163 \_unravel_scan_pdf_ext_toks:
4164 }

\_unravel_get_the:

4165 \_unravel_new_tex_expandable:nn { the } % 111
4166 {
4167 \_unravel_get_the:
4168 \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4169 \_unravel_back_input:V \l__unravel_tmpa_tl
4170 \_unravel_print_action:
4171 }
4172 \cs_new_protected_nopar:Npn \_unravel_get_the:
4173 {
4174 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4175 \_unravel_print_action:
4176 \int_if_odd:nTF \l__unravel_head_char_int
4177 { % \unexpanded, \detokenize
4178 \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4179 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4180 \_unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4181 }
4182 { % \the
4183 \_unravel_get_x_next:
4184 \_unravel_scan_something_internal:n { 5 }
4185 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4186 \_unravel_set_action_text:x
4187 {
4188 \tl_head:N \l__unravel_head_tl
4189 => \tl_tail:N \l__unravel_head_tl
4190 }
4191 \tl_set:Nx \l__unravel_head_tl
4192 { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4193 }
4194 }

```

(End definition for _unravel_get_the:.)

```

4195 \_unravel_new_tex_expandable:nn { top_bot_mark } % 112
4196 { \_unravel_back_input_tl_o: }

4197 \_unravel_new_tex_expandable:nn { end_template } % 117
4198 {
4199 \msg_error:nn { unravel } { not-implemented } { end-template }
4200 \_unravel_back_input_tl_o:
4201 }

```

2.14.1 Conditionals

```

\_unravel_pass_text:
\_unravel_pass_text_done:w
4202 \cs_new_protected_nopar:Npn \_unravel_pass_text:
4203 {

```

```

4204 \__unravel_input_if_empty:TF
4205 { \__unravel_pass_text_empty: }
4206 {
4207   \__unravel_input_get:N \l__unravel_tmpb_gtl
4208   \if_true:
4209     \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one
4210     \exp_after:wN \__unravel_pass_text_done:w
4211     \fi:
4212     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4213     \exp_after:wN \__unravel_pass_text:
4214   \else:
4215     \use:c { fi: }
4216     \int_set_eq:NN \l__unravel_if_nesting_int \c_one
4217     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4218     \exp_after:wN \__unravel_pass_text_nested:
4219     \fi:
4220   }
4221 }
4222 \cs_new_protected_nopar:Npn \__unravel_pass_text_done:w
4223 {
4224   \__unravel_get_next:
4225   \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4226   \else:
4227   }

```

(End definition for __unravel_pass_text:.)

__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 $\langle head \rangle$ 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: .

```

4228 \cs_new_protected_nopar:Npn \__unravel_pass_text_nested:
4229 {
4230   \__unravel_input_if_empty:TF
4231   { \__unravel_pass_text_empty: }
4232   {
4233     \__unravel_input_get:N \l__unravel_tmpb_gtl
4234     \if_true:
4235       \if_true:
4236         \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4237         \int_decr:N \l__unravel_if_nesting_int

```

```

4238         \use_none:nnnnn
4239         \fi:
4240         \use_none:nnn
4241         \fi:
4242         \int_incr:N \l__unravel_if_nesting_int
4243         \fi:
4244         \__unravel_input_gpop:N \l__unravel_unused_gtl
4245         \int_compare:nNnTF \l__unravel_if_nesting_int = \c_zero
4246         { \__unravel_pass_text: }
4247         { \__unravel_pass_text_nested: }
4248     }
4249 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

4250 \cs_new_protected_nopar:Npn \__unravel_pass_text_empty:
4251 {
4252     \msg_error:nn { unravel } { runaway-if }
4253     \__unravel_exit:w
4254 }

```

(End definition for __unravel_pass_text_empty:.)

__unravel_cond_push:
__unravel_cond_pop:

```

4255 \cs_new_protected:Npn \__unravel_cond_push:
4256 {
4257     \tl_gput_left:Nx \g__unravel_if_limit_tl
4258     { { \int_use:N \g__unravel_if_limit_int } }
4259     \int_gincr:N \g__unravel_if_depth_int
4260     \int_gzero:N \g__unravel_if_limit_int
4261 }
4262 \cs_new_protected_nopar:Npn \__unravel_cond_pop:
4263 {
4264     \int_gset:Nn \g__unravel_if_limit_int
4265     { \tl_head:N \g__unravel_if_limit_tl }
4266     \tl_gset:Nx \g__unravel_if_limit_tl
4267     { \tl_tail:N \g__unravel_if_limit_tl }
4268     \int_gdecr:N \g__unravel_if_depth_int
4269 }

```

(End definition for __unravel_cond_push: and __unravel_cond_pop:.)

_unravel_change_if_limit:nn

```

4270 \cs_new_protected:Npn \_unravel_change_if_limit:nn #1#2
4271 {
4272     \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4273     { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4274     {
4275         \tl_clear:N \l__unravel_tpa_tl

```

```

4276 \prg_replicate:nn { \g__unravel_if_depth_int - #2 - \c_one }
4277 {
4278   \tl_put_right:Nx \l__unravel_tmpa_tl
4279   { { \tl_head:N \g__unravel_if_limit_tl } }
4280   \tl_gset:Nx \g__unravel_if_limit_tl
4281   { \tl_tail:N \g__unravel_if_limit_tl }
4282 }
4283 \tl_gset:Nx \g__unravel_if_limit_tl
4284 { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4285 }
4286 }

```

(End definition for __unravel_change_if_limit:nn.)

```

4287 \__unravel_new_tex_expandable:nn { if_test } % 107
4288 {
4289   \__unravel_cond_push:
4290   \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
4291 }

```

__unravel_cond_aux:nn

```

4292 \cs_new_protected:Npn \__unravel_cond_aux:n #1
4293 {
4294   \int_case:nnF \l__unravel_head_char_int
4295   {
4296     { 12 } { \__unravel_test_ifx:n {#1} }
4297     { 16 } { \__unravel_test_case:n {#1} }
4298     { 21 } { \__unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4299   }
4300   {
4301     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4302     \__unravel_print_action:
4303     \int_case:nn \l__unravel_head_char_int
4304     {
4305       { 0 } { \__unravel_test_two_chars: } % if
4306       { 1 } { \__unravel_test_two_chars: } % ifcat
4307       { 2 } % ifnum
4308       { \__unravel_test_two_vals:N \__unravel_scan_int: }
4309       { 3 } % ifdim
4310       { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4311       { 4 } { \__unravel_scan_int: } % ifodd
4312       % { 5 } { } % ifvmode
4313       % { 6 } { } % ifhmode
4314       % { 7 } { } % ifmmode
4315       % { 8 } { } % ifinner
4316       { 9 } { \__unravel_scan_int: } % ifvoid
4317       { 10 } { \__unravel_scan_int: } % ifhbox
4318       { 11 } { \__unravel_scan_int: } % ifvbox
4319       { 13 } { \__unravel_scan_int: } % ifeof
4320       % { 14 } { } % iftrue
4321       % { 15 } { } % iffals

```

```

4322         { 17 } { \_unravel_test_ifdefined: } % ifdefined
4323         { 18 } { \_unravel_test_ifcsname: } % ifcsname
4324         { 19 } % iffontchar
4325         { \_unravel_scan_font_ident: \_unravel_scan_int: }
4326         % { 20 } { } % ifincsname % ^A todo: something?
4327         { 22 } % ifpdfabsnum
4328         { \_unravel_test_two_vals:N \_unravel_scan_int: }
4329         { 23 } % ifpdfabsdim
4330         { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4331     }
4332     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
4333     \_unravel_set_action_text:x { \tl_to_str:N \l\_unravel_head_tl }
4334     \l\_unravel_head_tl \scan_stop:
4335     \exp_after:wN \_unravel_cond_true:n
4336     \else:
4337     \exp_after:wN \_unravel_cond_false:n
4338     \fi:
4339     {#1}
4340 }
4341 }

```

(End definition for _unravel_cond_aux:nn.)

_unravel_cond_true:n

```

4342 \cs_new_protected:Npn \_unravel_cond_true:n #1
4343 {
4344     \_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4345     \_unravel_print_action:x { \g\_unravel_action_text_str = true }
4346 }

```

(End definition for _unravel_cond_true:n.)

_unravel_cond_false:n

_unravel_cond_false_loop:n

_unravel_cond_false_common:

```

4347 \cs_new_protected:Npn \_unravel_cond_false:n #1
4348 {
4349     \_unravel_cond_false_loop:n {#1}
4350     \_unravel_cond_false_common:
4351     \_unravel_print_action:x { \g\_unravel_action_text_str = false }
4352 }
4353 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4354 {
4355     \_unravel_pass_text:
4356     \int_compare:nNnTF \g\_unravel_if_depth_int = {#1}
4357     {
4358         \token_if_eq_meaning:NNT \l\_unravel_head_token \or:
4359         {
4360             \msg_error:nn { unravel } { extra-or }
4361             \_unravel_cond_false_loop:n {#1}
4362         }
4363     }

```



```

4364     {
4365         \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4366         { \__unravel_cond_pop: }
4367         \__unravel_cond_false_loop:n {#1}
4368     }
4369 }
4370 \cs_new_protected_nopar:Npn \__unravel_cond_false_common:
4371 {
4372     \token_if_eq_meaning:NNTF \l__unravel_head_token \fi:
4373     { \__unravel_cond_pop: }
4374     { \int_gset:Nn \g__unravel_if_limit_int { 2 } } % wait for fi
4375 }

```

(End definition for __unravel_cond_false:n, __unravel_cond_false_loop:n, and __unravel_cond_false_common:.)

__unravel_test_two_vals:N

```

4376 \cs_new_protected:Npn \__unravel_test_two_vals:N #1
4377 {
4378     #1
4379     \__unravel_get_x_non_blank:
4380     \tl_if_in:nVF { < = > } \l__unravel_head_tl
4381     {
4382         \msg_error:nn { unravel } { missing-equals }
4383         \__unravel_back_input:
4384         \tl_set:Nn \l__unravel_head_tl { = }
4385     }
4386     \__unravel_prev_input:V \l__unravel_head_tl
4387     #1
4388 }

```

(End definition for __unravel_test_two_vals:N.)

__unravel_test_two_chars:

__unravel_test_two_chars_aux:

```

4389 \cs_new_protected_nopar:Npn \__unravel_test_two_chars:
4390 {
4391     \__unravel_test_two_chars_aux:
4392     \__unravel_prev_input:V \l__unravel_head_tl
4393     \__unravel_test_two_chars_aux:
4394     \__unravel_prev_input:V \l__unravel_head_tl
4395 }
4396 \cs_new_protected_nopar:Npn \__unravel_test_two_chars_aux:
4397 {
4398     \__unravel_get_x_next:
4399     \gtl_if_tl:NF \l__unravel_head_gtl
4400     {
4401         \tl_set:Nx \l__unravel_head_tl
4402         {
4403             \gtl_if_head_is_group_begin:NNTF \l__unravel_head_gtl
4404             { \c_group_begin_token } { \c_group_end_token }

```

```

4405     }
4406   }
4407   \tl_put_left:Nn \l__unravel_head_tl { \exp_not:N } % ^^A todo: prettify.
4408 }

```

(End definition for __unravel_test_two_chars: and __unravel_test_two_chars_aux:.)

```

\__unravel_test_ifx:n
\__unravel_test_ifx_aux:w
4409 \cs_new_protected:Npn \__unravel_test_ifx:n #1
4410 {
4411   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4412   \__unravel_print_action:
4413   \__unravel_get_next:
4414   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4415   \__unravel_get_next:
4416   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4417   \__unravel_set_action_text:x
4418   {
4419     Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4420     \gtl_to_str:N \l__unravel_tmpb_gtl
4421     \gtl_to_str:N \l__unravel_head_gtl
4422   }
4423   \gtl_head_do:NN \l__unravel_tmpb_gtl \__unravel_test_ifx_aux:w
4424   \exp_after:wN \__unravel_cond_true:n
4425   \else:
4426     \exp_after:wN \__unravel_cond_false:n
4427   \fi:
4428   {#1}
4429 }
4430 \cs_new_nopar:Npn \__unravel_test_ifx_aux:w
4431 { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }

```

(End definition for __unravel_test_ifx:n and __unravel_test_ifx_aux:w.)

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn
4432 \cs_new_protected:Npn \__unravel_test_case:n #1
4433 {
4434   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4435   \__unravel_print_action:
4436   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
4437   \__unravel_scan_int:
4438   \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4439   \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
4440   % ^^A does text_case_aux use prev_input_seq?
4441   \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
4442   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4443   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4444 }
4445 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
4446 {

```

```

4447 \int_compare:nNnTF {#1} = \c_zero
4448 { \__unravel_change_if_limit:nn { 4 } {#2} }
4449 {
4450   \__unravel_pass_text:
4451   \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
4452   {
4453     \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
4454     {
4455       \exp_args:Nf \__unravel_test_case_aux:nn
4456       { \int_eval:n { #1 - 1 } } {#2}
4457     }
4458     { \__unravel_cond_false_common: }
4459   }
4460   {
4461     \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4462     { \__unravel_cond_pop: }
4463     \__unravel_test_case_aux:nn {#1} {#2}
4464   }
4465 }
4466 }

```

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

__unravel_test_ifdefined:

```

4467 \cs_new_protected_nopar:Npn \__unravel_test_ifdefined:
4468 {
4469   \__unravel_input_if_empty:TF
4470   { \__unravel_pass_text_empty: }
4471   {
4472     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4473     \__unravel_set_action_text:x
4474     {
4475       Conditional:~ \tl_to_str:N \l__unravel_head_tl
4476       \gtl_to_str:N \l__unravel_tmpb_gtl
4477     }
4478     \__unravel_prev_input:x
4479     {
4480       \gtl_if_tl:NNTF \l__unravel_tmpb_gtl
4481       { \gtl_head:N \l__unravel_tmpb_gtl }
4482       { \gtl_to_str:N \l__unravel_tmpb_gtl }
4483     }
4484   }
4485 }

```

(End definition for __unravel_test_ifdefined:.)

__unravel_test_ifcsname:

```

4486 \cs_new_protected_nopar:Npn \__unravel_test_ifcsname:
4487 {
4488   \__unravel_csname_loop:

```

```

4489   \_unravel_prev_input:V \l_unravel_head_tl
4490 }

(End definition for \_unravel_test_ifcname:.)

4491 \_unravel_new_tex_expandable:nn { fi_or_else } % 108
4492 {
4493   \int_compare:nNnTF \l_unravel_head_char_int > \g_unravel_if_limit_int
4494   {
4495     \int_compare:nNnTF \g_unravel_if_limit_int = \c_zero
4496     {
4497       \int_compare:nNnTF \g_unravel_if_depth_int = \c_zero
4498       { \msg_error:nn { unravel } { extra-fi-or-else } }
4499       { \_unravel_insert_relax: }
4500     }
4501     { \msg_error:nn { unravel } { extra-fi-or-else } }
4502   }
4503   {
4504     \_unravel_set_action_text:
4505     \int_compare:nNnF \l_unravel_head_char_int = \c_two
4506     {
4507       \_unravel-fi-or-else-loop:
4508       \_unravel_set_action_text:x
4509       {
4510         \g_unravel_action_text_str \c_space_tl
4511         => ~ skipped ~ to ~ \tl_to_str:N \l_unravel_head_tl
4512       }
4513     }
4514     % ^^A todo: in this print_action the token itself is missing.
4515     \_unravel_print_action:
4516     \_unravel_cond_pop:
4517   }
4518 }
4519 \cs_new_protected_nopar:Npn \_unravel-fi-or-else-loop:
4520 {
4521   \int_compare:nNnF \l_unravel_head_char_int = \c_two
4522   {
4523     \_unravel_pass_text:
4524     \_unravel_set_cmd:
4525     \_unravel-fi-or-else-loop:
4526   }
4527 }

```

2.15 User interaction

2.15.1 Print

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

__unravel_print:n

__unravel_print:x

```
4528 \cs_new_eq:NN \__unravel_print:n \iow_term:n
4529 \cs_generate_variant:Nn \__unravel_print:n { x }
```

(End definition for __unravel_print:n and __unravel_print:x.)

__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.

```
4530 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
4531 { \iow_wrap:nnnN { #1 #2 } { #1 } { } } \__unravel_print:n }
```

(End definition for __unravel_print_message:nn.)

__unravel_set_action_text:x

```
4532 \cs_new_protected:Npn \__unravel_set_action_text:x #1
4533 {
4534   \group_begin:
4535     \__unravel_set_escapechar:n { 92 }
4536     \str_gset:Nx \g__unravel_action_text_str {#1}
4537   \group_end:
4538 }
```

(End definition for __unravel_set_action_text:x.)

__unravel_set_action_text:

```
4539 \cs_new_protected_nopar:Npn \__unravel_set_action_text:
4540 {
4541   \__unravel_set_action_text:x
4542   {
4543     \tl_to_str:N \l__unravel_head_tl
4544     \tl_if_single_token:VT \l__unravel_head_tl
4545     { = ~ \exp_after:wN \token_to_meaning:N \l__unravel_head_tl }
4546   }
4547 }
```

(End definition for __unravel_set_action_text:.)

__unravel_print_state:

```
4548 \cs_new_protected:Npn \__unravel_print_state:
4549 {
4550   \group_begin:
4551     \__unravel_set_escapechar:n { 92 }
4552     \tl_use:N \g__unravel_before_print_state_tl
4553     \int_compare:nNnT \g__unravel_noise_int > \c_zero
4554     {
4555       \exp_args:Nx \__unravel_print_state_output:n
4556       { \gtl_to_str:N \g__unravel_output_gtl }
4557       \seq_set_map:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
4558       { \__unravel_to_str:n {##1} }
4559       \seq_remove_all:Nn \l__unravel_tmpa_seq { }
```

```

4560         \exp_args:Nx \__unravel_print_state_prev:n
4561         { \seq_use:Nn \l__unravel_tmpa_seq { \ } }
4562         \exp_args:Nx \__unravel_print_state_input:n
4563         { \__unravel_input_to_str: }
4564     }
4565     \group_end:
4566     \__unravel_prompt:
4567 }

```

(End definition for __unravel_print_state:.)

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

```

4568 \cs_new_protected:Npn \__unravel_print_state_output:n #1
4569 {
4570     \tl_if_empty:nF {#1}
4571     {
4572         \__unravel_print_message:nn { <| ~ }
4573         { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
4574     }
4575 }

```

(End definition for __unravel_print_state_output:n.)

__unravel_print_state_prev:n Never trim #1.

```

4576 \cs_new_protected:Npn \__unravel_print_state_prev:n #1
4577 { \__unravel_print_message:nn { || ~ } {#1} }

```

(End definition for __unravel_print_state_prev:n.)

__unravel_print_state_input:n 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.

```

4578 \cs_new_protected:Npn \__unravel_print_state_input:n #1
4579 {
4580     \__unravel_print_message:nn { |> ~ }
4581     { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
4582 }

```

(End definition for __unravel_print_state_input:n.)

__unravel_print_meaning:

```

4583 \cs_new_protected:Npn \__unravel_print_meaning:
4584 {
4585     \__unravel_input_if_empty:TF
4586     { \__unravel_print_message:nn { } { Empty-input! } }
4587     {
4588         \__unravel_input_get:N \l__unravel_tmpb_gtl
4589         \__unravel_print_message:nn { }

```

```

4590         {
4591             \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
4592             = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
4593         }
4594     }
4595 }

```

(End definition for __unravel_print_meaning:.)

__unravel_print_action:
__unravel_print_action:x

```

4596 \cs_new_protected:Npn \__unravel_print_action:
4597 {
4598     \int_gincr:N \g__unravel_step_int
4599     \__unravel_print:x
4600     {
4601         [====
4602         \bool_if:NT \g__unravel_number_steps_bool
4603         { ~ Step ~ \int_use:N \g__unravel_step_int \ }
4604         =====]~
4605         \int_compare:nNnTF
4606         { \str_count:N \g__unravel_action_text_str }
4607         > { \g__unravel_max_action_int }
4608         {
4609             \str_range:Nnn \g__unravel_action_text_str
4610             { 1 } { \g__unravel_max_action_int - 3 } ...
4611         }
4612         { \g__unravel_action_text_str }
4613     }
4614     \__unravel_print_state:
4615 }
4616 \cs_new_protected:Npn \__unravel_print_action:x #1
4617 {
4618     \__unravel_set_action_text:x {#1}
4619     \__unravel_print_action:
4620 }

```

(End definition for __unravel_print_action: and __unravel_print_action:x.)

__unravel_print_gtl_action:N

```

4621 \cs_new_protected:Npn \__unravel_print_gtl_action:N #1
4622 {
4623     \__unravel_print_action:x { \gtl_to_str:N #1 }
4624 }

```

(End definition for __unravel_print_gtl_action:N.)

__unravel_print_done:x

```

4625 \cs_new_eq:NN \__unravel_print_done:x \__unravel_print_action:x

```

(End definition for __unravel_print_done:x.)

```

\__unravel_print_assigned_token:
\__unravel_print_assigned_register:
4626 \cs_new_protected_nopar:Npn \__unravel_print_assigned_token:
4627 {
4628   \__unravel_after_assignment: % ^^A todo: simplify
4629   \__unravel_print_action:x
4630   {
4631     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4632     = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
4633   }
4634   \__unravel_omit_after_assignment:w
4635 }
4636 \cs_new_protected_nopar:Npn \__unravel_print_assigned_register:
4637 {
4638   \__unravel_after_assignment: % ^^A todo: simplify
4639   \__unravel_print_action:x
4640   {
4641     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4642     \tl_if_single:NT \l__unravel_defined_tl
4643     { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
4644     = \exp_after:wN \__unravel_the:w \l__unravel_defined_tl
4645   }
4646   \__unravel_omit_after_assignment:w
4647 }

```

(End definition for __unravel_print_assigned_token: and __unravel_print_assigned_register:.)

```

\__unravel_print_welcome: Welcome message.
4648 \cs_new_protected_nopar:Npn \__unravel_print_welcome:
4649 {
4650   \__unravel_print_message:nn { }
4651   {
4652     \bool_if:NTF \g__unravel_welcome_message_bool
4653     {
4654       \\\
4655       =====~ Welcome~ to~ the~ unravel~ package~ =====\\
4656       \iow_indent:n
4657       {
4658         "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\
4659         "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\\
4660         ">|"~ denotes~ tokens~ that~ we~ will~ act~ on. \\\
4661         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\\
4662       }
4663     }
4664     { [=====Start=====] }
4665   }
4666   \__unravel_print_state:
4667 }

```

(End definition for __unravel_print_welcome:.)

`__unravel_print_outcome:` Final message.

```

4668 \cs_new_protected_nopar:Npn \__unravel_print_outcome:
4669   { \__unravel_print:n { [====~End~====] } }

(End definition for \__unravel_print_outcome:.)

```

2.15.2 Prompt

`__unravel_prompt:`

```

4670 \cs_new_protected_nopar:Npn \__unravel_prompt:
4671   {
4672     \int_gdecr:N \g__unravel_nonstop_int
4673     \int_compare:nNnF \g__unravel_nonstop_int > \c_zero
4674       {
4675         \group_begin:
4676           \__unravel_set_escapechar:n { -1 }
4677           \int_set_eq:NN \tex_endlinechar:D \c_minus_one
4678           \tl_use:N \g__unravel_before_prompt_tl
4679           \__unravel_prompt_aux:
4680         \group_end:
4681       }
4682   }
4683 \cs_new_protected_nopar:Npn \__unravel_prompt_aux:
4684   {
4685     \int_compare:nNnT { \etex_interactionmode:D } = { 3 }
4686       {
4687         \bool_if:NTF \g__unravel_explicit_prompt_bool
4688           { \ior_get_str:Nc \c__unravel_prompt_ior }
4689           { \ior_get_str:Nc \c__unravel_noprompt_ior }
4690           { Your~input }
4691         \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
4692       }
4693   }
4694 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
4695   {
4696     \tl_if_empty:nF {#1}
4697     {
4698       \exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
4699       {
4700         { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
4701         { q }
4702         {
4703           \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4704           \int_gzero:N \g__unravel_nonstop_int
4705         }
4706         { x }
4707         {
4708           \group_end:
4709           \exp_after:wN \__unravel_exit:w \__unravel_exit:w
4710         }

```

```

4711 { X } { \tex_batchmode:D \tex_end:D }
4712 { s } { \_unravel_prompt_scan_int:nn {#1}
4713 \_unravel_prompt_silent_steps:n }
4714 { o } { \_unravel_prompt_scan_int:nn {#1}
4715 { \int_gset:Nn \g__unravel_noise_int } }
4716 { C }
4717 {
4718 \tl_gset_rescan:Nnx \g__unravel_tmpc_tl
4719 { \ExplSyntaxOn } { \tl_tail:n {#1} }
4720 \tl_gput_left:Nn \g__unravel_tmpc_tl
4721 { \tl_gclear:N \g__unravel_tmpc_tl }
4722 \group_insert_after:N \g__unravel_tmpc_tl
4723 }
4724 { | } { \_unravel_prompt_scan_int:nn {#1}
4725 \_unravel_prompt_vert:n }
4726 { a } { \_unravel_prompt_all: }
4727 }
4728 { \_unravel_prompt_help: }
4729 }
4730 }
4731 \cs_new_protected:Npn \_unravel_prompt_scan_int:nn #1
4732 {
4733 \tex_afterassignment:D \_unravel_prompt_scan_int_after:wn
4734 \l__unravel_prompt_tmpa_int =
4735 \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
4736 \use_ii:nn #1 \scan_stop:
4737 }
4738 \cs_new_protected:Npn \_unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
4739 {
4740 #2 \l__unravel_prompt_tmpa_int
4741 \tl_if_blank:nF {#1} { \_unravel_prompt_treat:n {#1} }
4742 }
4743 \cs_new_protected:Npn \_unravel_prompt_help:
4744 {
4745 \_unravel_print:n { "m":~meaning-of~first~token }
4746 \_unravel_print:n { "q":~semi-quiet~(same-as~"o1") }
4747 \_unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
4748 \_unravel_print:n { "s<num>":~do~<num>~steps~silently }
4749 \_unravel_print:n
4750 { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
4751 \_unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
4752 \_unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
4753 \_unravel_print:n { "a":~print~state~again,~without~truncating }
4754 \_unravel_prompt_aux:
4755 }
4756 \cs_new_protected:Npn \_unravel_prompt_silent_steps:n #1
4757 {
4758 \int_compare:nNnF {#1} < \c_zero
4759 {
4760 \int_gset_eq:NN \g__unravel_noise_int \c_minus_one

```

```

4761         \tl_gset:Nn \g__unravel_before_prompt_tl
4762         {
4763             \int_gset_eq:NN \g__unravel_noise_int \c_one
4764             \tl_gclear:N \g__unravel_before_prompt_tl
4765         }
4766         \int_gset:Nn \g__unravel_nonstop_int {#1}
4767     }
4768 }
4769 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
4770 {
4771     \int_compare:nNnTF {#1} < { 0 }
4772     { \__unravel_prompt_vert:Nn > {#1} }
4773     { \__unravel_prompt_vert:Nn < {#1} }
4774 }
4775 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
4776 {
4777     \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4778     \tl_gset:Nf \g__unravel_before_print_state_tl
4779     {
4780         \exp_args:Nnf \exp_stop_f: \int_compare:nNnTF
4781         { \int_eval:n { \seq_count:N \g__unravel_prev_input_seq - #2 } }
4782         #1 { \seq_count:N \g__unravel_prev_input_seq }
4783         {
4784             \int_gset:Nn \g__unravel_nonstop_int
4785             { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
4786         }
4787         {
4788             \int_gset_eq:NN \g__unravel_noise_int \c_one
4789             \tl_gclear:N \g__unravel_before_print_state_tl
4790         }
4791     }
4792 }
4793 \cs_new_protected_nopar:Npn \__unravel_prompt_all:
4794 {
4795     \tl_gset:Nx \g__unravel_tmpc_tl
4796     {
4797         \exp_not:n
4798         {
4799             \tl_gclear:N \g__unravel_tmpc_tl
4800             \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
4801             \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
4802             \__unravel_print_state:
4803         }
4804         \__unravel_prompt_all_aux:N \g__unravel_max_output_int
4805         \__unravel_prompt_all_aux:N \g__unravel_max_input_int
4806     }
4807     \group_insert_after:N \g__unravel_tmpc_tl
4808 }
4809 \cs_new:Npn \__unravel_prompt_all_aux:N #1
4810 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for `_unravel_prompt:`)

2.16 Keys

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

```

4811 \keys_define:nn { unravel/defaults }
4812 {
4813   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
4814   internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
4815   max-action .int_gset:N = \g__unravel_default_max_action_int ,
4816   max-output .int_gset:N = \g__unravel_default_max_output_int ,
4817   max-input .int_gset:N = \g__unravel_default_max_input_int ,
4818   number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
4819   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
4820 }
4821 \keys_define:nn { unravel }
4822 {
4823   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
4824   internal-debug .bool_gset:N = \g__unravel_internal_debug_bool ,
4825   max-action .int_gset:N = \g__unravel_max_action_int ,
4826   max-output .int_gset:N = \g__unravel_max_output_int ,
4827   max-input .int_gset:N = \g__unravel_max_input_int ,
4828   number-steps .bool_gset:N = \g__unravel_number_steps_bool ,
4829   welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,
4830 }

```

The `machine` option is somewhat special so it is clearer to define it 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.

```

4831 \tl_map_inline:nn { { /defaults } { } }
4832 {
4833   \keys_define:nn { unravel #1 }
4834   {
4835     machine .meta:nn =
4836     { unravel #1 }
4837     {
4838       explicit-prompt = false ,
4839       internal-debug = false ,
4840       max-action = \c_max_int ,
4841       max-output = \c_max_int ,
4842       max-input = \c_max_int ,
4843       number-steps = false ,
4844       welcome-message = false ,
4845     } ,
4846   }
4847 }

```

2.17 Main command

\unravel Simply call an underlying code-level command.

```
4848 \NewDocumentCommand \unravel { 0 { } m } { \unravel:nn {#1} {#2} }
```

(End definition for \unravel. This function is documented on page 1.)

\unravelsetup Simply call an underlying code-level command.

```
4849 \NewDocumentCommand \unravelsetup { m } { \unravel_setup:n {#1} }
```

(End definition for \unravelsetup. This function is documented on page 1.)

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

```
4850 \cs_new_protected:Npn \unravel_setup:n #1
4851 {
4852   \keys_set:nn { unravel/defaults } {#1}
4853   \keys_set:nn { unravel } {#1}
4854 }
```

(End definition for \unravel_setup:n. This function is documented on page 1.)

\unravel:nn Initialize and setup keys. 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.

```
4855 \cs_new_protected:Npn \unravel:nn #1#2
4856 {
4857   \__unravel_init_key_vars:
4858   \keys_set:nn { unravel } {#1}
4859   \__unravel_init_vars:
4860   \__unravel_input_gset:n {#2}
4861   \__unravel_print_welcome:
4862   \__unravel_main_loop:
4863   \__unravel_exit_point:
4864   \__unravel_print_outcome:
4865   \__unravel_final_test:
4866   \__unravel_exit_point:
4867 }
```

(End definition for \unravel:nn.)

__unravel_init_key_vars: Give variables that are affected by keys their default values (also controlled by keys).

```
4868 \cs_new_protected_nopar:Npn \__unravel_init_key_vars:
4869 {
4870   \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bool
4871   \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
4872   \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
4873   \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bool
```

```

4874 \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
4875 \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
4876 \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
4877 }

```

(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.

```

4878 \cs_new_protected_nopar:Npn \_unravel_init_vars:
4879 {
4880   \seq_gclear:N \g__unravel_prev_input_seq
4881   \gtl_gclear:N \g__unravel_output_gtl
4882   \int_gzero:N \g__unravel_step_int
4883   \tl_gclear:N \g__unravel_if_limit_tl
4884   \int_gzero:N \g__unravel_if_limit_int
4885   \int_gzero:N \g__unravel_if_depth_int
4886   \gtl_gclear:N \g__unravel_after_assignment_gtl
4887   \bool_gset_true:N \g__unravel_set_box_allowed_bool
4888   \bool_gset_false:N \g__unravel_name_in_progress_bool
4889 }

```

(End definition for _unravel_init_vars:.)

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

```

4890 \cs_new_protected_nopar:Npn \_unravel_main_loop:
4891 {
4892   \_unravel_get_x_next:
4893   \_unravel_set_cmd:
4894   \_unravel_do_step:
4895   \_unravel_main_loop:
4896 }

```

(End definition for _unravel_main_loop:.)

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

```

\_unravel_final_bad:
4897 \cs_new_protected_nopar:Npn \_unravel_final_test:
4898 {
4899   \bool_if:nTF
4900   {
4901     \tl_if_empty_p:N \g__unravel_if_limit_tl
4902     && \int_compare_p:nNn \g__unravel_if_limit_int = \c_zero
4903     && \int_compare_p:nNn \g__unravel_if_depth_int = \c_zero
4904     && \seq_if_empty_p:N \g__unravel_prev_input_seq
4905   }
4906   { \_unravel_input_if_empty:TF { } { \_unravel_final_bad: } }
4907   { \_unravel_final_bad: }
4908 }
4909 \cs_new_protected_nopar:Npn \_unravel_final_bad:

```

```

4910 {
4911   \msg_error:nxx { unravel } { internal }
4912   { the-last-unravel-finished-badly }
4913 }

```

(End definition for _unravel_final_test: and _unravel_final_bad:.)

2.18 Messages

```

4914 \msg_new:nnn { unravel } { unknown-primitive }
4915   { Internal~error:~the~primitive~'#1'~is~not~known. }
4916 \msg_new:nnn { unravel } { extra-fi-or-else }
4917   { Extra~fi,~or,~or~else. }
4918 \msg_new:nnn { unravel } { missing-lbrace }
4919   { Missing~left~brace~inserted. }
4920 \msg_new:nnn { unravel } { missing-dollar }
4921   { Missing~dollar~inserted. }
4922 \msg_new:nnn { unravel } { unknown-expandable }
4923   { Internal~error:~the~expandable~command~'#1'~is~not~known. }
4924 \msg_new:nnn { unravel } { missing-font-id }
4925   { Missing~font~identifier.~\iow_char:N\~nullfont~inserted. }
4926 \msg_new:nnn { unravel } { missing-rparen }
4927   { Missing~right~parenthesis~inserted~for~expression. }
4928 \msg_new:nnn { unravel } { incompatible-units }
4929   { Mu~glue/dimen~used~as~a~normal~glue/dimen~or~vice-versa. }
4930 \msg_new:nnn { unravel } { missing-mu }
4931   { Missing~mu~unit. }
4932 \msg_new:nnn { unravel } { missing-cs }
4933   { Missing~control~sequence.~\iow_char:N\~inaccessible~inserted. }
4934 \msg_new:nnn { unravel } { missing-box }
4935   { Missing~box~inserted. }
4936 \msg_new:nnn { unravel } { missing-to }
4937   { Missing~keyword~'to'~inserted. }
4938 \msg_new:nnn { unravel } { improper-leaders }
4939   { Leaders~not~followed~by~proper~glue. }
4940 \msg_new:nnn { unravel } { extra-close }
4941   { Extra~right~brace~or~\iow_char:N\~endgroup. }
4942 \msg_new:nnn { unravel } { off-save }
4943   { Something~is~wrong~with~groups. }
4944 \msg_new:nnn { unravel } { hrule-bad-mode }
4945   { \iow_char\~hrule~used~in~wrong~mode. }
4946 \msg_new:nnn { unravel } { invalid-mode }
4947   { Invalid~mode~for~this~command. }
4948 \msg_new:nnn { unravel } { color-stack-action-missing }
4949   { Missing~color~stack~action. }
4950 \msg_new:nnn { unravel } { action-type-missing }
4951   { Missing~action~type. }
4952 \msg_new:nnn { unravel } { identifier-type-missing }
4953   { Missing~identifier~type. }
4954 \msg_new:nnn { unravel } { destination-type-missing }

```

```

4955 { Missing~destination~type. }
4956 \msg_new:nnn { unravel } { erroneous-prefixes }
4957 { Prefixes~applied~to~non~assignment~command. }
4958 \msg_new:nnn { unravel } { improper-setbox }
4959 { \iow_char:N\setbox~while~fetching~base~of~an~accent. }
4960 \msg_new:nnn { unravel } { after~advance }
4961 {
4962   Missing~register~after~\iow_char:N\advance,~
4963   \iow_char:N\multiply,~or~\iow_char:N\divide.
4964 }
4965 \msg_new:nnn { unravel } { bad-unless }
4966 { \iow_char:N\unless~not~followed~by~conditional. }
4967 \msg_new:nnn { unravel } { missing-endcsname }
4968 { Missing~\iow_char:N\endcsname~inserted. }
4969 \msg_new:nnn { unravel } { runaway-if }
4970 { Runaway~\iow_char:N\if... }
4971 \msg_new:nnn { unravel } { runaway-macro-parameter }
4972 {
4973   Runaway-macro-parameter~\# #2~after \\\
4974   \iow_indent:n {#1}
4975 }
4976 \msg_new:nnn { unravel } { extra-or }
4977 { Extra~\iow_char:N\or. }
4978 \msg_new:nnn { unravel } { missing-equals }
4979 { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
4980 \msg_new:nnn { unravel } { internal }
4981 { Internal~error:~'#1'.~\ Please~report. }
4982 \msg_new:nnn { unravel } { not-implemented }
4983 { The~following~feature~is~not~implemented:~'#1'. }
4984 \msg_new:nnn { unravel } { endinput-ignored }
4985 { The~primitive~\iow_char:N\endinput~was~ignored. }
4986 </package>

```