

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2015/09/30

Contents

1	unravel documentation	1
1.1	Options	3
1.2	Differences between <code>unravel</code> and TeX's processing	3
1.3	Future perhaps	4
2	unravel implementation	5
2.1	Primitives, variants, and helpers	9
2.1.1	Renamed primitives	9
2.1.2	Variants	9
2.1.3	Miscellaneous helpers	10
2.1.4	String helpers	11
2.1.5	Helpers for control flow	13
2.1.6	Helpers concerning tokens	14
2.1.7	Helpers for previous input	16
2.2	Variables	17
2.2.1	User interaction	17
2.2.2	Working with tokens	19
2.2.3	Numbers and conditionals	21
2.2.4	Boxes and groups	21
2.2.5	Constants	22
2.2.6	TeX parameters	22
2.3	Numeric codes	22
2.4	Get next token	37
2.5	Manipulating the input	41
2.5.1	Elementary operations	41
2.5.2	Insert token for error recovery	47
2.5.3	Macro calls	48
2.6	Expand next token	49

*This file has version number 0.2, last revised 2015/09/30.

2.7	Basic scanning subroutines	50
2.8	Working with boxes	70
2.9	Paragraphs	74
2.10	Groups	77
2.11	Modes	79
2.12	One step	80
2.13	Commands	81
2.13.1	Characters: from 0 to 15	81
2.13.2	Boxes: from 16 to 31	85
2.13.3	From 32 to 47	90
2.13.4	Maths: from 48 to 56	95
2.13.5	From 57 to 70	95
2.13.6	Extensions	98
2.13.7	Assignments	105
2.14	Expandable primitives	116
2.14.1	Conditionals	122
2.15	User interaction	130
2.15.1	Print	130
2.15.2	Prompt	135
2.15.3	Errors	138
2.16	Keys	138
2.17	Main command	139
2.18	Messages	142

1 unravel documentation

The aim of this L^AT_EX package is to help debug complicated macros. This is done by letting the user step through the execution of some T_EX code, going through the details of nested expansions, performing assignments, as well as some simple typesetting commands. To use this package, one should normally run T_EX in a terminal.

`\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`, `XƒTeX`, `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 `LATEX` on the following file.

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

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

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

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

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

Given all the work that `unravel` has to do to emulate \TeX , it is not fast on very large pieces of code. For instance, running it on `\documentclass{article}` takes about 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 `pdf \TeX` , 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</u> <u>max-output</u> <u>max-input</u>	Integer options (defaults 50, 300, 300) determining the maximum number of characters displayed for the action, the output part, and the input part.
<u>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`, `\discretionary`, as well as all primitives specific to engines other than pdf`TEX`. This list may sadly be incomplete!
- `\aftergroup` is only partially implemented.
- `\everyhbox`, `\everyvbox`, `\everymath`, `\everydisplay`, `\lastkern`, `\lastnodetype`, `\lastpenalty`, `\lastskip`, `\currentiflevel` and `\currentiftyp` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgrouptype` too.
- Tokens passed to `\afterassignment` are not yet kept after `unravel` is done even if there has been no assignment.
- Tokens passed to `\aftergroup` are lost when `unravel` is done.
- In X_Y`TEX`, characters beyond the basic multilingual plane may break `unravel` (not tested).
- For `unravel`, category codes are fixed when a file is read using `\input`, while `TEX` only fixes category codes when the corresponding characters are converted to tokens. Similarly, the argument of `\scantokens` is converted to the new category code régime in one go, and the result must be balanced.
- Explicit begin-group and end-group characters other than the usual left and right braces may make `unravel` choke, or may be silently replaced by the usual left and right braces.
- `\endinput` is ignored with a warning, as it is very difficult to implement it in a way similar to `TEX`'s, and as it is most often used at the very end of files, in a redundant way.
- `\outer` is not supported.

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`).
- Use the `file-error` fatal error message: first implement `\@@_file_if_exist:nTF` and use it to determine whether `\input` will throw a fatal error in `\batchmode` and `\nonstopmode`.

- Use the `interwoven-preambles` fatal error message once alignments are implemented.
- Look at all places where T_EX's procedure `prepare_mag` is called.
- Find out why so many input levels are used (see the log of the `unrave1003` testfile for instance)

2 unravel implementation

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

15 \expandafter\ifx\csname currentgrouplevel\endcsname\relax\else
16 \ifnum\currentgrouplevel>1 \errmessage{unravel loaded in a group}
17 \endinput\expandafter\expandafter\expandafter\endgroup\fi\fi

```

Make spaces ignored and make ~ a space, to prettify code.

```
18 \catcode 32 = 9 \relax
19 \catcode 126 = 10 \relax
```

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

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

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

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

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

(End definition for __unravel_setup_restore:.)

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

`__unravel_setup_save_aux:n`

```
26 \protected \gdef \__unravel_setup_save:
27 {
28   \edef \l__unravel_setup_restore_tl
29   {
30     \__unravel_setup_save_aux:w 0 =
31     \endlinechar = \the \endlinechar
32     \escapechar = \the \escapechar
33     \newlinechar = \the \newlinechar
34     \relax
35   }
36 }
37 \long \gdef \__unravel_setup_save_aux:w #1 =
38 {
39   \catcode #1 = \the \catcode #1 ~
40   \ifnum 255 > #1 ~
41     \expandafter \__unravel_setup_save_aux:w
42     \the \numexpr #1 + 1 \expandafter =
43     \fi
44 }
```

(End definition for __unravel_setup_save:.)

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

```
45 \protected \long \gdef \__unravel_setup_catcodes:nnn #1 #2 #3
46 {
47   \ifnum #1 > #2 ~ \else
48     \catcode #1 = #3 ~
49     \expandafter \__unravel_setup_catcodes:nnn \expandafter
```

```

50         { \the \numexpr #1 + 1 } {#2} {#3}
51     \fi
52 }

```

(End definition for `_unravel_setup_catcodes:nnn`.)

`_unravel_setup_latex:` This saves the catcodes and related parameters, then sets them to the value they normally have in a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\varepsilon}$ package (in particular, @ is a letter).

```

53 \protected \gdef \_unravel_setup_latex:
54 {
55     \_unravel_setup_save:
56     \_unravel_setup_catcodes:nnn {0} {8} {15}
57     \catcode 9 = 10 ~
58     \catcode 10 = 12 ~
59     \catcode 11 = 15 ~
60     \catcode 12 = 13 ~
61     \catcode 13 = 5 ~
62     \_unravel_setup_catcodes:nnn {14} {31} {15}
63     \catcode 32 = 10 ~
64     \catcode 33 = 12 ~
65     \catcode 34 = 12 ~
66     \catcode 35 = 6 ~
67     \catcode 36 = 3 ~
68     \catcode 37 = 14 ~
69     \catcode 38 = 4 ~
70     \_unravel_setup_catcodes:nnn {39} {63} {12}
71     \_unravel_setup_catcodes:nnn {64} {90} {11}
72     \catcode 91 = 12 ~
73     \catcode 92 = 0 ~
74     \catcode 93 = 12 ~
75     \catcode 94 = 7 ~
76     \catcode 95 = 8 ~
77     \catcode 96 = 12 ~
78     \_unravel_setup_catcodes:nnn {97} {122} {11}
79     \catcode 123 = 1 ~
80     \catcode 124 = 12 ~
81     \catcode 125 = 2 ~
82     \catcode 126 = 13 ~
83     \catcode 127 = 15 ~
84     \_unravel_setup_catcodes:nnn {128} {255} {12}
85     \endlinechar = 13 ~
86     \escapechar = 92 ~
87     \newlinechar = 10 ~
88 }

```

(End definition for `_unravel_setup_latex:.`)

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

```

89 \protected \gdef \_unravel_setup_unravel:

```



```

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

```

(End definition for __unravel_setup_unravel:.)

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

```

127 \expandafter \endgroup \__unravel_setup_latex:

```

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

```

128 \RequirePackage{expl3,xparse}[2015/09/11]
129 \RequirePackage{gtl}[2015/09/21]
130 \csname cs_if_exist:cF\endcsname{str_range:nnn}{\RequirePackage{l3str}}

```

Before loading unravel, restore catcodes, so that the implicit \ExplSyntaxOn in \ProvidesExplPackage picks up the correct catcodes to restore when \ExplSyntaxOff

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

```

131 \csname use:n\endcsname
132 {%
133   \csname __unravel_setup_restore:\endcsname
134   \ProvidesExplPackage
135     {unravel} {2015/09/30} {0.2} {Watching TeX digest tokens}%
136   \csname __unravel_setup_unravel:\endcsname
137 }%
```

2.1 Primitives, variants, and helpers

2.1.1 Renamed primitives

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

```

\__unravel_set_escapechar:n 138 \cs_new_eq:NN \__unravel_currentgrouptype: \etex_currentgrouptype:D
\__unravel_nullfont:        139 \cs_new_protected_nopar:Npn \__unravel_set_escapechar:n
\__unravel_hbox:w          140 { \int_set:Nn \tex_escapechar:D }
\__unravel_the:w           141 \cs_new_eq:NN \__unravel_everyeof:w \etex_everyeof:D
                             142 \cs_new_eq:NN \__unravel_everypar:w \tex_everypar:D
                             143 \cs_new_eq:NN \__unravel_hbox:w \tex_hbox:D
                             144 \cs_new_eq:NN \__unravel_mag: \tex_mag:D
                             145 \cs_new_eq:NN \__unravel_nullfont: \tex_nullfont:D
                             146 \cs_new_eq:NN \__unravel_the:w \tex_the:D
```

(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.

```

147 \cs_new_eq:NN \c__unravel_prompt_ior \c_sixteen
148 \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.

```

149 \cs_generate_variant:Nn \str_head:n { f }
150 \cs_generate_variant:Nn \tl_to_str:n { o }
151 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
152 \cs_generate_variant:Nn \tl_if_in:nnF { nV }
153 \cs_generate_variant:Nn \tl_if_in:nnTF { nV }
154 \cs_generate_variant:Nn \tl_if_in:NnTF { No , NV }
```

```

155 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
156 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nnx }
157 \cs_generate_variant:Nn \gtl_gput_right:Nn { NV }
158 \cs_generate_variant:Nn \ior_get_str:NN { Nc }
159 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
160 \cs_generate_variant:Nn \gtl_to_str:N { c }
161 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
162 \cs_generate_variant:Nn \gtl_get_left:NN { c }
163 \cs_generate_variant:Nn \gtl_gset:Nn { c }
164 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
165 \cs_generate_variant:Nn \gtl_gclear:N { c }
166 \cs_generate_variant:Nn \gtl_gclear_new:N { c }

```

2.1.3 Miscellaneous helpers

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

```

167 \cs_new_protected_nopar:Npn \__unravel_tmp:w { }

```

(End definition for __unravel_tmp:w.)

```

\__unravel_file_get:nN
\__unravel_file_get_aux:wN
168 \cs_set_protected:Npn \__unravel_tmp:w #1
169 {
170   \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
171   {
172     \group_begin:
173       \__unravel_everyeof:w { #1 ##2 }
174       \exp_after:wN \__unravel_file_get_aux:wN
175       \exp_after:wN \prg_do_nothing:
176       \tex_input:D ##1 \scan_stop:
177   }
178   \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
179   {
180     \group_end:
181     \tl_set:Nx ##2
182     { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
183   }
184 }
185 \exp_args:No \__unravel_tmp:w { \token_to_str:N : : }

```

(End definition for __unravel_file_get:nN.)

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

The surrounding `\int_eval:n` lets us dump digits in the input stream while keeping the function fully expandable.

```

186 \cs_new:Npn \__unravel_tl_first_int:N #1
187 {
188   \int_eval:n
189   {
190     \exp_after:wN \__unravel_tl_first_int_aux:Nn
191     \exp_after:wN \__unravel_tl_first_int_aux:Nn
192     #1 ? 0 ? \q_stop
193   }
194 }
195 \cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
196 {
197   \tl_if_single:nT {#2}
198   {
199     \token_if_eq_catcode:NNT + #2
200     {
201       \if_int_compare:w 1 < 1 #2 \exp_stop_f:
202       #2
203       \exp_after:wN \use_i_ii:nnn
204       \exp_after:wN \__unravel_tl_first_int_aux:Nn
205       \exp_after:wN \use_none_delimit_by_q_stop:w
206       \fi:
207     }
208   }
209   #1
210 }

```

(End definition for `__unravel_tl_first_int:N`.)

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

```

211 \cs_new_protected_nopar:Npn \__unravel_prepare_mag:
212 {
213   \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
214   {
215     \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
216     {
217       \__unravel_tex_error:nn { incompatible-mag } { }
218       \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
219     }
220   }
221   \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
222   {
223     \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
224     \int_gset:Nn \__unravel_mag: { 1000 }
225   }
226   \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
227 }

```

(End definition for `__unravel_prepare_mag:.`)

2.1.4 String helpers

`__unravel_strip_escape:w` This is based on the 2013-07-19 (and earlier) version of `\cs_to_str:N`. There are three cases. If the escape character is printable, the charcode test is false, and `__unravel_strip_escape_aux:N` removes one character. If the escape character is a space, the charcode test is true, and if there is no escape character, the test is unfinished after `\token_to_str:N \`. In both of those cases, `__unravel_strip_escape_aux:w` inserts `-__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.

```

228 \cs_new_nopar:Npn \__unravel_strip_escape:w
229   {
230     \tex_romannumeral:D
231     \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
232     \__unravel_strip_escape_aux:N
233   }
234 \cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero }
235 \cs_new:Npn \__unravel_strip_escape_aux:w #1#2
236   { - \__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.

```

\__unravel_to_str_auxi:w
\__unravel_to_str_auxii:w
\__unravel_gtl_to_str:n
237 \cs_new:Npn \__unravel_to_str:n #1
238   {
239     \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
240     { \__unravel_to_str_auxi:w #1 ? \q_stop }
241     { \tl_to_str:n }
242     {#1}
243   }
244 \cs_set:Npn \__unravel_tmp:w #1
245   {
246     \cs_new:Npn \__unravel_to_str_auxi:w ##1##2 \q_stop
247     {
248       \exp_after:wN \__unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark
249       #1 tl \q_mark \q_stop
250     }
251     \cs_new:Npn \__unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
252     { \cs_if_exist_use:cF { \__unravel_ ##2 _to_str:n } { \tl_to_str:n } }
253   }
254 \exp_args:No \__unravel_tmp:w { \tl_to_str:n { s_ } }
255 \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.

```

256 \cs_new:Npn \_unravel_str_truncate_left:nn #1#2
257   {
258     \exp_args:Nf \_unravel_str_truncate_left_aux:nnn
259       { \str_count:n {#1} } {#1} {#2}
260   }
261 \cs_new:Npn \_unravel_str_truncate_left_aux:nnn #1#2#3
262   {
263     \int_compare:nNnTF {#1} > {#3}
264     {
265       ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
266       \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
267     }
268     { \tl_to_str:n {#2} }
269   }

```

(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.

```

270 \cs_new:Npn \_unravel_str_truncate_right:nn #1#2
271   {
272     \exp_args:Nf \_unravel_str_truncate_right_aux:nnn
273       { \str_count:n {#1} } {#1} {#2}
274   }
275 \cs_new:Npn \_unravel_str_truncate_right_aux:nnn #1#2#3
276   {
277     \int_compare:nNnTF {#1} > {#3}
278     {
279       \str_range:nnn {#2} { 1 } { #3 - 25 } ~
280       ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
281     }
282     { \tl_to_str:n {#2} }
283   }

```

(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:
284 \cs_new_eq:NN \_unravel_exit_point: \prg_do_nothing:
285 \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:`

```

286 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
287 \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).

```

288 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
289 {
290   \int_compare:nNnTF
291     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
292     { \prg_return_false: }
293     {
294       \int_compare:nNnTF
295         \l__unravel_head_cmd_int
296         > { \__unravel_tex_use:n { max_internal } }
297         { \prg_return_false: }
298         { \prg_return_true: }
299     }
300 }

```

(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
\__unravel_meaning_to_char_auxii:w
301 \cs_new:Npn \__unravel_meaning_to_char:n #1
302   { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
303 \cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
304   { \__unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
305 \cs_new:Npn \__unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
306   { \tl_if_empty:nTF {#2} { ~ } {#2} }
307 \cs_generate_variant:Nn \__unravel_meaning_to_char:n { o }
308 \cs_new:Npn \__unravel_token_to_char:N #1
309   { \__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` `NTF` one does not think that undefined is expandable.

```

310 \prg_new_conditional:Npnn \__unravel_token_if_expandable:N #1
311   { p , T , F , TF }
312   {
313     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
314     \prg_return_false:
315   }
316   \else:
317     \prg_return_true:

```

```

317   \fi:
318   }

(End definition for \_unravel_token_if_expandable:NTF.)

```

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

```

\_unravel_token_if_protected:NTF
319 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
320 { p , T , F , TF }
321 {
322   \_unravel_token_if_expandable:NTF #1
323   {
324     \token_if_protected_macro:NTF #1
325     { \prg_return_true: }
326     {
327       \token_if_protected_long_macro:NTF #1
328       { \prg_return_true: }
329       { \prg_return_false: }
330     }
331   }
332   { \prg_return_true: }
333 }

```

(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.

```

334 \group_begin:
335   \char_set_catcode_active:n { 'Z }
336   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
337   { TF }
338   {
339     \group_begin:
340     \_unravel_set_escapechar:n { 92 }
341     \tl_set:Nx \l__unravel_tmpa_tl
342     { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
343     \tl_if_empty:NTF \l__unravel_tmpa_tl
344     {
345       \exp_args:Nx \char_set_lccode:nn
346       { ' \str_head:n {#1} } { 'Z }

```



```

347         \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
348         { \group_end: \prg_return_true: }
349         { \group_end: \prg_return_false: }
350     }
351     { \group_end: \prg_return_true: }
352 }
353 \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.

```

354 \prg_new_protected_conditional:Npnm \_unravel_gtl_if_head_is_definable:N #1
355 { TF , F }
356 {
357     \gtl_if_single_token:NTF #1
358     {
359         \gtl_if_head_is_N_type:NTF #1
360         {
361             \exp_last_unbraced:Nx \_unravel_token_if_definable:NTF
362             { \gtl_head:N #1 }
363             { \prg_return_true: }
364             { \prg_return_false: }
365         }
366         { \prg_return_false: }
367     }
368     { \prg_return_false: }
369 }

```

(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
\_unravel_prev_input:n
\_unravel_prev_input:V
\_unravel_prev_input:x
370 \cs_new_protected:Npn \_unravel_prev_input_silent:n #1
371 {
372     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_tl
373     \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
374     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_tl
375 }
376 \cs_generate_variant:Nn \_unravel_prev_input_silent:n { V , x }
377 \cs_new_protected:Npn \_unravel_prev_input:n #1
378 {
379     \_unravel_prev_input_silent:n {#1}
380     \_unravel_print_action:x { \tl_to_str:n {#1} }
381 }
382 \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`

```

383 \cs_new_protected:Npn \_unravel_prev_input_gtl:N #1
384   {
385     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
386     \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
387     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
388   }

```

(End definition for `_unravel_prev_input_gtl:N`.)

`_unravel_prev_input_join_get:nN`
`_unravel_join_get_aux:NNN`

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.

```

389 \cs_new_protected:Npn \_unravel_prev_input_join_get:nN #1
390   {
391     \int_case:nnF {#1}
392     {
393       { 2 } { \_unravel_join_get_aux:NNN \skip_eval:n \etex_glueexpr:D }
394       { 3 } { \_unravel_join_get_aux:NNN \muskip_eval:n \etex_muexpr:D }
395     }
396     {
397       \msg_error:nnn { unravel } { internal } { join-factor }
398       \_unravel_join_get_aux:NNN \use:n \prg_do_nothing:
399     }
400   }
401 \cs_new_protected:Npn \_unravel_join_get_aux:NNN #1#2#3
402   {
403     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
404     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
405     \tl_set:Nx #3 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl } }
406   }

```

(End definition for `_unravel_prev_input_join_get:nN`.)

2.2 Variables

2.2.1 User interaction

`\g__unravel_before_print_state_tl`
`\g__unravel_before_prompt_tl`

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

```

407 \tl_new:N \g__unravel_before_print_state_tl
408 \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`

```

409 \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` 410 `\int_new:N \g__unravel_nonstop_int`
411 `\int_new:N \g__unravel_noise_int`
412 `\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
`\g__unravel_default_explicit_prompt_bool` `default_` booleans store the default value of these options, and are affected by
`\g__unravel_default_internal_debug_bool` `\unravelsetup` or `\unravel_setup:n`.
`\g__unravel_internal_debug_bool` 413 `\bool_new:N \g__unravel_default_explicit_prompt_bool`
`\g__unravel_default_number_steps_bool` 414 `\bool_new:N \g__unravel_default_internal_debug_bool`
`\g__unravel_number_steps_bool` 415 `\bool_new:N \g__unravel_default_number_steps_bool`
`\g__unravel_default_welcome_message_bool` 416 `\bool_gset_true:N \g__unravel_default_number_steps_bool`
`\g__unravel_welcome_message_bool` 417 `\bool_new:N \g__unravel_default_welcome_message_bool`
418 `\bool_gset_true:N \g__unravel_default_welcome_message_bool`
419 `\bool_new:N \g__unravel_explicit_prompt_bool`
420 `\bool_new:N \g__unravel_internal_debug_bool`
421 `\bool_new:N \g__unravel_number_steps_bool`
422 `\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.
423 `\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.
424 `\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` 425 `\int_new:N \g__unravel_default_max_action_int`
`\g__unravel_default_max_input_int` 426 `\int_new:N \g__unravel_default_max_output_int`
`\g__unravel_max_action_int` 427 `\int_new:N \g__unravel_default_max_input_int`
`\g__unravel_max_output_int` 428 `\int_gset:Nn \g__unravel_default_max_action_int { 50 }`
`\g__unravel_max_input_int` 429 `\int_gset:Nn \g__unravel_default_max_output_int { 300 }`
430 `\int_gset:Nn \g__unravel_default_max_input_int { 300 }`
431 `\int_new:N \g__unravel_max_action_int`
432 `\int_new:N \g__unravel_max_output_int`
433 `\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.

```
434 \bool_new:N \g__unravel_speedup_macros_bool
435 \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.

```
436 \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.

```
437 \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`

```
438 \int_new:N \g__unravel_input_tmpa_int
439 \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` 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.

```
440 \seq_new:N \g__unravel_prev_input_seq
441 \tl_new:N \l__unravel_prev_input_tl
442 \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.

```
443 \gtl_new:N \g__unravel_output_gtl
```

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

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

```

\l__unravel_head_t1
\l__unravel_head_token
\l__unravel_head_cmd_int
\l__unravel_head_char_int
444 \gtl_new:N \l__unravel_head_gtl
445 \tl_new:N \l__unravel_head_t1
446 \token_new:Nn \l__unravel_head_token { ? }
447 \int_new:N \l__unravel_head_cmd_int
448 \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_t1`

```

449 \tl_new:N \l__unravel_head_meaning_t1

```

(End definition for `\l__unravel_head_meaning_t1`. This variable is documented on page ??.)

`\l__unravel_tmpa_t1` Temporary storage. The `\l__unravel_unused_gtl` is only used once, to ignore some unwanted tokens.

```

\l__unravel_tmpb_gtl
\g__unravel_tmpe_t1
\l__unravel_tmpa_seq
\l__unravel_unused_gtl
450 \tl_new:N \l__unravel_tmpa_t1
451 \gtl_new:N \l__unravel_unused_gtl
452 \gtl_new:N \l__unravel_tmpb_gtl
453 \tl_new:N \g__unravel_tmpe_t1
454 \seq_new:N \l__unravel_tmpa_seq

```

(End definition for `\l__unravel_tmpa_t1` and others. These variables are documented on page ??.)

`\l__unravel_defined_t1` 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.

```

455 \tl_new:N \l__unravel_defined_t1
456 \tl_new:N \l__unravel_defining_t1

```

(End definition for `\l__unravel_defined_t1` and `\l__unravel_defining_t1`. These variables are documented on page ??.)

`__unravel_inaccessible:w`

```

457 \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
458 \gtl_new:N \g__unravel_after_assignment_gtl
459 \bool_new:N \g__unravel_set_box_allowed_bool
460 \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 ??.)

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

```
461 \gtl_new:N \l__unravel_after_group_gtl
```

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

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

```
462 \group_begin:
463   \cs_set:Npx \__unravel_tmp:w #1 { \c_hash_str #1 }
464   \tl_const:Nx \c__unravel_parameters_tl
465     { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
466 \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.

```
467 \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 468 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 469 \int_new:N \g__unravel_if_limit_int
470 \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`

```
471 \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.

```
472 \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.

```
473 \int_new:N \g__unravel_ends_int
474 \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 475 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 476 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 477 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 478 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 479 \tl_const:Nn \c__unravel_lq_tl { ' }
\c__unravel_dq_tl 480 \tl_const:Nn \c__unravel_rq_tl { ' }
\c__unravel_lp_tl 481 \tl_const:Nn \c__unravel_dq_tl { " }
\c__unravel_rp_tl 482 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 483 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 484 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 485 \tl_const:Nn \c__unravel_comma_tl { , }
486 \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:`.

```
487 \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 ??.)

2.2.6 \TeX parameters

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

```
488 \int_new:N \g__unravel_mag_set_int
```

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

2.3 Numeric codes

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

```

\__unravel_tex_const:nn
  \__unravel_tex_use:n
489 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
490   { \int_const:cn { c__unravel_tex_#1_int } {#2} }
491 \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
492 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
493   {
494     \tl_const:cx { c__unravel_tex_#1_tl }
495     { { \__unravel_tex_use:n {#2} } {#3} }
496   }

```

(End definition for __unravel_tex_primitive:nnn.)

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
497 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
498   {
499     \cs_new_protected_nopar:cpn
500     { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
501   }
502 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
503   {
504     \cs_new_eq:cc
505     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
506     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
507   }

```

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

```

\__unravel_new_tex_expandable:nn
508 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
509   {
510     \cs_new_protected_nopar:cpn
511     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
512   }

```

(End definition for __unravel_new_tex_expandable:nn.)

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

```

513 \__unravel_tex_const:nn { relax } { 0 }
514 \__unravel_tex_const:nn { begin-group_char } { 1 }
515 \__unravel_tex_const:nn { end-group_char } { 2 }
516 \__unravel_tex_const:nn { math_char } { 3 }

```



```

517 \__unravel_tex_const:nn { tab_mark           } { 4 }
518 \__unravel_tex_const:nn { alignment_char     } { 4 }
519 \__unravel_tex_const:nn { car_ret           } { 5 }
520 \__unravel_tex_const:nn { macro_char        } { 6 }
521 \__unravel_tex_const:nn { superscript_char  } { 7 }
522 \__unravel_tex_const:nn { subscript_char   } { 8 }
523 \__unravel_tex_const:nn { endv             } { 9 }
524 \__unravel_tex_const:nn { blank_char       } { 10 }
525 \__unravel_tex_const:nn { the_char         } { 11 }
526 \__unravel_tex_const:nn { other_char       } { 12 }
527 \__unravel_tex_const:nn { par_end         } { 13 }
528 \__unravel_tex_const:nn { stop            } { 14 }
529 \__unravel_tex_const:nn { delim_num       } { 15 }
530 \__unravel_tex_const:nn { max_char_code    } { 15 }
531 \__unravel_tex_const:nn { char_num        } { 16 }
532 \__unravel_tex_const:nn { math_char_num   } { 17 }
533 \__unravel_tex_const:nn { mark            } { 18 }
534 \__unravel_tex_const:nn { xray            } { 19 }
535 \__unravel_tex_const:nn { make_box        } { 20 }
536 \__unravel_tex_const:nn { hmove          } { 21 }
537 \__unravel_tex_const:nn { vmove          } { 22 }
538 \__unravel_tex_const:nn { un_hbox        } { 23 }
539 \__unravel_tex_const:nn { un_vbox        } { 24 }
540 \__unravel_tex_const:nn { remove_item     } { 25 }
541 \__unravel_tex_const:nn { hskip          } { 26 }
542 \__unravel_tex_const:nn { vskip          } { 27 }
543 \__unravel_tex_const:nn { mskip          } { 28 }
544 \__unravel_tex_const:nn { kern           } { 29 }
545 \__unravel_tex_const:nn { mkern          } { 30 }
546 \__unravel_tex_const:nn { leader_ship    } { 31 }
547 \__unravel_tex_const:nn { halign         } { 32 }
548 \__unravel_tex_const:nn { valign         } { 33 }
549 \__unravel_tex_const:nn { no_align       } { 34 }
550 \__unravel_tex_const:nn { vrule          } { 35 }
551 \__unravel_tex_const:nn { hrule          } { 36 }
552 \__unravel_tex_const:nn { insert         } { 37 }
553 \__unravel_tex_const:nn { vadjust       } { 38 }
554 \__unravel_tex_const:nn { ignore_spaces  } { 39 }
555 \__unravel_tex_const:nn { after_assignment } { 40 }
556 \__unravel_tex_const:nn { after_group    } { 41 }
557 \__unravel_tex_const:nn { break_penalty  } { 42 }
558 \__unravel_tex_const:nn { start_par      } { 43 }
559 \__unravel_tex_const:nn { ital_corr      } { 44 }
560 \__unravel_tex_const:nn { accent         } { 45 }
561 \__unravel_tex_const:nn { math_accent    } { 46 }
562 \__unravel_tex_const:nn { discretionary } { 47 }
563 \__unravel_tex_const:nn { eq_no         } { 48 }
564 \__unravel_tex_const:nn { left_right     } { 49 }
565 \__unravel_tex_const:nn { math_comp     } { 50 }
566 \__unravel_tex_const:nn { limit_switch  } { 51 }

```

```

567 \__unravel_tex_const:nn { above } { 52 }
568 \__unravel_tex_const:nn { math_style } { 53 }
569 \__unravel_tex_const:nn { math_choice } { 54 }
570 \__unravel_tex_const:nn { non_script } { 55 }
571 \__unravel_tex_const:nn { vcenter } { 56 }
572 \__unravel_tex_const:nn { case_shift } { 57 }
573 \__unravel_tex_const:nn { message } { 58 }
574 \__unravel_tex_const:nn { extension } { 59 }
575 \__unravel_tex_const:nn { in_stream } { 60 }
576 \__unravel_tex_const:nn { begin_group } { 61 }
577 \__unravel_tex_const:nn { end_group } { 62 }
578 \__unravel_tex_const:nn { omit } { 63 }
579 \__unravel_tex_const:nn { ex_space } { 64 }
580 \__unravel_tex_const:nn { no_boundary } { 65 }
581 \__unravel_tex_const:nn { radical } { 66 }
582 \__unravel_tex_const:nn { end_cs_name } { 67 }
583 \__unravel_tex_const:nn { min_internal } { 68 }
584 \__unravel_tex_const:nn { char_given } { 68 }
585 \__unravel_tex_const:nn { math_given } { 69 }
586 \__unravel_tex_const:nn { last_item } { 70 }
587 \__unravel_tex_const:nn { max_non_prefixed_command } { 70 }
588 \__unravel_tex_const:nn { toks_register } { 71 }
589 \__unravel_tex_const:nn { assign_toks } { 72 }
590 \__unravel_tex_const:nn { assign_int } { 73 }
591 \__unravel_tex_const:nn { assign_dimen } { 74 }
592 \__unravel_tex_const:nn { assign_glue } { 75 }
593 \__unravel_tex_const:nn { assign_mu_glue } { 76 }
594 \__unravel_tex_const:nn { assign_font_dimen } { 77 }
595 \__unravel_tex_const:nn { assign_font_int } { 78 }
596 \__unravel_tex_const:nn { set_aux } { 79 }
597 \__unravel_tex_const:nn { set_prev_graf } { 80 }
598 \__unravel_tex_const:nn { set_page_dimen } { 81 }
599 \__unravel_tex_const:nn { set_page_int } { 82 }
600 \__unravel_tex_const:nn { set_box_dimen } { 83 }
601 \__unravel_tex_const:nn { set_shape } { 84 }
602 \__unravel_tex_const:nn { def_code } { 85 }
603 \__unravel_tex_const:nn { def_family } { 86 }
604 \__unravel_tex_const:nn { set_font } { 87 }
605 \__unravel_tex_const:nn { def_font } { 88 }
606 \__unravel_tex_const:nn { register } { 89 }
607 \__unravel_tex_const:nn { max_internal } { 89 }
608 \__unravel_tex_const:nn { advance } { 90 }
609 \__unravel_tex_const:nn { multiply } { 91 }
610 \__unravel_tex_const:nn { divide } { 92 }
611 \__unravel_tex_const:nn { prefix } { 93 }
612 \__unravel_tex_const:nn { let } { 94 }
613 \__unravel_tex_const:nn { shorthand_def } { 95 }
614 \__unravel_tex_const:nn { read_to_cs } { 96 }
615 \__unravel_tex_const:nn { def } { 97 }
616 \__unravel_tex_const:nn { set_box } { 98 }

```

```

617 \__unravel_tex_const:nn { hyph_data           } { 99 }
618 \__unravel_tex_const:nn { set_interaction     } { 100 }
619 \__unravel_tex_const:nn { letterspace_font   } { 101 }
620 \__unravel_tex_const:nn { pdf_copy_font      } { 102 }
621 \__unravel_tex_const:nn { max_command        } { 102 }
622 \__unravel_tex_const:nn { undefined_cs       } { 103 }
623 \__unravel_tex_const:nn { expand_after       } { 104 }
624 \__unravel_tex_const:nn { no_expand          } { 105 }
625 \__unravel_tex_const:nn { input              } { 106 }
626 \__unravel_tex_const:nn { if_test            } { 107 }
627 \__unravel_tex_const:nn { fi_or_else         } { 108 }
628 \__unravel_tex_const:nn { cs_name            } { 109 }
629 \__unravel_tex_const:nn { convert            } { 110 }
630 \__unravel_tex_const:nn { the                 } { 111 }
631 \__unravel_tex_const:nn { top_bot_mark       } { 112 }
632 \__unravel_tex_const:nn { call                } { 113 }
633 \__unravel_tex_const:nn { end_template       } { 117 }

```

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

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

- `case_shift` is shifted by 3983.
- `assign_toks` is shifted by `local_base=3412`.
- `assign_int` is shifted by `int_base=5263`.
- `assign_dimen` is shifted by `dimen_base=5830`.
- `assign_glue` and `assign_mu_glue` are shifted by `glue_base=2882`.
- `set_shape` is shifted (in 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`.

```

634 \__unravel_tex_primitive:nnn { relax           } { relax   } { 256 }
635 \__unravel_tex_primitive:nnn { span           } { tab_mark } { 256 }
636 \__unravel_tex_primitive:nnn { cr            } { car_ret  } { 257 }
637 \__unravel_tex_primitive:nnn { crcr          } { car_ret  } { 258 }
638 \__unravel_tex_primitive:nnn { par           } { par_end  } { 256 }
639 \__unravel_tex_primitive:nnn { end           } { stop     } { 0 }
640 \__unravel_tex_primitive:nnn { dump          } { stop     } { 1 }
641 \__unravel_tex_primitive:nnn { delimiter     } { delim_num } { 0 }
642 \__unravel_tex_primitive:nnn { char          } { char_num } { 0 }
643 \__unravel_tex_primitive:nnn { mathchar      } { math_char_num } { 0 }
644 \__unravel_tex_primitive:nnn { mark          } { mark     } { 0 }
645 \__unravel_tex_primitive:nnn { marks         } { mark     } { 5 }
646 \__unravel_tex_primitive:nnn { show         } { xray     } { 0 }
647 \__unravel_tex_primitive:nnn { showbox      } { xray     } { 1 }
648 \__unravel_tex_primitive:nnn { showthe      } { xray     } { 2 }

```

```

649 \_unravel_tex_primitive:nnn { showlists           } { xray } { 3 }
650 \_unravel_tex_primitive:nnn { showgroups         } { xray } { 4 }
651 \_unravel_tex_primitive:nnn { showtokens         } { xray } { 5 }
652 \_unravel_tex_primitive:nnn { showifs           } { xray } { 6 }
653 \_unravel_tex_primitive:nnn { box               } { make_box } { 0 }
654 \_unravel_tex_primitive:nnn { copy             } { make_box } { 1 }
655 \_unravel_tex_primitive:nnn { lastbox          } { make_box } { 2 }
656 \_unravel_tex_primitive:nnn { vsplit           } { make_box } { 3 }
657 \_unravel_tex_primitive:nnn { vtop              } { make_box } { 4 }
658 \_unravel_tex_primitive:nnn { vbox             } { make_box } { 5 }
659 \_unravel_tex_primitive:nnn { hbox             } { make_box } { 106 }
660 \_unravel_tex_primitive:nnn { moveright         } { hmove } { 0 }
661 \_unravel_tex_primitive:nnn { moveleft         } { hmove } { 1 }
662 \_unravel_tex_primitive:nnn { lower            } { vmove } { 0 }
663 \_unravel_tex_primitive:nnn { raise            } { vmove } { 1 }
664 \_unravel_tex_primitive:nnn { unhbox           } { un_hbox } { 0 }
665 \_unravel_tex_primitive:nnn { unhcopy          } { un_hbox } { 1 }
666 \_unravel_tex_primitive:nnn { unvbox           } { un_vbox } { 0 }
667 \_unravel_tex_primitive:nnn { unvcopy          } { un_vbox } { 1 }
668 \_unravel_tex_primitive:nnn { pagediscards      } { un_vbox } { 2 }
669 \_unravel_tex_primitive:nnn { splitdiscards     } { un_vbox } { 3 }
670 \_unravel_tex_primitive:nnn { unpenalty         } { remove_item } { 12 }
671 \_unravel_tex_primitive:nnn { unkern            } { remove_item } { 11 }
672 \_unravel_tex_primitive:nnn { unskip           } { remove_item } { 10 }
673 \_unravel_tex_primitive:nnn { hfil             } { hskip } { 0 }
674 \_unravel_tex_primitive:nnn { hfill           } { hskip } { 1 }
675 \_unravel_tex_primitive:nnn { hss              } { hskip } { 2 }
676 \_unravel_tex_primitive:nnn { hfilneg          } { hskip } { 3 }
677 \_unravel_tex_primitive:nnn { hskip           } { hskip } { 4 }
678 \_unravel_tex_primitive:nnn { vfil             } { vskip } { 0 }
679 \_unravel_tex_primitive:nnn { vfill           } { vskip } { 1 }
680 \_unravel_tex_primitive:nnn { vss              } { vskip } { 2 }
681 \_unravel_tex_primitive:nnn { vfilneg          } { vskip } { 3 }
682 \_unravel_tex_primitive:nnn { vskip           } { vskip } { 4 }
683 \_unravel_tex_primitive:nnn { mskip           } { mskip } { 5 }
684 \_unravel_tex_primitive:nnn { kern             } { kern } { 1 }
685 \_unravel_tex_primitive:nnn { mkern            } { mkern } { 99 }
686 \_unravel_tex_primitive:nnn { shipout          } { leader_ship } { 99 }
687 \_unravel_tex_primitive:nnn { leaders          } { leader_ship } { 100 }
688 \_unravel_tex_primitive:nnn { cleaders         } { leader_ship } { 101 }
689 \_unravel_tex_primitive:nnn { xleaders        } { leader_ship } { 102 }
690 \_unravel_tex_primitive:nnn { halign           } { halign } { 0 }
691 \_unravel_tex_primitive:nnn { valign         } { valign } { 0 }
692 \_unravel_tex_primitive:nnn { beginL          } { valign } { 4 }
693 \_unravel_tex_primitive:nnn { endL           } { valign } { 5 }
694 \_unravel_tex_primitive:nnn { beginR         } { valign } { 8 }
695 \_unravel_tex_primitive:nnn { endR           } { valign } { 9 }
696 \_unravel_tex_primitive:nnn { noalign        } { no_align } { 0 }
697 \_unravel_tex_primitive:nnn { vrule          } { vrule } { 0 }
698 \_unravel_tex_primitive:nnn { hrule          } { hrule } { 0 }

```

```

699 \__unravel_tex_primitive:nnn { insert           } { insert } { 0 }
700 \__unravel_tex_primitive:nnn { vadjust          } { vadjust } { 0 }
701 \__unravel_tex_primitive:nnn { ignorespaces   } { ignore_spaces } { 0 }
702 \__unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
703 \__unravel_tex_primitive:nnn { aftergroup    } { after_group } { 0 }
704 \__unravel_tex_primitive:nnn { penalty       } { break_penalty } { 0 }
705 \__unravel_tex_primitive:nnn { indent        } { start_par } { 1 }
706 \__unravel_tex_primitive:nnn { noindent      } { start_par } { 0 }
707 \__unravel_tex_primitive:nnn { quitvmode     } { start_par } { 2 }
708 \__unravel_tex_primitive:nnn { /              } { ital_corr } { 0 }
709 \__unravel_tex_primitive:nnn { accent          } { accent } { 0 }
710 \__unravel_tex_primitive:nnn { mathaccent       } { math_accent } { 0 }
711 \__unravel_tex_primitive:nnn { -                  } { discretionary } { 1 }
712 \__unravel_tex_primitive:nnn { discretionary    } { discretionary } { 0 }
713 \__unravel_tex_primitive:nnn { eqno              } { eq_no } { 0 }
714 \__unravel_tex_primitive:nnn { leqno            } { eq_no } { 1 }
715 \__unravel_tex_primitive:nnn { left              } { left_right } { 30 }
716 \__unravel_tex_primitive:nnn { right             } { left_right } { 31 }
717 \__unravel_tex_primitive:nnn { middle            } { left_right } { 17 }
718 \__unravel_tex_primitive:nnn { mathord           } { math_comp } { 16 }
719 \__unravel_tex_primitive:nnn { mathop            } { math_comp } { 17 }
720 \__unravel_tex_primitive:nnn { mathbin           } { math_comp } { 18 }
721 \__unravel_tex_primitive:nnn { mathrel           } { math_comp } { 19 }
722 \__unravel_tex_primitive:nnn { mathopen          } { math_comp } { 20 }
723 \__unravel_tex_primitive:nnn { mathclose         } { math_comp } { 21 }
724 \__unravel_tex_primitive:nnn { mathpunct         } { math_comp } { 22 }
725 \__unravel_tex_primitive:nnn { mathinner         } { math_comp } { 23 }
726 \__unravel_tex_primitive:nnn { underline         } { math_comp } { 26 }
727 \__unravel_tex_primitive:nnn { overline          } { math_comp } { 27 }
728 \__unravel_tex_primitive:nnn { displaylimits     } { limit_switch } { 0 }
729 \__unravel_tex_primitive:nnn { limits            } { limit_switch } { 1 }
730 \__unravel_tex_primitive:nnn { nolimits          } { limit_switch } { 2 }
731 \__unravel_tex_primitive:nnn { above             } { above } { 0 }
732 \__unravel_tex_primitive:nnn { over              } { above } { 1 }
733 \__unravel_tex_primitive:nnn { atop              } { above } { 2 }
734 \__unravel_tex_primitive:nnn { abovewithdelims   } { above } { 3 }
735 \__unravel_tex_primitive:nnn { overwithdelims    } { above } { 4 }
736 \__unravel_tex_primitive:nnn { atopwithdelims    } { above } { 5 }
737 \__unravel_tex_primitive:nnn { displaystyle     } { math_style } { 0 }
738 \__unravel_tex_primitive:nnn { textstyle        } { math_style } { 2 }
739 \__unravel_tex_primitive:nnn { scriptstyle      } { math_style } { 4 }
740 \__unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
741 \__unravel_tex_primitive:nnn { mathchoice        } { math_choice } { 0 }
742 \__unravel_tex_primitive:nnn { nonscript         } { non_script } { 0 }
743 \__unravel_tex_primitive:nnn { vcenter          } { vcenter } { 0 }
744 \__unravel_tex_primitive:nnn { lowercase        } { case_shift } { 256 }
745 \__unravel_tex_primitive:nnn { uppercase        } { case_shift } { 512 }
746 \__unravel_tex_primitive:nnn { message          } { message } { 0 }
747 \__unravel_tex_primitive:nnn { errmessage       } { message } { 1 }
748 \__unravel_tex_primitive:nnn { openout          } { extension } { 0 }

```

```

749 \__unravel_tex_primitive:nnn { write } { extension } { 1 }
750 \__unravel_tex_primitive:nnn { closeout } { extension } { 2 }
751 \__unravel_tex_primitive:nnn { special } { extension } { 3 }
752 \__unravel_tex_primitive:nnn { immediate } { extension } { 4 }
753 \__unravel_tex_primitive:nnn { setlanguage } { extension } { 5 }
754 \__unravel_tex_primitive:nnn { pdfliteral } { extension } { 6 }
755 \__unravel_tex_primitive:nnn { pdfobj } { extension } { 7 }
756 \__unravel_tex_primitive:nnn { pdfrefobj } { extension } { 8 }
757 \__unravel_tex_primitive:nnn { pdfxform } { extension } { 9 }
758 \__unravel_tex_primitive:nnn { pdfrefxform } { extension } { 10 }
759 \__unravel_tex_primitive:nnn { pdfximage } { extension } { 11 }
760 \__unravel_tex_primitive:nnn { pdfrefximage } { extension } { 12 }
761 \__unravel_tex_primitive:nnn { pdfannot } { extension } { 13 }
762 \__unravel_tex_primitive:nnn { pdfstartlink } { extension } { 14 }
763 \__unravel_tex_primitive:nnn { pdfendlink } { extension } { 15 }
764 \__unravel_tex_primitive:nnn { pdfoutline } { extension } { 16 }
765 \__unravel_tex_primitive:nnn { pdfdest } { extension } { 17 }
766 \__unravel_tex_primitive:nnn { pdfthread } { extension } { 18 }
767 \__unravel_tex_primitive:nnn { pdfstartthread } { extension } { 19 }
768 \__unravel_tex_primitive:nnn { pdfendthread } { extension } { 20 }
769 \__unravel_tex_primitive:nnn { pdfsavepos } { extension } { 21 }
770 \__unravel_tex_primitive:nnn { pdfinfo } { extension } { 22 }
771 \__unravel_tex_primitive:nnn { pdfcatalog } { extension } { 23 }
772 \__unravel_tex_primitive:nnn { pdfnames } { extension } { 24 }
773 \__unravel_tex_primitive:nnn { pdffontattr } { extension } { 25 }
774 \__unravel_tex_primitive:nnn { pdfincludechars } { extension } { 26 }
775 \__unravel_tex_primitive:nnn { pdfmapfile } { extension } { 27 }
776 \__unravel_tex_primitive:nnn { pdfmapline } { extension } { 28 }
777 \__unravel_tex_primitive:nnn { pdftrailer } { extension } { 29 }
778 \__unravel_tex_primitive:nnn { pdfresettimer } { extension } { 30 }
779 \__unravel_tex_primitive:nnn { pdffontexpand } { extension } { 31 }
780 \__unravel_tex_primitive:nnn { pdfsetrandomseed } { extension } { 32 }
781 \__unravel_tex_primitive:nnn { pdfsnaprefpoint } { extension } { 33 }
782 \__unravel_tex_primitive:nnn { pdfsnapy } { extension } { 34 }
783 \__unravel_tex_primitive:nnn { pdfsnapycomp } { extension } { 35 }
784 \__unravel_tex_primitive:nnn { pdfglyphtounicode } { extension } { 36 }
785 \__unravel_tex_primitive:nnn { pdfcolorstack } { extension } { 37 }
786 \__unravel_tex_primitive:nnn { pdfsetmatrix } { extension } { 38 }
787 \__unravel_tex_primitive:nnn { pdfsave } { extension } { 39 }
788 \__unravel_tex_primitive:nnn { pdfrestore } { extension } { 40 }
789 \__unravel_tex_primitive:nnn { pdfnobuiltintounicode } { extension } { 41 }
790 \__unravel_tex_primitive:nnn { openin } { in_stream } { 1 }
791 \__unravel_tex_primitive:nnn { closein } { in_stream } { 0 }
792 \__unravel_tex_primitive:nnn { begingroup } { begin_group } { 0 }
793 \__unravel_tex_primitive:nnn { endgroup } { end_group } { 0 }
794 \__unravel_tex_primitive:nnn { omit } { omit } { 0 }
795 \__unravel_tex_primitive:nnn { ~ } { ex_space } { 0 }
796 \__unravel_tex_primitive:nnn { noboundary } { no_boundary } { 0 }
797 \__unravel_tex_primitive:nnn { radical } { radical } { 0 }
798 \__unravel_tex_primitive:nnn { endcsname } { end_cs_name } { 0 }

```

```

799 \__unravel_tex_primitive:nnn { lastpenalty           } { last_item } { 0 }
800 \__unravel_tex_primitive:nnn { lastkern             } { last_item } { 1 }
801 \__unravel_tex_primitive:nnn { lastskip           } { last_item } { 2 }
802 \__unravel_tex_primitive:nnn { lastnodetype     } { last_item } { 3 }
803 \__unravel_tex_primitive:nnn { inputlineno      } { last_item } { 4 }
804 \__unravel_tex_primitive:nnn { badness           } { last_item } { 5 }
805 \__unravel_tex_primitive:nnn { pdftexversion    } { last_item } { 6 }
806 \__unravel_tex_primitive:nnn { pdflastobj       } { last_item } { 7 }
807 \__unravel_tex_primitive:nnn { pdflastxform     } { last_item } { 8 }
808 \__unravel_tex_primitive:nnn { pdflastximage    } { last_item } { 9 }
809 \__unravel_tex_primitive:nnn { pdflastximagepages } { last_item } { 10 }
810 \__unravel_tex_primitive:nnn { pdflastannot       } { last_item } { 11 }
811 \__unravel_tex_primitive:nnn { pdflastxpos        } { last_item } { 12 }
812 \__unravel_tex_primitive:nnn { pdflastypos        } { last_item } { 13 }
813 \__unravel_tex_primitive:nnn { pdfretval          } { last_item } { 14 }
814 \__unravel_tex_primitive:nnn { pdflastximagecolordepth } { last_item } { 15 }
815 \__unravel_tex_primitive:nnn { pdfelapsedtime     } { last_item } { 16 }
816 \__unravel_tex_primitive:nnn { pdfshellescape    } { last_item } { 17 }
817 \__unravel_tex_primitive:nnn { pdfrandomseed     } { last_item } { 18 }
818 \__unravel_tex_primitive:nnn { pdflastlink       } { last_item } { 19 }
819 \__unravel_tex_primitive:nnn { eTeXversion        } { last_item } { 20 }
820 \__unravel_tex_primitive:nnn { currentgrouplevel  } { last_item } { 21 }
821 \__unravel_tex_primitive:nnn { currentgrouptype  } { last_item } { 22 }
822 \__unravel_tex_primitive:nnn { currentiflevel    } { last_item } { 23 }
823 \__unravel_tex_primitive:nnn { currentiftype     } { last_item } { 24 }
824 \__unravel_tex_primitive:nnn { currentifbranch   } { last_item } { 25 }
825 \__unravel_tex_primitive:nnn { gluestretchorder  } { last_item } { 26 }
826 \__unravel_tex_primitive:nnn { glueshrinkorder   } { last_item } { 27 }
827 \__unravel_tex_primitive:nnn { fontcharwd       } { last_item } { 28 }
828 \__unravel_tex_primitive:nnn { fontcharht       } { last_item } { 29 }
829 \__unravel_tex_primitive:nnn { fontcharwdp      } { last_item } { 30 }
830 \__unravel_tex_primitive:nnn { fontcharic       } { last_item } { 31 }
831 \__unravel_tex_primitive:nnn { parshapelength    } { last_item } { 32 }
832 \__unravel_tex_primitive:nnn { parshapeindent    } { last_item } { 33 }
833 \__unravel_tex_primitive:nnn { parshapedimen     } { last_item } { 34 }
834 \__unravel_tex_primitive:nnn { gluestretch       } { last_item } { 35 }
835 \__unravel_tex_primitive:nnn { glueshrink        } { last_item } { 36 }
836 \__unravel_tex_primitive:nnn { mutoglue         } { last_item } { 37 }
837 \__unravel_tex_primitive:nnn { gluetomu         } { last_item } { 38 }
838 \__unravel_tex_primitive:nnn { numexpr          } { last_item } { 39 }
839 \__unravel_tex_primitive:nnn { dimexpr          } { last_item } { 40 }
840 \__unravel_tex_primitive:nnn { glueexpr         } { last_item } { 41 }
841 \__unravel_tex_primitive:nnn { muexpr           } { last_item } { 42 }
842 \__unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
843 \__unravel_tex_primitive:nnn { output           } { assign_toks } { 1 }
844 \__unravel_tex_primitive:nnn { everypar         } { assign_toks } { 2 }
845 \__unravel_tex_primitive:nnn { everymath        } { assign_toks } { 3 }
846 \__unravel_tex_primitive:nnn { everydisplay     } { assign_toks } { 4 }
847 \__unravel_tex_primitive:nnn { everyhbox       } { assign_toks } { 5 }
848 \__unravel_tex_primitive:nnn { everyvbox       } { assign_toks } { 6 }

```

```

849 \_unravel_tex_primitive:nnn { everyjob           } { assign_toks } { 7 }
850 \_unravel_tex_primitive:nnn { everycr           } { assign_toks } { 8 }
851 \_unravel_tex_primitive:nnn { errhelp         } { assign_toks } { 9 }
852 \_unravel_tex_primitive:nnn { pdfpagesattr    } { assign_toks } { 10 }
853 \_unravel_tex_primitive:nnn { pdfpageattr     } { assign_toks } { 11 }
854 \_unravel_tex_primitive:nnn { pdfpageresources } { assign_toks } { 12 }
855 \_unravel_tex_primitive:nnn { pdfpkmode       } { assign_toks } { 13 }
856 \_unravel_tex_primitive:nnn { everyeof        } { assign_toks } { 14 }
857 \_unravel_tex_primitive:nnn { pretolerance    } { assign_int  } { 0 }
858 \_unravel_tex_primitive:nnn { tolerance       } { assign_int  } { 1 }
859 \_unravel_tex_primitive:nnn { linepenalty     } { assign_int  } { 2 }
860 \_unravel_tex_primitive:nnn { hyphenpenalty   } { assign_int  } { 3 }
861 \_unravel_tex_primitive:nnn { exhyphenpenalty } { assign_int  } { 4 }
862 \_unravel_tex_primitive:nnn { clubpenalty     } { assign_int  } { 5 }
863 \_unravel_tex_primitive:nnn { widowpenalty    } { assign_int  } { 6 }
864 \_unravel_tex_primitive:nnn { displaywidowpenalty } { assign_int  } { 7 }
865 \_unravel_tex_primitive:nnn { brokenpenalty   } { assign_int  } { 8 }
866 \_unravel_tex_primitive:nnn { binoppenalty    } { assign_int  } { 9 }
867 \_unravel_tex_primitive:nnn { relpenalty      } { assign_int  } { 10 }
868 \_unravel_tex_primitive:nnn { predisplaypenalty } { assign_int  } { 11 }
869 \_unravel_tex_primitive:nnn { postdisplaypenalty } { assign_int  } { 12 }
870 \_unravel_tex_primitive:nnn { interlinepenalty } { assign_int  } { 13 }
871 \_unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int  } { 14 }
872 \_unravel_tex_primitive:nnn { finalhyphendemerits } { assign_int  } { 15 }
873 \_unravel_tex_primitive:nnn { adjdemerits     } { assign_int  } { 16 }
874 \_unravel_tex_primitive:nnn { mag              } { assign_int  } { 17 }
875 \_unravel_tex_primitive:nnn { delimiterfactor  } { assign_int  } { 18 }
876 \_unravel_tex_primitive:nnn { looseness       } { assign_int  } { 19 }
877 \_unravel_tex_primitive:nnn { time            } { assign_int  } { 20 }
878 \_unravel_tex_primitive:nnn { day             } { assign_int  } { 21 }
879 \_unravel_tex_primitive:nnn { month           } { assign_int  } { 22 }
880 \_unravel_tex_primitive:nnn { year            } { assign_int  } { 23 }
881 \_unravel_tex_primitive:nnn { showboxbreadth } { assign_int  } { 24 }
882 \_unravel_tex_primitive:nnn { showboxdepth   } { assign_int  } { 25 }
883 \_unravel_tex_primitive:nnn { hbadness       } { assign_int  } { 26 }
884 \_unravel_tex_primitive:nnn { vbadness       } { assign_int  } { 27 }
885 \_unravel_tex_primitive:nnn { pausing        } { assign_int  } { 28 }
886 \_unravel_tex_primitive:nnn { tracingonline   } { assign_int  } { 29 }
887 \_unravel_tex_primitive:nnn { tracingmacros   } { assign_int  } { 30 }
888 \_unravel_tex_primitive:nnn { tracingstats    } { assign_int  } { 31 }
889 \_unravel_tex_primitive:nnn { tracingparagraphs } { assign_int  } { 32 }
890 \_unravel_tex_primitive:nnn { tracingpages    } { assign_int  } { 33 }
891 \_unravel_tex_primitive:nnn { tracingoutput   } { assign_int  } { 34 }
892 \_unravel_tex_primitive:nnn { tracinglostchars } { assign_int  } { 35 }
893 \_unravel_tex_primitive:nnn { tracingcommands } { assign_int  } { 36 }
894 \_unravel_tex_primitive:nnn { tracingrestores } { assign_int  } { 37 }
895 \_unravel_tex_primitive:nnn { uchyph         } { assign_int  } { 38 }
896 \_unravel_tex_primitive:nnn { outputpenalty   } { assign_int  } { 39 }
897 \_unravel_tex_primitive:nnn { maxdeadcycles   } { assign_int  } { 40 }
898 \_unravel_tex_primitive:nnn { hangafter      } { assign_int  } { 41 }

```



```

899 \_unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
900 \_unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
901 \_unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
902 \_unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
903 \_unravel_tex_primitive:nnn { defaultshyphenchar } { assign_int } { 46 }
904 \_unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
905 \_unravel_tex_primitive:nnn { endlinedchar } { assign_int } { 48 }
906 \_unravel_tex_primitive:nnn { newlinedchar } { assign_int } { 49 }
907 \_unravel_tex_primitive:nnn { language } { assign_int } { 50 }
908 \_unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
909 \_unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
910 \_unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
911 \_unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
912 \_unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
913 \_unravel_tex_primitive:nnn { pdfcompresslevel } { assign_int } { 56 }
914 \_unravel_tex_primitive:nnn { pdfdecimaldigits } { assign_int } { 57 }
915 \_unravel_tex_primitive:nnn { pdfmovechars } { assign_int } { 58 }
916 \_unravel_tex_primitive:nnn { pdfimageresolution } { assign_int } { 59 }
917 \_unravel_tex_primitive:nnn { pdfpkresolution } { assign_int } { 60 }
918 \_unravel_tex_primitive:nnn { pdfuniquerename } { assign_int } { 61 }
919 \_unravel_tex_primitive:nnn
920 { pdfoptionalwaysusepdfpagebox } { assign_int } { 62 }
921 \_unravel_tex_primitive:nnn
922 { pdfoptionpdfinclusionerrorlevel } { assign_int } { 63 }
923 \_unravel_tex_primitive:nnn
924 { pdfoptionpdfminorversion } { assign_int } { 64 }
925 \_unravel_tex_primitive:nnn { pdfminorversion } { assign_int } { 64 }
926 \_unravel_tex_primitive:nnn { pdfforcepagebox } { assign_int } { 65 }
927 \_unravel_tex_primitive:nnn { pdfpagebox } { assign_int } { 66 }
928 \_unravel_tex_primitive:nnn
929 { pdfinclusionerrorlevel } { assign_int } { 67 }
930 \_unravel_tex_primitive:nnn { pdfgamma } { assign_int } { 68 }
931 \_unravel_tex_primitive:nnn { pdfimagegamma } { assign_int } { 69 }
932 \_unravel_tex_primitive:nnn { pdfimagehicolor } { assign_int } { 70 }
933 \_unravel_tex_primitive:nnn { pdfimageapplygamma } { assign_int } { 71 }
934 \_unravel_tex_primitive:nnn { pdfadjustspacing } { assign_int } { 72 }
935 \_unravel_tex_primitive:nnn { pdfprotrudechars } { assign_int } { 73 }
936 \_unravel_tex_primitive:nnn { pdftracingfonts } { assign_int } { 74 }
937 \_unravel_tex_primitive:nnn { pdfobjcompresslevel } { assign_int } { 75 }
938 \_unravel_tex_primitive:nnn
939 { pdfadjustinterwordglue } { assign_int } { 76 }
940 \_unravel_tex_primitive:nnn { pdfprependkern } { assign_int } { 77 }
941 \_unravel_tex_primitive:nnn { pdfappendkern } { assign_int } { 78 }
942 \_unravel_tex_primitive:nnn { pdfgentounicode } { assign_int } { 79 }
943 \_unravel_tex_primitive:nnn { pdfdraftmode } { assign_int } { 80 }
944 \_unravel_tex_primitive:nnn { pdfinclusioncopyfonts } { assign_int } { 81 }
945 \_unravel_tex_primitive:nnn { tracingassigns } { assign_int } { 82 }
946 \_unravel_tex_primitive:nnn { tracinggroups } { assign_int } { 83 }
947 \_unravel_tex_primitive:nnn { tracingifs } { assign_int } { 84 }
948 \_unravel_tex_primitive:nnn { tracingscantokens } { assign_int } { 85 }

```

```

949 \_ unravel\_tex\_primitive:nnn { tracingnesting           } { assign\_int } { 86 }
950 \_ unravel\_tex\_primitive:nnn { predisplaydirection     } { assign\_int } { 87 }
951 \_ unravel\_tex\_primitive:nnn { lastlinefit          } { assign\_int } { 88 }
952 \_ unravel\_tex\_primitive:nnn { savingvdiscards     } { assign\_int } { 89 }
953 \_ unravel\_tex\_primitive:nnn { savinghyphcodes     } { assign\_int } { 90 }
954 \_ unravel\_tex\_primitive:nnn { TeXxETstate        } { assign\_int } { 91 }
955 \_ unravel\_tex\_primitive:nnn { parindent           } { assign\_dimen } { 0 }
956 \_ unravel\_tex\_primitive:nnn { mathsurround        } { assign\_dimen } { 1 }
957 \_ unravel\_tex\_primitive:nnn { lineskiplimit       } { assign\_dimen } { 2 }
958 \_ unravel\_tex\_primitive:nnn { hsize                } { assign\_dimen } { 3 }
959 \_ unravel\_tex\_primitive:nnn { vsize                } { assign\_dimen } { 4 }
960 \_ unravel\_tex\_primitive:nnn { maxdepth             } { assign\_dimen } { 5 }
961 \_ unravel\_tex\_primitive:nnn { splitmaxdepth        } { assign\_dimen } { 6 }
962 \_ unravel\_tex\_primitive:nnn { boxmaxdepth          } { assign\_dimen } { 7 }
963 \_ unravel\_tex\_primitive:nnn { hfuzz                 } { assign\_dimen } { 8 }
964 \_ unravel\_tex\_primitive:nnn { vfuzz                 } { assign\_dimen } { 9 }
965 \_ unravel\_tex\_primitive:nnn { delimitershortfall   } { assign\_dimen } { 10 }
966 \_ unravel\_tex\_primitive:nnn { nulldelimiterspace   } { assign\_dimen } { 11 }
967 \_ unravel\_tex\_primitive:nnn { scriptspace          } { assign\_dimen } { 12 }
968 \_ unravel\_tex\_primitive:nnn { predisplaysize       } { assign\_dimen } { 13 }
969 \_ unravel\_tex\_primitive:nnn { displaywidth         } { assign\_dimen } { 14 }
970 \_ unravel\_tex\_primitive:nnn { displayindent       } { assign\_dimen } { 15 }
971 \_ unravel\_tex\_primitive:nnn { overfullrule        } { assign\_dimen } { 16 }
972 \_ unravel\_tex\_primitive:nnn { hangindent           } { assign\_dimen } { 17 }
973 \_ unravel\_tex\_primitive:nnn { hoffset              } { assign\_dimen } { 18 }
974 \_ unravel\_tex\_primitive:nnn { voffset              } { assign\_dimen } { 19 }
975 \_ unravel\_tex\_primitive:nnn { emergencystretch     } { assign\_dimen } { 20 }
976 \_ unravel\_tex\_primitive:nnn { pdfhorigin           } { assign\_dimen } { 21 }
977 \_ unravel\_tex\_primitive:nnn { pdfvorigin           } { assign\_dimen } { 22 }
978 \_ unravel\_tex\_primitive:nnn { pdfpagewidth         } { assign\_dimen } { 23 }
979 \_ unravel\_tex\_primitive:nnn { pdfpageheight        } { assign\_dimen } { 24 }
980 \_ unravel\_tex\_primitive:nnn { pdflinkmargin        } { assign\_dimen } { 25 }
981 \_ unravel\_tex\_primitive:nnn { pdfdestmargin        } { assign\_dimen } { 26 }
982 \_ unravel\_tex\_primitive:nnn { pdfthreadmargin     } { assign\_dimen } { 27 }
983 \_ unravel\_tex\_primitive:nnn { pdffirstlineheight   } { assign\_dimen } { 28 }
984 \_ unravel\_tex\_primitive:nnn { pdfastlinedepth     } { assign\_dimen } { 29 }
985 \_ unravel\_tex\_primitive:nnn { pdfeachlineheight    } { assign\_dimen } { 30 }
986 \_ unravel\_tex\_primitive:nnn { pdfeachlinedepth     } { assign\_dimen } { 31 }
987 \_ unravel\_tex\_primitive:nnn { pdfignoreddimen     } { assign\_dimen } { 32 }
988 \_ unravel\_tex\_primitive:nnn { pdfpxdimen          } { assign\_dimen } { 33 }
989 \_ unravel\_tex\_primitive:nnn { lineskip             } { assign\_glue } { 0 }
990 \_ unravel\_tex\_primitive:nnn { baselineskip         } { assign\_glue } { 1 }
991 \_ unravel\_tex\_primitive:nnn { parskip              } { assign\_glue } { 2 }
992 \_ unravel\_tex\_primitive:nnn { abovedisplayskip     } { assign\_glue } { 3 }
993 \_ unravel\_tex\_primitive:nnn { belowdisplayskip     } { assign\_glue } { 4 }
994 \_ unravel\_tex\_primitive:nnn { abovedisplayshortskip } { assign\_glue } { 5 }
995 \_ unravel\_tex\_primitive:nnn { belowdisplayshortskip } { assign\_glue } { 6 }
996 \_ unravel\_tex\_primitive:nnn { leftskip             } { assign\_glue } { 7 }
997 \_ unravel\_tex\_primitive:nnn { rightskip            } { assign\_glue } { 8 }
998 \_ unravel\_tex\_primitive:nnn { topskip              } { assign\_glue } { 9 }

```

```

999 \__unravel_tex_primitive:nnn { splittopskip           } { assign_glue } { 10 }
1000 \__unravel_tex_primitive:nnn { tabskip                } { assign_glue } { 11 }
1001 \__unravel_tex_primitive:nnn { spaceskip           } { assign_glue } { 12 }
1002 \__unravel_tex_primitive:nnn { xspaceskip        } { assign_glue } { 13 }
1003 \__unravel_tex_primitive:nnn { parfillskip       } { assign_glue } { 14 }
1004 \__unravel_tex_primitive:nnn { thinmuskip        } { assign_mu_glue } { 15 }
1005 \__unravel_tex_primitive:nnn { medmuskip         } { assign_mu_glue } { 16 }
1006 \__unravel_tex_primitive:nnn { thickmuskip       } { assign_mu_glue } { 17 }
1007 \__unravel_tex_primitive:nnn { fontdimen        } { assign_font_dimen } { 0 }
1008 \__unravel_tex_primitive:nnn { hyphenchar        } { assign_font_int } { 0 }
1009 \__unravel_tex_primitive:nnn { skewchar          } { assign_font_int } { 1 }
1010 \__unravel_tex_primitive:nnn { lpcode              } { assign_font_int } { 2 }
1011 \__unravel_tex_primitive:nnn { rpcode              } { assign_font_int } { 3 }
1012 \__unravel_tex_primitive:nnn { efcodes              } { assign_font_int } { 4 }
1013 \__unravel_tex_primitive:nnn { tagcode            } { assign_font_int } { 5 }
1014 \__unravel_tex_primitive:nnn { pdfnoligatures     } { assign_font_int } { 6 }
1015 \__unravel_tex_primitive:nnn { knbscode            } { assign_font_int } { 7 }
1016 \__unravel_tex_primitive:nnn { stbscode            } { assign_font_int } { 8 }
1017 \__unravel_tex_primitive:nnn { shbscode            } { assign_font_int } { 9 }
1018 \__unravel_tex_primitive:nnn { knbccode           } { assign_font_int } { 10 }
1019 \__unravel_tex_primitive:nnn { knaccode            } { assign_font_int } { 11 }
1020 \__unravel_tex_primitive:nnn { spacefactor        } { set_aux } { 102 }
1021 \__unravel_tex_primitive:nnn { prevdepth          } { set_aux } { 1 }
1022 \__unravel_tex_primitive:nnn { prevgraf            } { set_prev_graf } { 0 }
1023 \__unravel_tex_primitive:nnn { pagegoal            } { set_page_dimen } { 0 }
1024 \__unravel_tex_primitive:nnn { pagetotal            } { set_page_dimen } { 1 }
1025 \__unravel_tex_primitive:nnn { pagestretch         } { set_page_dimen } { 2 }
1026 \__unravel_tex_primitive:nnn { pagefilstretch      } { set_page_dimen } { 3 }
1027 \__unravel_tex_primitive:nnn { pagefillstretch     } { set_page_dimen } { 4 }
1028 \__unravel_tex_primitive:nnn { pagefilllstretch    } { set_page_dimen } { 5 }
1029 \__unravel_tex_primitive:nnn { pageshrink          } { set_page_dimen } { 6 }
1030 \__unravel_tex_primitive:nnn { pagedepth            } { set_page_dimen } { 7 }
1031 \__unravel_tex_primitive:nnn { deadcycles           } { set_page_int } { 0 }
1032 \__unravel_tex_primitive:nnn { insertpenalties    } { set_page_int } { 1 }
1033 \__unravel_tex_primitive:nnn { interactionmode      } { set_page_int } { 2 }
1034 \__unravel_tex_primitive:nnn { wd                    } { set_box_dimen } { 1 }
1035 \__unravel_tex_primitive:nnn { dp                    } { set_box_dimen } { 2 }
1036 \__unravel_tex_primitive:nnn { ht                    } { set_box_dimen } { 3 }
1037 \__unravel_tex_primitive:nnn { parshape             } { set_shape } { 0 }
1038 \__unravel_tex_primitive:nnn { interlinepenalties   } { set_shape } { 1 }
1039 \__unravel_tex_primitive:nnn { clubpenalties        } { set_shape } { 2 }
1040 \__unravel_tex_primitive:nnn { widowpenalties        } { set_shape } { 3 }
1041 \__unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
1042 \__unravel_tex_primitive:nnn { catcode              } { def_code } { 0 }
1043 \__unravel_tex_primitive:nnn { lccode                } { def_code } { 256 }
1044 \__unravel_tex_primitive:nnn { uccode                } { def_code } { 512 }
1045 \__unravel_tex_primitive:nnn { sfcode                } { def_code } { 768 }
1046 \__unravel_tex_primitive:nnn { mathcode             } { def_code } { 1024 }
1047 \__unravel_tex_primitive:nnn { delcode              } { def_code } { 1591 }
1048 \__unravel_tex_primitive:nnn { textfont             } { def_family } { -48 }

```

```

1049 \__unravel_tex_primitive:nnn { scriptfont           } { def_family } { -32 }
1050 \__unravel_tex_primitive:nnn { scriptscriptfont     } { def_family } { -16 }
1051 \__unravel_tex_primitive:nnn { nullfont             } { set_font   } { 0 }
1052 \__unravel_tex_primitive:nnn { font              } { def_font   } { 0 }
1053 \__unravel_tex_primitive:nnn { count            } { register   } { 1 000 000 }
1054 \__unravel_tex_primitive:nnn { dimen              } { register   } { 2 000 000 }
1055 \__unravel_tex_primitive:nnn { skip              } { register   } { 3 000 000 }
1056 \__unravel_tex_primitive:nnn { muskip           } { register   } { 4 000 000 }
1057 \__unravel_tex_primitive:nnn { advance          } { advance    } { 0 }
1058 \__unravel_tex_primitive:nnn { multiply          } { multiply    } { 0 }
1059 \__unravel_tex_primitive:nnn { divide           } { divide     } { 0 }
1060 \__unravel_tex_primitive:nnn { long              } { prefix     } { 1 }
1061 \__unravel_tex_primitive:nnn { outer            } { prefix     } { 2 }
1062 \__unravel_tex_primitive:nnn { global           } { prefix     } { 4 }
1063 \__unravel_tex_primitive:nnn { protected        } { prefix     } { 8 }
1064 \__unravel_tex_primitive:nnn { let                } { let        } { 0 }
1065 \__unravel_tex_primitive:nnn { futurelet        } { let        } { 1 }
1066 \__unravel_tex_primitive:nnn { chardef          } { shorthand_def } { 0 }
1067 \__unravel_tex_primitive:nnn { mathchardef     } { shorthand_def } { 1 }
1068 \__unravel_tex_primitive:nnn { countdef         } { shorthand_def } { 2 }
1069 \__unravel_tex_primitive:nnn { dimendef        } { shorthand_def } { 3 }
1070 \__unravel_tex_primitive:nnn { skipdef         } { shorthand_def } { 4 }
1071 \__unravel_tex_primitive:nnn { muskipdef       } { shorthand_def } { 5 }
1072 \__unravel_tex_primitive:nnn { toksdef         } { shorthand_def } { 6 }
1073 \__unravel_tex_primitive:nnn { read             } { read_to_cs  } { 0 }
1074 \__unravel_tex_primitive:nnn { readline          } { read_to_cs  } { 1 }
1075 \__unravel_tex_primitive:nnn { def              } { def         } { 0 }
1076 \__unravel_tex_primitive:nnn { gdef            } { def         } { 1 }
1077 \__unravel_tex_primitive:nnn { edef            } { def         } { 2 }
1078 \__unravel_tex_primitive:nnn { xdef            } { def         } { 3 }
1079 \__unravel_tex_primitive:nnn { setbox          } { set_box     } { 0 }
1080 \__unravel_tex_primitive:nnn { hyphenation        } { hyph_data  } { 0 }
1081 \__unravel_tex_primitive:nnn { patterns          } { hyph_data  } { 1 }
1082 \__unravel_tex_primitive:nnn { batchmode        } { set_interaction } { 0 }
1083 \__unravel_tex_primitive:nnn { nonstopmode      } { set_interaction } { 1 }
1084 \__unravel_tex_primitive:nnn { scrollmode       } { set_interaction } { 2 }
1085 \__unravel_tex_primitive:nnn { errorstopmode    } { set_interaction } { 3 }
1086 \__unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1087 \__unravel_tex_primitive:nnn { pdfcopyfont    } { pdf_copy_font } { 0 }
1088 \__unravel_tex_primitive:nnn { undefined      } { undefined_cs } { 0 }
1089 \__unravel_tex_primitive:nnn { ndefined       } { undefined_cs } { 0 }
1090 \__unravel_tex_primitive:nnn { expandafter     } { expand_after } { 0 }
1091 \__unravel_tex_primitive:nnn { unless         } { expand_after } { 1 }
1092 \__unravel_tex_primitive:nnn { pdfprimitive   } { no_expand   } { 1 }
1093 \__unravel_tex_primitive:nnn { noexpand       } { no_expand   } { 0 }
1094 \__unravel_tex_primitive:nnn { input          } { input       } { 0 }
1095 \__unravel_tex_primitive:nnn { endinput       } { input       } { 1 }
1096 \__unravel_tex_primitive:nnn { scantokens     } { input       } { 2 }
1097 \__unravel_tex_primitive:nnn { if             } { if_test     } { 0 }
1098 \__unravel_tex_primitive:nnn { ifcat          } { if_test     } { 1 }

```

```

1099 \__unravel_tex_primitive:nnn { ifnum           } { if_test } { 2 }
1100 \__unravel_tex_primitive:nnn { ifdim           } { if_test } { 3 }
1101 \__unravel_tex_primitive:nnn { ifodd          } { if_test } { 4 }
1102 \__unravel_tex_primitive:nnn { ifvmode     } { if_test } { 5 }
1103 \__unravel_tex_primitive:nnn { ifhmode     } { if_test } { 6 }
1104 \__unravel_tex_primitive:nnn { ifmmode     } { if_test } { 7 }
1105 \__unravel_tex_primitive:nnn { ifinner     } { if_test } { 8 }
1106 \__unravel_tex_primitive:nnn { ifvoid      } { if_test } { 9 }
1107 \__unravel_tex_primitive:nnn { ifhbox      } { if_test } { 10 }
1108 \__unravel_tex_primitive:nnn { ifvbox      } { if_test } { 11 }
1109 \__unravel_tex_primitive:nnn { ifx          } { if_test } { 12 }
1110 \__unravel_tex_primitive:nnn { ifeof        } { if_test } { 13 }
1111 \__unravel_tex_primitive:nnn { iftrue      } { if_test } { 14 }
1112 \__unravel_tex_primitive:nnn { iffalse     } { if_test } { 15 }
1113 \__unravel_tex_primitive:nnn { ifcase      } { if_test } { 16 }
1114 \__unravel_tex_primitive:nnn { ifdefined   } { if_test } { 17 }
1115 \__unravel_tex_primitive:nnn { ifcsname    } { if_test } { 18 }
1116 \__unravel_tex_primitive:nnn { iffontchar  } { if_test } { 19 }
1117 \__unravel_tex_primitive:nnn { ifincsname  } { if_test } { 20 }
1118 \__unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1119 \__unravel_tex_primitive:nnn { ifpdfabsnum  } { if_test } { 22 }
1120 \__unravel_tex_primitive:nnn { ifpdfabsdim  } { if_test } { 23 }
1121 \__unravel_tex_primitive:nnn { fi          } { fi_or_else } { 2 }
1122 \__unravel_tex_primitive:nnn { else        } { fi_or_else } { 3 }
1123 \__unravel_tex_primitive:nnn { or          } { fi_or_else } { 4 }
1124 \__unravel_tex_primitive:nnn { csname      } { cs_name } { 0 }
1125 \__unravel_tex_primitive:nnn { number      } { convert } { 0 }
1126 \__unravel_tex_primitive:nnn { romannumeral } { convert } { 1 }
1127 \__unravel_tex_primitive:nnn { string      } { convert } { 2 }
1128 \__unravel_tex_primitive:nnn { meaning    } { convert } { 3 }
1129 \__unravel_tex_primitive:nnn { fontname    } { convert } { 4 }
1130 \__unravel_tex_primitive:nnn { eTeXrevision } { convert } { 5 }
1131 \__unravel_tex_primitive:nnn { pdftexrevision } { convert } { 6 }
1132 \__unravel_tex_primitive:nnn { pdftexbanner } { convert } { 7 }
1133 \__unravel_tex_primitive:nnn { pdffontname } { convert } { 8 }
1134 \__unravel_tex_primitive:nnn { pdffontobjnum } { convert } { 9 }
1135 \__unravel_tex_primitive:nnn { pdffontsize } { convert } { 10 }
1136 \__unravel_tex_primitive:nnn { pdfpageref } { convert } { 11 }
1137 \__unravel_tex_primitive:nnn { pdfxformname } { convert } { 12 }
1138 \__unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
1139 \__unravel_tex_primitive:nnn { pdfescapename } { convert } { 14 }
1140 \__unravel_tex_primitive:nnn { leftmarginkern } { convert } { 15 }
1141 \__unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
1142 \__unravel_tex_primitive:nnn { pdfstrcmp } { convert } { 17 }
1143 \__unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
1144 \__unravel_tex_primitive:nnn { pdfescapehex } { convert } { 19 }
1145 \__unravel_tex_primitive:nnn { pdfunescapehex } { convert } { 20 }
1146 \__unravel_tex_primitive:nnn { pdfcreationdate } { convert } { 21 }
1147 \__unravel_tex_primitive:nnn { pdffilemoddate } { convert } { 22 }
1148 \__unravel_tex_primitive:nnn { pdffilesize } { convert } { 23 }

```

```

1149 \__unravel_tex_primitive:nnn { pdfmdfivesum      } { convert } { 24 }
1150 \__unravel_tex_primitive:nnn { pdffiledump      } { convert } { 25 }
1151 \__unravel_tex_primitive:nnn { pdfmatch        } { convert } { 26 }
1152 \__unravel_tex_primitive:nnn { pdflastmatch   } { convert } { 27 }
1153 \__unravel_tex_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
1154 \__unravel_tex_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
1155 \__unravel_tex_primitive:nnn { pdfinsertht   } { convert } { 30 }
1156 \__unravel_tex_primitive:nnn { pdfximagebbox  } { convert } { 31 }
1157 \__unravel_tex_primitive:nnn { jobname        } { convert } { 32 }
1158 \__unravel_tex_primitive:nnn { the            } { the } { 0 }
1159 \__unravel_tex_primitive:nnn { unexpanded     } { the } { 1 }
1160 \__unravel_tex_primitive:nnn { detokenize      } { the } { 5 }
1161 \__unravel_tex_primitive:nnn { topmark        } { top_bot_mark } { 0 }
1162 \__unravel_tex_primitive:nnn { firstmark      } { top_bot_mark } { 1 }
1163 \__unravel_tex_primitive:nnn { botmark        } { top_bot_mark } { 2 }
1164 \__unravel_tex_primitive:nnn { splitfirstmark } { top_bot_mark } { 3 }
1165 \__unravel_tex_primitive:nnn { splitbotmark  } { top_bot_mark } { 4 }
1166 \__unravel_tex_primitive:nnn { topmarks      } { top_bot_mark } { 5 }
1167 \__unravel_tex_primitive:nnn { firstmarks    } { top_bot_mark } { 6 }
1168 \__unravel_tex_primitive:nnn { botmarks      } { top_bot_mark } { 7 }
1169 \__unravel_tex_primitive:nnn { splitfirstmarks } { top_bot_mark } { 8 }
1170 \__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.

```

1171 \cs_new_protected_nopar:Npn \__unravel_get_next:
1172 {
1173   \__unravel_input_if_empty:TF
1174     { \__unravel_exit:w }
1175     {
1176       \__unravel_input_gpop:N \l__unravel_head_gtl
1177       \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1178       \gtl_if_tl:NTF \l__unravel_head_gtl
1179         {
1180           \tl_set:Nx \l__unravel_head_tl

```

```

1181         { \gtl_head:N \l__unravel_head_gtl }
1182     }
1183     { \tl_clear:N \l__unravel_head_tl }
1184 }
1185 }
1186 \cs_new_protected_nopar:Npn \__unravel_get_next_aux:w
1187 { \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`.

```

1188 \cs_new_protected_nopar:Npn \__unravel_get_token:
1189 {
1190     \__unravel_get_next:
1191     \__unravel_set_cmd:
1192 }

```

(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.

```

1193 \cs_new_protected_nopar:Npn \__unravel_set_cmd:
1194 {
1195     \__unravel_set_cmd_aux_meaning:
1196     \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1197     { }
1198     {
1199         \__unravel_token_if_expandable:NTF \l__unravel_head_token
1200         {
1201             \token_if_macro:NTF \l__unravel_head_token
1202             { \__unravel_set_cmd_aux_macro: }
1203             { \__unravel_set_cmd_aux_unknown: }
1204         }
1205         {
1206             \token_if_cs:NTF \l__unravel_head_token
1207             { \__unravel_set_cmd_aux_cs: }
1208             { \__unravel_set_cmd_aux_char: }
1209         }
1210     }
1211 }

```

(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.

```

1212 \cs_new_protected_nopar:Npn \_unravel_set_cmd_aux_meaning:
1213 {
1214   \tl_set:Nx \l__unravel_head_meaning_tl
1215     {
1216       \exp_after:wN \_unravel_strip_escape:w
1217       \token_to_meaning:N \l__unravel_head_token
1218       \tl_to_str:n { : }
1219     }
1220   \tl_set:Nx \l__unravel_head_meaning_tl
1221     {
1222       \exp_after:wN \_unravel_set_cmd_aux_meaning:w
1223       \l__unravel_head_meaning_tl \q_stop
1224     }
1225 }
1226 \use:x
1227 {
1228   \cs_new:Npn \exp_not:N \_unravel_set_cmd_aux_meaning:w
1229     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1230 }

```

(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 that as the command and character integers.

```

1231 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nTF #1#2
1232 {
1233   \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1234     {
1235       \exp_last_unbraced:Nv \_unravel_set_cmd_aux_primitive:nn
1236         { c__unravel_tex_#1_tl }
1237       #2
1238     }
1239 }
1240 \cs_generate_variant:Nn \_unravel_set_cmd_aux_primitive:nTF { o }
1241 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nn #1#2
1242 {
1243   \int_set:Nn \l__unravel_head_cmd_int {#1}
1244   \int_set:Nn \l__unravel_head_char_int {#2}
1245 }

```

(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.

```

1246 \cs_new_protected_nopar:Npn \_unravel_set_cmd_aux_macro:
1247 {
1248   \int_set:Nn \l__unravel_head_cmd_int { \_unravel_tex_use:n { call } }

```



```

1249   \int_zero:N \l__unravel_head_char_int
1250   }

```

(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`.

```

1251 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_unknown:
1252   {
1253     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1254     \c__unravel_tex_relax_tl
1255     \msg_error:nxx { unravel } { unknown-primitive }
1256     { \l__unravel_head_meaning_tl }
1257   }

```

(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.

```

1258 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_cs:
1259   {
1260     \tl_if_in:NoTF \l__unravel_head_meaning_tl
1261     { \tl_to_str:n { elect-font } }
1262     {
1263       \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1264       \c__unravel_tex_nullfont_tl
1265     }
1266     { \__unravel_set_cmd_aux_numeric: }
1267   }

```

(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.

```

1268 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_numeric:
1269   {
1270     \tl_set:Nx \l__unravel_tmpa_tl
1271     {
1272       \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1273       \l__unravel_head_meaning_tl + 0
1274     }

```

```

1275 \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1276 \l__unravel_tmpa_tl \q_stop
1277 }
1278 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1279 {
1280 \if_int_compare:w '#1 < 'A \exp_stop_f:
1281 \exp_not:N \q_mark
1282 \exp_after:wN \use_i:nn
1283 \fi:
1284 #1 \__unravel_set_cmd_aux_numeric:N
1285 }
1286 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1287 {
1288 \str_case:nnF {#1}
1289 {
1290 { char } { \__unravel_set_cmd_aux_given:n { char_given } }
1291 { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1292 }
1293 {
1294 \__unravel_set_cmd_aux_primitive:nTF {#1}
1295 { }
1296 { \__unravel_set_cmd_aux_unknown: }
1297 \int_add:Nn \l__unravel_head_char_int { 100 000 }
1298 }
1299 \int_add:Nn \l__unravel_head_char_int {#2}
1300 }
1301 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1302 {
1303 \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1304 \int_zero:N \l__unravel_head_char_int
1305 }

```

(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 ‘.

```

1306 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_char:
1307 {
1308 \tl_set:Nx \l__unravel_head_meaning_tl
1309 { \token_to_meaning:N \l__unravel_head_token }
1310 \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1311 { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1312 \exp_after:wN \__unravel_set_cmd_aux_char:w
1313 \l__unravel_head_meaning_tl \q_stop
1314 \exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1315 { ‘ \__unravel_token_to_char:N \l__unravel_head_token }

```

```

1316 }
1317 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1318 {
1319   \int_set:Nn \l__unravel_head_cmd_int
1320   { \__unravel_tex_use:n { #1_char } }
1321 }

```

(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.

```

1322 \cs_new_nopar:Npn \__unravel_input_to_str:
1323 {
1324   \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1325   \__unravel_input_to_str_aux:n
1326 }
1327 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1328 { \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.

```

1329 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1330 {
1331   \int_compare:nNnTF \g__unravel_input_int = \c_zero
1332   { \use_i:nn }
1333   {
1334     \gtl_if_empty:cTF
1335     { g__unravel_input_ \int_use:N \g__unravel_input_int_gtl }
1336     {
1337       \int_gdecr:N \g__unravel_input_int
1338       \__unravel_input_if_empty:TF
1339     }
1340     {
1341       \__unravel_input_split:
1342       \use_ii:nn
1343     }
1344   }
1345 }

```

(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

```

1346 \cs_new_protected_nopar:Npn \__unravel_input_split:

```

```

1347 {
1348   \int_compare:nNnT \g__unravel_input_int = \c_one
1349   {
1350     \exp_args:Nc \__unravel_input_split_aux:N
1351     { g__unravel_input_1_gtl }
1352   }
1353 }
1354 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1355 {
1356   \gtl_if_tl:NT #1
1357   {
1358     \gtl_if_head_is_N_type:NT #1
1359     {
1360       \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1361       \exp_last_unbraced:Nx \__unravel_input_split_auxii:N
1362       { \tl_head:N \l__unravel_input_tmpa_tl }
1363     }
1364   }
1365 }
1366 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1367 {
1368   \token_if_parameter:NF #1
1369   {
1370     \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1371     { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1372     \group_begin:
1373     \cs_set:Npn \__unravel_input_split_auxiii:w
1374     ##1 \__unravel_input_split_end: { + 1 }
1375     \int_gset:Nn \g__unravel_input_int
1376     { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1377     \group_end:
1378     \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1379     \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1380   }
1381 }
1382 \cs_new_nopar:Npn \__unravel_input_split_end: { }
1383 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1384 #1 \__unravel_input_split_end:
1385 {
1386   \gtl_gclear_new:c
1387   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1388   \gtl_gset:cn
1389   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1390   \int_gdecr:N \g__unravel_input_tmpa_int
1391 }

```

(End definition for __unravel_input_split:.)

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

```

1392 \cs_new_protected_nopar:Npn \__unravel_input_gset:n

```

```

1393 {
1394   \int_gzero:N \g__unravel_input_int
1395   \__unravel_back_input:n
1396 }

```

(End definition for __unravel_input_gset:n.)

__unravel_input_get:N

```

1397 \cs_new_protected:Npn \__unravel_input_get:N #1
1398 {
1399   \__unravel_input_if_empty:TF
1400   { \gtl_set:Nn #1 { \q_no_value } }
1401   {
1402     \gtl_get_left:cN
1403     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1404   }
1405 }

```

(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.

```

1406 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1407 {
1408   \__unravel_input_if_empty:TF
1409   { \gtl_set:Nn #1 { \q_no_value } }
1410   {
1411     \gtl_gpop_left:cN
1412     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1413   }
1414 }

```

(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.

```

1415 \cs_new_protected_nopar:Npn \__unravel_input_merge:
1416 {
1417   \int_gdecr:N \g__unravel_input_int
1418   \gtl_gconcat:ccc
1419   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1420   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1421   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1422   \gtl_gclear:c
1423   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1424 }

```

(End definition for __unravel_input_merge:.)

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

```

1425 \prg_new_protected_conditional:Npnn \_unravel_input_gpop_item:N #1 { F }
1426   {
1427     \int_compare:nNnTF \g__unravel_input_int = \c_zero
1428       { \prg_return_false: }
1429       {
1430         \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1431           { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1432       }
1433   }
1434 \cs_new_protected:Npn \_unravel_input_gpop_item_aux:NN #1#2
1435   {
1436     \gtl_gpop_left_item:NNTF #1#2
1437     { \prg_return_true: }
1438     {
1439       \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero
1440         { \prg_return_false: }
1441         {
1442           \int_compare:nNnTF \g__unravel_input_int = \c_one
1443             { \prg_return_false: }
1444             {
1445               \_unravel_input_merge:
1446               \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1447                 {
1448                   g__unravel_input_
1449                   \int_use:N \g__unravel_input_int _gtl
1450                 }
1451               #2
1452             }
1453         }
1454     }
1455   }

```

(End definition for `_unravel_input_gpop_item:NNTF`.)

`_unravel_input_gpop_tl:N`

```

1456 \cs_new_protected:Npn \_unravel_input_gpop_tl:N #1
1457   { \tl_clear:N #1 \_unravel_input_gpop_tl_aux:N #1 }
1458 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:N #1
1459   {
1460     \int_compare:nNnF \g__unravel_input_int = \c_zero
1461       {
1462         \exp_args:Nc \_unravel_input_gpop_tl_aux:NN
1463           { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1464       }

```

```

1465 }
1466 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
1467 {
1468   \gtl_if_tl:NTF #1
1469   {
1470     \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1471     \gtl_gclear:N #1
1472     \int_gdecr:N \g__unravel_input_int
1473     \__unravel_input_gpop_tl_aux:N #2
1474   }
1475   {
1476     \int_compare:nNnTF \g__unravel_input_int > \c_one
1477     { \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero }
1478     { \use_i:nn }
1479     {
1480       \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1481       \gtl_gpop_left_tl:N #1
1482     }
1483     {
1484       \__unravel_input_merge:
1485       \__unravel_input_gpop_tl_aux:N #2
1486     }
1487   }
1488 }

```

(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.

```

1489 \cs_new_protected_nopar:Npn \__unravel_back_input:n
1490 {
1491   \int_gincr:N \g__unravel_input_int
1492   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1493   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1494 }
1495 \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.

```

1496 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1497 {
1498   \gtl_if_tl:NTF #1
1499   { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1500   {
1501     \gtl_gconcat:cNc
1502     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1503     #1
1504     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }

```

```

1505     }
1506   }

```

(End definition for _unravel_back_input_gt1:N.)

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

```

1507 \cs_new_protected_nopar:Npn \_unravel_back_input:
1508   { \_unravel_back_input_gt1:N \l_unravel_head_gt1 }

```

(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.

```

1509 \cs_new_protected_nopar:Npn \_unravel_back_input_tl_o:
1510   {
1511     \tl_set:Nx \l_unravel_tmpa_tl
1512       { \exp_args:NW \exp_not:o \l_unravel_head_tl }
1513     \_unravel_back_input:V \l_unravel_tmpa_tl
1514     \_unravel_print_done:x
1515     { \tl_to_str:N \l_unravel_head_tl = \tl_to_str:N \l_unravel_tmpa_tl }
1516   }

```

(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.

```

1517 \cs_new_protected_nopar:Npn \_unravel_insert_relax:
1518   {
1519     \_unravel_back_input:
1520     \gtl_set_eq:NN \l_unravel_head_gt1 \c_unravel_frozen_relax_gt1
1521     \_unravel_back_input:
1522     \_unravel_print_action:
1523   }

```

(End definition for _unravel_insert_relax:.)

`_unravel_insert_group_begin_error:`

```

1524 \cs_new_protected_nopar:Npn \_unravel_insert_group_begin_error:
1525   {
1526     \msg_error:nn { unravel } { missing-lbrace }
1527     \_unravel_back_input:
1528     \gtl_set_eq:NN \l_unravel_head_gt1 \c_group_begin_gt1
1529     \_unravel_back_input:
1530     \_unravel_print_action:
1531   }

```

(End definition for _unravel_insert_group_begin_error:.)

`_unravel_insert_dollar_error:`

```
1532 \cs_new_protected_nopar:Npn \_unravel_insert_dollar_error:
1533 {
1534   \_unravel_back_input:
1535   \_unravel_back_input:n { $ } % $
1536   \msg_error:nn { unravel } { missing-dollar }
1537   \_unravel_print_action:
1538 }
```

(End definition for _unravel_insert_dollar_error:.)

2.5.3 Macro calls

`_unravel_macro_prefix:N`
`_unravel_macro_parameter:N`
`_unravel_macro_replacement:N`

```
1539 \use:x
1540 {
1541   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:NN #1 }
1542   {
1543     \exp_not:n { \exp_after:wN \_unravel_macro_split_do:wN }
1544     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1545     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1546     \exp_not:N \q_stop
1547   }
1548   \exp_not:n { \cs_new:Npn \_unravel_macro_split_do:wN }
1549   \exp_not:n { #1 } \tl_to_str:n { : } \exp_not:n { #2 -> }
1550   \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1551   { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
1552 }
1553 \cs_new:Npn \_unravel_macro_prefix:N #1
1554 { \_unravel_macro_split_do:NN #1 \use_i:nnn }
1555 \cs_new:Npn \_unravel_macro_parameter:N #1
1556 { \_unravel_macro_split_do:NN #1 \use_ii:nnn }
1557 \cs_new:Npn \_unravel_macro_replacement:N #1
1558 { \_unravel_macro_split_do:NN #1 \use_iii:nnn }
```

(End definition for _unravel_macro_prefix:N, _unravel_macro_parameter:N, and _unravel_macro_replacement:N.)

`_unravel_macro_call:`
`_unravel_macro_call_safe:`
`_unravel_macro_call_quick:`
`_unravel_macro_call_quick_loop:NNN`
`_unravel_macro_call_quick_runaway:Nw`

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

```
1559 \cs_new_protected_nopar:Npn \_unravel_macro_call:
1560 {
1561   \bool_if:NTF \g_unravel_speedup_macros_bool
1562   {
1563     \tl_set:Nx \l_unravel_tmpa_tl
1564     { ^ \exp_after:wN \_unravel_macro_parameter:N \l_unravel_head_tl }
1565     \tl_if_in:NVTF \c_unravel_parameters_tl \l_unravel_tmpa_tl
1566     { \_unravel_macro_call_quick: } { \_unravel_macro_call_safe: }
```

```

1567     }
1568     { \_unravel_macro_call_safe: }
1569     \exp_args:NV \_unravel_back_input:o \l__unravel_head_tl
1570     \_unravel_print_done:x { \g__unravel_action_text_str }
1571   }
1572 \cs_new_protected_nopar:Npn \_unravel_macro_call_safe:
1573   {
1574     \_unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1575     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1576   }
1577 \cs_new_protected_nopar:Npn \_unravel_macro_call_quick:
1578   {
1579     \exp_after:wN \_unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1580     { ? \use_none_delimit_by_q_stop:w } \q_stop
1581   }
1582 \cs_new_protected:Npn \_unravel_macro_call_quick_loop:NNN #1#2#3
1583   {
1584     \use_none:n #2
1585     \_unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1586     { \_unravel_macro_call_quick_runaway:Nw #3 }
1587     \tl_put_right:Nx \l__unravel_head_tl
1588     { { \exp_not:V \l__unravel_tmpa_tl } }
1589     \_unravel_macro_call_quick_loop:NNN
1590     #3
1591   }
1592 \cs_new_protected:Npn \_unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1593   {
1594     \msg_error:nxxx { unravel } { runaway-macro-parameter }
1595     { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} }
1596   }

```

(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 mimick T_EX's structure, distinguishing macros from other commands (not quite sure why).

```

1597 \cs_new_protected_nopar:Npn \_unravel_expand:
1598   {
1599     \_unravel_set_action_text:
1600     \bool_if:NT \g__unravel_internal_debug_bool
1601       {
1602         \_unravel_set_cmd:
1603         \iow_term:x { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_int } }
1604       }
1605     \token_if_macro:NTF \l__unravel_head_token
1606     { \_unravel_macro_call: }
1607     { \_unravel_expand_nonmacro: }
1608   }

```

(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.

```
1609 \cs_new_protected_nopar:Npn \__unravel_expand_nonmacro:
1610   {
1611     \__unravel_set_cmd_aux_meaning:
1612     \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1613     {
1614       \cs_if_exist_use:cF
1615       { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1616       { \msg_error:nxx { unravel } { internal } { expandable } }
1617     }
1618     {
1619       \msg_error:nxx { unravel } { unknown-primitive }
1620       { \l__unravel_head_meaning_tl }
1621       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
1622       \__unravel_print_action:
1623     }
1624   }
```

(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.

```
1625 \cs_new_protected_nopar:Npn \__unravel_get_x_next:
1626   {
1627     \__unravel_get_next:
1628     \__unravel_token_if_expandable:NT \l__unravel_head_token
1629     {
1630       \__unravel_expand:
1631       \__unravel_get_x_next:
1632     }
1633   }
```

(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.

```
1634 \cs_new_protected_nopar:Npn \__unravel_get_x_or_protected:
1635   {
1636     \__unravel_get_next:
1637     \__unravel_token_if_protected:NF \l__unravel_head_token
1638     {
1639       \__unravel_expand:
1640       \__unravel_get_x_or_protected:
1641     }
1642   }
```

(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.

```
1643 \cs_new_protected_nopar:Npn \__unravel_get_x_non_blank:
1644   {
1645     \__unravel_get_x_next:
1646     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1647       { \__unravel_get_x_non_blank: }
1648   }
```

(End definition for `__unravel_get_x_non_blank:`.)

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

```
1649 \cs_new_protected_nopar:Npn \__unravel_get_x_non_relax:
1650   {
1651     \__unravel_get_x_next:
1652     \token_if_eq_meaning:NNT \l__unravel_head_token \scan_stop:
1653       { \__unravel_get_x_non_relax: }
1654     {
1655       \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1656         { \__unravel_get_x_non_relax: }
1657     }
1658   }
```

(End definition for `__unravel_get_x_non_relax:`.)

`__unravel_skip_optional_space:`

```
1659 \cs_new_protected_nopar:Npn \__unravel_skip_optional_space:
1660   {
1661     \__unravel_get_x_next:
1662     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1663       { \__unravel_back_input: }
1664   }
```

(End definition for `__unravel_skip_optional_space:`.)

`__unravel_scan_optional_equals:` See TeX's `scan_optional_equals`. In all cases we forcefully insert an equal sign in the output, because this sign is required, as `__unravel_scan_something_internal:n` leaves raw numbers in `\g__unravel_prev_input_seq`.

```
1665 \cs_new_protected_nopar:Npn \__unravel_scan_optional_equals:
1666   {
1667     \__unravel_get_x_non_blank:
1668     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1669       { \__unravel_prev_input:n { = } }
1670     {
1671       \__unravel_prev_input_silent:n { = }
1672       \__unravel_back_input:
1673     }
1674   }
```

(End definition for `_unravel_scan_optional_equals:`.)

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

```
1675 \cs_new_protected_nopar:Npn \_unravel_scan_left_brace:
1676   {
1677     \_unravel_get_x_non_relax:
1678     \token_if_eq_catcode:NNF \l\_unravel_head_token \c_group_begin_token
1679       { \_unravel_insert_group_begin_error: }
1680   }
```

(End definition for `_unravel_scan_left_brace:`.)

`_unravel_scan_keyword:n` The details of how \TeX looks for keywords are quite tricky to get right, in particular with respect to expansion, case-insensitivity, and spaces. We get rid of the case issue by requiring the keyword to be given in both cases, intertwined: for instance, `_unravel_scan_keyword:n { pPtT }`. Then loop through pairs of letters (which should be matching 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

1681 \cs_new_protected:Npn \_unravel_scan_keyword:n #1
1682   { \_unravel_scan_keyword:nTF {#1} { } { } }
1683 \prg_new_protected_conditional:Npnn \_unravel_scan_keyword:n #1
1684   { T , F , TF }
1685   {
1686     \seq_gput_right:NV \g\_unravel_prev_input_seq \c_empty_gt1
1687     \_unravel_scan_keyword_loop:NNN \c_true_bool
1688     #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1689   }
1690 \cs_new_protected:Npn \_unravel_scan_keyword_loop:NNN #1#2#3
1691   {
1692     \quark_if_recursion_tail_stop_do:nn {#2}
```

```

1693     { \_unravel_scan_keyword_true: }
1694 \quark_if_recursion_tail_stop_do:nn {#3}
1695     { \msg_error:nxx { unravel } { internal } { odd-keyword-length } }
1696 \_unravel_get_x_next:
1697 \_unravel_scan_keyword_test:NNTF #2#3
1698     {
1699         \_unravel_prev_input_gtl:N \l_unravel_head_gtl
1700         \_unravel_scan_keyword_loop:NNN \c_false_bool
1701     }
1702     {
1703         \token_if_eq_catcode:NMF \l_unravel_head_token \c_space_token
1704         { \_unravel_scan_keyword_false:w }
1705         \bool_if:NF #1
1706         { \_unravel_scan_keyword_false:w }
1707         \_unravel_scan_keyword_loop:NNN #1#2#3
1708     }
1709 }
1710 \prg_new_protected_conditional:Npnn \_unravel_scan_keyword_test:NN #1#2
1711 { TF }
1712 {
1713     \_unravel_gtl_if_head_is_definable:NTF \l_unravel_head_gtl
1714     { \prg_return_false: }
1715     {
1716         \str_if_eq_x:nnTF
1717         { \str_head:f { \gtl_to_str:N \l_unravel_head_gtl } } {#1}
1718         { \prg_return_true: }
1719         {
1720             \str_if_eq_x:nnTF
1721             { \str_head:f { \gtl_to_str:N \l_unravel_head_gtl } } {#2}
1722             { \prg_return_true: }
1723             { \prg_return_false: }
1724         }
1725     }
1726 }
1727 \cs_new_protected_nopar:Npn \_unravel_scan_keyword_true:
1728 {
1729     \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_tmpb_gtl
1730     \_unravel_prev_input:x { \gtl_to_str:N \l_unravel_tmpb_gtl }
1731     \prg_return_true:
1732 }
1733 \cs_new_protected_nopar:Npn \_unravel_scan_keyword_false:w
1734 #1 \q_recursion_stop
1735 {
1736     \_unravel_back_input:
1737     \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_tmpb_gtl
1738     \_unravel_back_input_gtl:N \l_unravel_tmpb_gtl
1739     \prg_return_false:
1740 }

```

(End definition for _unravel_scan_keyword:n.)

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

```
1741 \cs_new_protected_nopar:Npn \__unravel_scan_to:
1742   {
1743     \__unravel_scan_keyword:nF { tTo0 }
1744     {
1745       \msg_error:nn { unravel } { missing-to }
1746       \__unravel_prev_input:n { to }
1747     }
1748   }
```

(End definition for __unravel_scan_to:.)

`__unravel_scan_font_ident:` Find a font identifier.

```
1749 \cs_new_protected_nopar:Npn \__unravel_scan_font_ident:
1750   {
1751     \__unravel_get_x_non_blank:
1752     \__unravel_set_cmd:
1753     \int_case:nnF \l__unravel_head_cmd_int
1754       {
1755         { \__unravel_tex_use:n { def_font } }
1756         { \__unravel_prev_input:V \l__unravel_head_tl }
1757         { \__unravel_tex_use:n { letterspace_font } }
1758         { \__unravel_prev_input:V \l__unravel_head_tl }
1759         { \__unravel_tex_use:n { pdf_copy_font } }
1760         { \__unravel_prev_input:V \l__unravel_head_tl }
1761         { \__unravel_tex_use:n { set_font } }
1762         { \__unravel_prev_input:V \l__unravel_head_tl }
1763         { \__unravel_tex_use:n { def_family } }
1764         {
1765           \__unravel_prev_input:V \l__unravel_head_tl
1766           \__unravel_scan_int:
1767         }
1768       }
1769     {
1770       \msg_error:nn { unravel } { missing-font-id }
1771       \__unravel_back_input:
1772       \__unravel_prev_input:n { \__unravel_nullfont: }
1773     }
1774   }
```

(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`).

```
1775 \cs_new_protected_nopar:Npn \__unravel_scan_font_int:
1776   {
1777     \int_case:nnF \l__unravel_head_char_int
1778       {
1779         { 0 } { \__unravel_scan_font_ident: }
1780         { 1 } { \__unravel_scan_font_ident: }
1781         { 6 } { \__unravel_scan_font_ident: }
```

```

1782     }
1783     { \_unravel_scan_font_ident: \_unravel_scan_int: }
1784   }

```

(End definition for _unravel_scan_font_int:.)

_unravel_scan_font_dimen: Find operands for \fontdimen.

```

1785 \cs_new_protected_nopar:Npn \_unravel_scan_font_dimen:
1786   {
1787     \_unravel_scan_int:
1788     \_unravel_scan_font_ident:
1789   }

```

(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_get:nnNN (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).

```

1790 \cs_new_protected:Npn \_unravel_scan_something_internal:n #1
1791   {

```



```

1792 \__unravel_set_cmd:
1793 \__unravel_set_action_text:
1794 \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1795 \exp_after:wN \__unravel_scan_something_aux:nwn
1796   \l__unravel_tmpa_tl \q_stop {#1}
1797 }
1798 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1799 {
1800   \int_compare:nNnTF
1801     { ( #1 + \c_two ) / \c_four } > { ( #3 + \c_two ) / \c_four }
1802     {
1803       \__unravel_back_input:
1804       \msg_error:nn { unravel } { missing-something }
1805       \tl_clear:N \l__unravel_tmpa_tl
1806     }
1807     {
1808       \tl_if_empty:nF {#2}
1809       {
1810         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
1811         \__unravel_print_action:
1812         #2
1813         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1814       }
1815       \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl
1816     }
1817   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
1818   \__unravel_set_action_text:
1819   \__unravel_set_action_text:x
1820   { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:N \l__unravel_tmpa_tl }
1821   \int_compare:nNnF {#3} > { 3 } { \__unravel_print_action: }
1822   \int_gset:Nn \g__unravel_val_level_int {#1}
1823 }

```

(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.

```

1824 \cs_new_nopar:Npn \__unravel_thing_case:
1825 {
1826   \int_case:nnF \l__unravel_head_cmd_int
1827   {
1828     { 68 } { 0 } % char_given
1829     { 69 } { 0 } % math_given
1830     { 70 } { \__unravel_thing_last_item: } % last_item
1831     { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
1832     { 72 } { 5 } % assign_toks

```

```

1833     { 73 } { 0                } % assign_int
1834     { 74 } { 1                } % assign_dimen
1835     { 75 } { 2                } % assign_glue
1836     { 76 } { 3                } % assign_mu_glue
1837     { 77 } { 1 \_unravel_scan_font_dimen:  } % assign_font_dimen
1838     { 78 } { 0 \_unravel_scan_font_int:    } % assign_font_int
1839     { 79 } { \_unravel_thing_set_aux:      } % set_aux
1840     { 80 } { 0                } % set_prev_graf
1841     { 81 } { 1                } % set_page_dimen
1842     { 82 } { 0                } % set_page_int
1843     { 83 } { 1 \_unravel_scan_int:        } % set_box_dimen
1844     { 84 } { 0 \_unravel_scan_int:        } % set_shape
1845     { 85 } { 0 \_unravel_scan_int:        } % def_code
1846     { 86 } { 4 \_unravel_scan_int:        } % def_family
1847     { 87 } { 4                } % set_font
1848     { 88 } { 4                } % def_font
1849     { 89 } { \_unravel_thing_register:    } % register
1850   }
1851   { 8 }
1852 }
1853 \cs_new_nopar:Npn \_unravel_thing_set_aux:
1854   { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
1855 \cs_new_nopar:Npn \_unravel_thing_last_item:
1856   {
1857     \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
1858     {
1859       \int_case:nnF \l__unravel_head_char_int
1860       {
1861         { 1 } { 1 } % lastkern
1862         { 2 } { 2 } % lastskip
1863       }
1864       { 0 } % other integer parameters
1865     }
1866   {
1867     \int_case:nnF \l__unravel_head_char_int
1868     {
1869       { 26 } { 0 \_unravel_scan_normal_glue: } % gluestretchorder
1870       { 27 } { 0 \_unravel_scan_normal_glue: } % glueshrinkorder
1871       { 28 } % fontcharwd
1872       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1873       { 29 } % fontcharht
1874       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1875       { 30 } % fontchardp
1876       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1877       { 31 } % fontcharic
1878       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1879       { 32 } { 1 \_unravel_scan_int: } % parshapelength
1880       { 33 } { 1 \_unravel_scan_int: } % parshapeindent
1881       { 34 } { 1 \_unravel_scan_int: } % parshapedimen
1882       { 35 } { 1 \_unravel_scan_normal_glue: } % gluestretch

```

```

1883     { 36 } { 1 \_unravel_scan_normal_glue: } % glueshrink
1884     { 37 } { 2 \_unravel_scan_mu_glue: } % mutoglu
1885     { 38 } { 3 \_unravel_scan_normal_glue: } % glutomu
1886     { 39 } % numexpr
1887     { 0 \_unravel_scan_expr:N \_unravel_scan_int: }
1888     { 40 } % dimexpr
1889     { 1 \_unravel_scan_expr:N \_unravel_scan_normal_dimen: }
1890     { 41 } % glueexpr
1891     { 2 \_unravel_scan_expr:N \_unravel_scan_normal_glue: }
1892     { 42 } % muexpr
1893     { 3 \_unravel_scan_expr:N \_unravel_scan_mu_glue: }
1894   }
1895   { }
1896 }
1897 }
1898 \cs_new_nopar:Npn \_unravel_thing_register:
1899 {
1900   \int_eval:n { \l_unravel_head_char_int / 1 000 000 - 1 }
1901   \int_compare:nNnT { \tl_tail:V \l_unravel_head_char_int } = \c_zero
1902     { \_unravel_scan_int: }
1903 }

```

(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.

```

1904 \cs_new_protected:Npn \_unravel_scan_toks_register:
1905 {
1906   \int_compare:nNnT \l_unravel_head_char_int = \c_zero
1907     { \_unravel_scan_int: }
1908 }

```

(End definition for `_unravel_scan_toks_register:`.)

`_unravel_thing_use_get:nnNN` Given a level found #1 and a target level #2 (with #1 in [0, 3] or #1 and #2 in [4, 5]), turn the token list #3 into the desired level, and store the result in #4. This step may trigger \TeX errors, which should precisely match the expected ones.

```

1909 \cs_new_protected:Npn \_unravel_thing_use_get:nnNN #1#2#3#4
1910 {
1911   \int_compare:nNnTF {#2} < { 3 }
1912   {
1913     \int_compare:nNnT {#1} = { 3 }
1914     { \_unravel_tex_error:nV { incompatible-units } #3 }
1915     \tl_set:Nx #4
1916     {
1917       \int_case:nn { \int_min:nn {#1} {#2} }
1918       {
1919         { 0 } \int_eval:n
1920         { 1 } \dim_eval:n
1921         { 2 } \skip_eval:n

```

```

1922     }
1923     { \int_compare:nNnT {#1} = { 3 } \etex_mutogluue:D #3 }
1924   }
1925 }
1926 {
1927   \int_compare:nNnTF {#2} = { 3 }
1928   {
1929     \int_case:nnF {#1}
1930     {
1931       { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
1932       { 3 } { \tl_set:Nx #4 { \muskip_eval:n {#3} } }
1933     }
1934     {
1935       \__unravel_tex_error:nV { incompatible-units } #3
1936       \tl_set:Nx #4 { \muskip_eval:n { \etex_gluetomu:D #3 } }
1937     }
1938   }
1939   { \tl_set:Nx #4 { \__unravel_the:w #3 } }
1940 }
1941 }

```

(End definition for __unravel_thing_use_get:nnNN.)

```

\__unravel_scan_expr:N
\__unravel_scan_expr_aux:NN
\__unravel_scan_factor:N
1942 \cs_new_protected:Npn \__unravel_scan_expr:N #1
1943   { \__unravel_scan_expr_aux:NN #1 \c_false_bool }
1944 \cs_new_protected:Npn \__unravel_scan_expr_aux:NN #1#2
1945   {
1946     \__unravel_get_x_non_blank:
1947     \__unravel_scan_factor:N #1
1948     \__unravel_scan_expr_op:NN #1#2
1949   }
1950 \cs_new_protected:Npn \__unravel_scan_expr_op:NN #1#2
1951   {
1952     \__unravel_get_x_non_blank:
1953     \tl_case:NnF \l__unravel_head_tl
1954     {
1955       \c__unravel_plus_tl
1956       {
1957         \__unravel_prev_input:V \l__unravel_head_tl
1958         \__unravel_scan_expr_aux:NN #1#2
1959       }
1960       \c__unravel_minus_tl
1961       {
1962         \__unravel_prev_input:V \l__unravel_head_tl
1963         \__unravel_scan_expr_aux:NN #1#2
1964       }
1965       \c__unravel_times_tl
1966       {
1967         \__unravel_prev_input:V \l__unravel_head_tl

```

```

1968     \_unravel_get_x_non_blank:
1969     \_unravel_scan_factor:N \_unravel_scan_int:
1970     \_unravel_scan_expr_op:NN #1#2
1971   }
1972 \c__unravel_over_tl
1973   {
1974     \_unravel_prev_input:V \l__unravel_head_tl
1975     \_unravel_get_x_non_blank:
1976     \_unravel_scan_factor:N \_unravel_scan_int:
1977     \_unravel_scan_expr_op:NN #1#2
1978   }
1979 \c__unravel_rp_tl
1980   {
1981     \bool_if:NTF #2
1982     { \_unravel_prev_input:V \l__unravel_head_tl }
1983     { \_unravel_back_input: }
1984   }
1985 }
1986 {
1987   \bool_if:NTF #2
1988   {
1989     \msg_error:nn { unravel } { missing-rparen }
1990     \_unravel_back_input:
1991     \_unravel_prev_input:V \c__unravel_rp_tl
1992   }
1993   {
1994     \token_if_eq_meaning:NNF \l__unravel_head_token \scan_stop:
1995     { \_unravel_back_input: }
1996   }
1997 }
1998 }
1999 \cs_new_protected:Npn \_unravel_scan_factor:N #1
2000   {
2001     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2002     {
2003       \_unravel_prev_input:V \l__unravel_head_tl
2004       \_unravel_scan_expr_aux:NN #1 \c_true_bool
2005     }
2006     {
2007       \_unravel_back_input:
2008       #1
2009     }
2010   }

```

(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.

```

2011 \cs_new_protected_nopar:Npn \_unravel_scan_signs:
2012   {

```

```

2013 \__unravel_get_x_non_blank:
2014 \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
2015 {
2016   \__unravel_prev_input:V \l__unravel_head_tl
2017   \__unravel_scan_signs:
2018 }
2019 {
2020   \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
2021   {
2022     \__unravel_prev_input:V \l__unravel_head_tl
2023     \__unravel_scan_signs:
2024   }
2025 }
2026 }

```

(End definition for __unravel_scan_signs:.)

```

\__unravel_scan_int:
\__unravel_scan_int_char: 2027 \cs_new_protected_nopar:Npn \__unravel_scan_int:
\__unravel_scan_int_lq: 2028 {
\__unravel_scan_int_explicit:n 2029   \__unravel_scan_signs:
2030   \__unravel_set_cmd:
2031   \__unravel_cmd_if_internal:TF
2032   { \__unravel_scan_something_internal:n { 0 } }
2033   { \__unravel_scan_int_char: }
2034 }
2035 \cs_new_protected_nopar:Npn \__unravel_scan_int_char:
2036 {
2037   \tl_case:NnF \l__unravel_head_tl
2038   {
2039     \c__unravel_lq_tl { \__unravel_scan_int_lq: }
2040     \c__unravel_rq_tl
2041     {
2042       \__unravel_prev_input:V \l__unravel_head_tl
2043       \__unravel_get_x_next:
2044       \__unravel_scan_int_explicit:n { ' }
2045     }
2046     \c__unravel_dq_tl
2047     {
2048       \__unravel_prev_input:V \l__unravel_head_tl
2049       \__unravel_get_x_next:
2050       \__unravel_scan_int_explicit:n { " }
2051     }
2052   }
2053   { \__unravel_scan_int_explicit:n { } }
2054 }
2055 \cs_new_protected_nopar:Npn \__unravel_scan_int_lq:
2056 {
2057   \__unravel_get_next:
2058   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl

```

```

2059     {
2060       \tl_set:Nx \l__unravel_head_tl
2061       { \__unravel_token_to_char:N \l__unravel_head_token }
2062     }
2063     \tl_set:Nx \l__unravel_tmpa_tl
2064     { \int_eval:n { \exp_after:wN ‘ \l__unravel_head_tl } }
2065     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2066     \__unravel_print_action:x
2067     { ‘ \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2068     \__unravel_skip_optional_space:
2069   }
2070 \cs_new_protected:Npn \__unravel_scan_int_explicit:n #1
2071   {
2072     \if_int_compare:w \c_one
2073     < #1 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2074     \exp_after:wN \use_i:nn
2075     \else:
2076     \exp_after:wN \use_ii:nn
2077     \fi:
2078     {
2079       \__unravel_prev_input:V \l__unravel_head_tl
2080       \__unravel_get_x_next:
2081       \__unravel_scan_int_explicit:n {#1}
2082     }
2083     {
2084       \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2085       { \__unravel_back_input: }
2086     }
2087   }

```

(End definition for __unravel_scan_int:.)

__unravel_scan_normal_dimen:

```

2088 \cs_new_protected_nopar:Npn \__unravel_scan_normal_dimen:
2089   { \__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 μ and 3 if the unit must be μ (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`.

```

2090 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2091   {
2092     \__unravel_scan_signs:
2093     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2094     \__unravel_set_cmd:
2095     \__unravel_cmd_if_internal:TF

```

```

2096     {
2097         \int_compare:nNnTF {#1} = { 3 }
2098         { \__unravel_scan_something_internal:n { 3 } }
2099         { \__unravel_scan_something_internal:n { 1 } }
2100         \int_compare:nNnT \g__unravel_val_level_int = { 0 }
2101         { \__unravel_scan_dim_unit:nN {#1} #2 }
2102     }
2103     { \__unravel_scan_dimen_char:nN {#1} #2 }
2104     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2105     \__unravel_prev_input_silent:V \l__unravel_head_tl
2106 }
2107 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
2108 {
2109     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2110     { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2111     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
2112     {
2113         \__unravel_prev_input:n { . }
2114         \__unravel_scan_decimal_loop:
2115     }
2116     {
2117         \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl
2118         {
2119             \__unravel_back_input:
2120             \__unravel_scan_int:
2121             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2122             { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2123             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
2124             {
2125                 \__unravel_input_gpop:N \l__unravel_tmpb_gtl
2126                 \__unravel_prev_input:n { . }
2127                 \__unravel_scan_decimal_loop:
2128             }
2129         }
2130         {
2131             \__unravel_back_input:
2132             \__unravel_scan_int:
2133         }
2134     }
2135     \__unravel_scan_dim_unit:nN {#1} #2
2136 }
2137 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
2138 {
2139     \bool_if:NT #2
2140     {
2141         \__unravel_scan_keyword:nT { fFiIlL }
2142         {
2143             \__unravel_scan_inf_unit_loop:
2144             \__unravel_break:w
2145         }
2146     }

```



```

2146     }
2147     \__unravel_get_x_non_blank:
2148     \__unravel_set_cmd:
2149     \__unravel_cmd_if_internal:TF
2150     {
2151         \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2152         \__unravel_scan_something_internal:n {#1}
2153         \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2154         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2155         \__unravel_break:w
2156     }
2157     { \__unravel_back_input: }
2158     \int_compare:nNnT {#1} = { 3 }
2159     {
2160         \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2161         \msg_error:nn { unravel } { missing-mudim }
2162         \__unravel_break:w
2163     }
2164     \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2165     \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2166     \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2167     \__unravel_scan_keyword:nT { tTrRuUeE }
2168     { \__unravel_prepare_mag: }
2169     \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2170     \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2171     \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2172     \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }
2173     \__unravel_scan_keyword:nT { mMmM } { \__unravel_break:w }
2174     \__unravel_scan_keyword:nT { bBpP } { \__unravel_break:w }
2175     \__unravel_scan_keyword:nT { dDdD } { \__unravel_break:w }
2176     \__unravel_scan_keyword:nT { cCcC } { \__unravel_break:w }
2177     \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
2178     \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
2179     \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
2180     \__unravel_break_point:
2181 }
2182 \cs_new_protected_nopar:Npn \__unravel_scan_inf_unit_loop:
2183 { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
2184 \cs_new_protected_nopar:Npn \__unravel_scan_decimal_loop:
2185 {
2186     \__unravel_get_x_next:
2187     \tl_if_empty:NTF \l__unravel_head_tl
2188     { \use_ii:nn }
2189     { \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl }
2190     {
2191         \__unravel_prev_input:V \l__unravel_head_tl
2192         \__unravel_scan_decimal_loop:
2193     }
2194     {
2195         \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token

```

```

2196         { \_unravel_back_input: }
2197         \_unravel_prev_input_silent:n { ~ }
2198     }
2199 }

```

(End definition for _unravel_scan_dimen:nN.)

```

\_unravel_scan_normal_glue:
  \_unravel_scan_mu_glue:

```

```

2200 \cs_new_protected_nopar:Npn \_unravel_scan_normal_glue:
2201   { \_unravel_scan_glue:n { 2 } }
2202 \cs_new_protected_nopar:Npn \_unravel_scan_mu_glue:
2203   { \_unravel_scan_glue:n { 3 } }

```

(End definition for _unravel_scan_normal_glue: and _unravel_scan_mu_glue:.)

```

\_unravel_scan_glue:n

```

```

2204 \cs_new_protected:Npn \_unravel_scan_glue:n #1
2205   {
2206     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2207     \_unravel_scan_signs:
2208     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2209     \_unravel_set_cmd:
2210     \_unravel_cmd_if_internal:TF
2211     {
2212       \_unravel_scan_something_internal:n {#1}
2213       \int_case:nnF \g__unravel_val_level_int
2214         {
2215           { 0 } { \_unravel_scan_dimen:nN {#1} \c_false_bool }
2216           { 1 } { }
2217         }
2218       { \_unravel_break:w }
2219     }
2220     { \_unravel_back_input: \_unravel_scan_dimen:nN {#1} \c_false_bool }
2221     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2222     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2223     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2224     \_unravel_scan_keyword:nT { pPlLuUsS }
2225     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2226     \_unravel_scan_keyword:nT { mMiInNuUsS }
2227     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2228     \_unravel_break_point:
2229     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2230     \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2231   }

```

(End definition for _unravel_scan_glue:n.)

```

\_unravel_scan_file_name:

```

```

2232 \cs_new_protected_nopar:Npn \_unravel_scan_file_name:
2233   {

```

```

2234 \bool_gset_true:N \g__unravel_name_in_progress_bool
2235 \__unravel_get_x_non_blank:
2236 \__unravel_scan_file_name_loop:
2237 \bool_gset_false:N \g__unravel_name_in_progress_bool
2238 \__unravel_prev_input_silent:n { ~ }
2239 }
2240 \cs_new_protected_nopar:Npn \__unravel_scan_file_name_loop:
2241 {
2242 \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2243 { \__unravel_back_input: }
2244 {
2245 \tl_set:Nx \l__unravel_tmpa_tl
2246 { \__unravel_token_to_char:N \l__unravel_head_token }
2247 \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2248 {
2249 \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2250 \__unravel_get_x_next:
2251 \__unravel_scan_file_name_loop:
2252 }
2253 }
2254 }

```

(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.

```

2255 \cs_new_protected_nopar:Npn \__unravel_scan_r_token:
2256 {
2257 \bool_do_while:nn
2258 { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2259 { \__unravel_get_next: }
2260 \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2261 {
2262 \msg_error:nn { unravel } { missing-cs }
2263 \__unravel_back_input:
2264 \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2265 }
2266 \__unravel_prev_input_silent:V \l__unravel_head_tl
2267 \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2268 }

```

(End definition for __unravel_scan_r_token:.)

`__unravel_scan_toks_to_str:`

```

2269 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2270 {
2271 \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2272 \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2273 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2274 \__unravel_prev_input_silent:x

```

```

2275     { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2276   }

```

(End definition for __unravel_scan_toks_to_str:.)

__unravel_scan_toks:NN

```

2277 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2278   {
2279     \bool_if:NT #1 { \__unravel_scan_param: }
2280     \__unravel_scan_left_brace:
2281     \bool_if:NTF #2
2282       { \__unravel_scan_group_x:N #1 }
2283       { \__unravel_scan_group_n:N #1 }
2284   }

```

(End definition for __unravel_scan_toks:NN.)

__unravel_scan_param: Collect the parameter text into \l__unravel_tmpa_tl, and when seeing either a begin-group or an end-group character, put it back into the input, stop looping, and put what we collected into \l__unravel_defining_tl and into the prev_input.

__unravel_scan_param_aux:

```

2285 \cs_new_protected_nopar:Npn \__unravel_scan_param:
2286   {
2287     \tl_clear:N \l__unravel_tmpa_tl
2288     \__unravel_scan_param_aux:
2289     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2290     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2291   }
2292 \cs_new_protected_nopar:Npn \__unravel_scan_param_aux:
2293   {
2294     \__unravel_get_next:
2295     \tl_concat:NNN \l__unravel_tmpa_tl
2296       \l__unravel_tmpa_tl \l__unravel_head_tl
2297     \tl_if_empty:NTF \l__unravel_head_tl
2298       { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2299   }

```

(End definition for __unravel_scan_param:.)

__unravel_scan_group_n:N

```

2300 \cs_new_protected:Npn \__unravel_scan_group_n:N #1
2301   {
2302     \__unravel_back_input:
2303     \__unravel_input_gpop_item:NF \l__unravel_head_tl
2304     {
2305       \msg_error:nn { unravel } { runaway-text }
2306       \__unravel_exit:w
2307     }
2308     \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2309     \bool_if:NT #1
2310     { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }

```

```

2311 \__unravel_prev_input_silent:V \l__unravel_head_tl
2312 }

```

(End definition for __unravel_scan_group_n:N.)

__unravel_scan_group_x:N

```

2313 \cs_new_protected:Npn \__unravel_scan_group_x:N #1
2314 {
2315 \__unravel_input_gpop_tl:N \l__unravel_head_tl
2316 \__unravel_back_input:V \l__unravel_head_tl
2317 \bool_if:NTF #1
2318 {
2319 \__unravel_prev_input_silent:V \c_left_brace_str
2320 \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2321 \__unravel_scan_group_xdef:n { 1 }
2322 }
2323 {
2324 \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl
2325 \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2326 \__unravel_scan_group_x:n { 1 }
2327 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
2328 \__unravel_prev_input_silent:x
2329 { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2330 }
2331 }

```

(End definition for __unravel_scan_group_x:N.)

__unravel_scan_group_xdef:n

```

2332 \cs_new_protected:Npn \__unravel_scan_group_xdef:n #1
2333 {
2334 \__unravel_get_token_x:N \c_true_bool
2335 \tl_if_empty:NTF \l__unravel_head_tl
2336 {
2337 \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2338 {
2339 \__unravel_prev_input_silent:V \c_left_brace_str
2340 \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2341 \__unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2342 }
2343 {
2344 \__unravel_prev_input_silent:V \c_right_brace_str
2345 \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2346 \int_compare:nNnF {#1} = \c_one
2347 { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2348 }
2349 }
2350 {
2351 \__unravel_prev_input_silent:V \l__unravel_head_tl
2352 \tl_put_right:Nx \l__unravel_defining_tl

```

```

2353         { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2354         \__unravel_scan_group_xdef:n {#1}
2355     }
2356 }
2357 \cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }

```

(End definition for __unravel_scan_group_xdef:n.)

__unravel_scan_group_x:n

```

2358 \cs_new_protected:Npn \__unravel_scan_group_x:n #1
2359 {
2360     \__unravel_get_token_x:N \c_false_bool
2361     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2362     \tl_if_empty:NTF \l__unravel_head_tl
2363     {
2364         \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2365         { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2366         {
2367             \int_compare:nNnF {#1} = \c_one
2368             { \__unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2369         }
2370     }
2371     { \__unravel_scan_group_x:n {#1} }
2372 }
2373 \cs_generate_variant:Nn \__unravel_scan_group_x:n { f }

```

(End definition for __unravel_scan_group_x:n.)

__unravel_get_token_x:N

```

2374 \cs_new_protected:Npn \__unravel_get_token_x:N #1
2375 {
2376     \__unravel_get_next:
2377     \__unravel_token_if_protected:NF \l__unravel_head_token
2378     {
2379         \__unravel_set_cmd:
2380         \int_compare:nNnTF
2381         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { the } }
2382         {
2383             \__unravel_get_the:
2384             \bool_if:NTF #1
2385             {
2386                 \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
2387                 \__unravel_prev_input:V \l__unravel_head_tl
2388             }
2389             {
2390                 \gtl_set:Nx \l__unravel_tmpb_gtl { \l__unravel_head_tl }
2391                 \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
2392                 \__unravel_print_action:
2393             }
2394         }
2395     }

```

```

2395         { \_unravel_expand: }
2396         \_unravel_get_token_x:N #1
2397     }
2398 }

```

(End definition for _unravel_get_token_x:N.)

_unravel_scan_alt_rule:

```

2399 \cs_new_protected_nopar:Npn \_unravel_scan_alt_rule:
2400 {
2401     \_unravel_scan_keyword:nTF { wWiIdDtThH }
2402     {
2403         \_unravel_scan_normal_dimen:
2404         \_unravel_scan_alt_rule:
2405     }
2406     {
2407         \_unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2408         {
2409             \_unravel_scan_normal_dimen:
2410             \_unravel_scan_alt_rule:
2411         }
2412         {
2413             \_unravel_scan_keyword:nT { dDeEpPtThH }
2414             {
2415                 \_unravel_scan_normal_dimen:
2416                 \_unravel_scan_alt_rule:
2417             }
2418         }
2419     }
2420 }

```

(End definition for _unravel_scan_alt_rule:.)

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

```

2421 \cs_new_protected_nopar:Npn \_unravel_scan_spec:
2422 {
2423     \_unravel_scan_keyword:nTF { tToO } { \_unravel_scan_normal_dimen: }
2424     {
2425         \_unravel_scan_keyword:nT { sSpPrReEaAdD }
2426         { \_unravel_scan_normal_dimen: }
2427     }
2428     \_unravel_scan_left_brace:
2429 }

```

(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.

```

2430 \cs_new_protected:Npn \__unravel_do_box:N #1
2431 {
2432   \__unravel_get_x_non_relax:
2433   \__unravel_set_cmd:
2434   \int_compare:nNnTF
2435     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2436     { \__unravel_do_begin_box:N #1 }
2437     {
2438       \bool_if:NTF #1
2439       {
2440         \int_case:nnTF \l__unravel_head_cmd_int
2441         {
2442           { \__unravel_tex_use:n { hrule } } { }
2443           { \__unravel_tex_use:n { vrule } } { }
2444         }
2445         { \__unravel_do_leaders_rule: }
2446         { \__unravel_do_box_error: }
2447       }
2448       { \__unravel_do_box_error: }
2449     }
2450 }

```

(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.

```

2451 \cs_new_protected_nopar:Npn \__unravel_do_box_error:
2452 {
2453   \__unravel_back_input:
2454   \msg_error:nn { unravel } { missing-box }
2455   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2456   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2457 }

```

(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 $\langle dim \rangle$) 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.

```

2458 \cs_new_protected:Npn \__unravel_do_begin_box:N #1
2459 {
2460   \__unravel_prev_input:V \l__unravel_head_tl
2461   \int_case:nnTF \l__unravel_head_char_int
2462     {
2463       { 0 } { \__unravel_scan_int: } % box
2464       { 1 } { \__unravel_scan_int: } % copy
2465       { 2 } { } % lastbox
2466       { 3 } % vsplit
2467       {
2468         \__unravel_scan_int:
2469         \__unravel_scan_to:
2470         \__unravel_scan_normal_dimen:
2471       }
2472     }
2473     { \__unravel_do_simple_box:N #1 }
2474     { \__unravel_do_box_explicit:N #1 }
2475 }

```

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

```

2476 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2477 {
2478   \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2479   {
2480     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2481     \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2482     { \__unravel_prepare_mag: }
2483     \tl_use:N \l__unravel_head_tl \scan_stop:
2484     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2485     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2486   }
2487 }

```

(End definition for `__unravel_do_simple_box:N`.)

`__unravel_do_leaders_fetch_skip:`

```

2488 \cs_new_protected_nopar:Npn \__unravel_do_leaders_fetch_skip:
2489 {
2490   \__unravel_get_x_non_relax:
2491   \__unravel_set_cmd:
2492   \int_compare:nNnTF \l__unravel_head_cmd_int
2493     = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2494     {
2495       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl

```

```

2496     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2497     \__unravel_do_append_glue:
2498   }
2499   {
2500     \__unravel_back_input:
2501     \msg_error:nn { unravel } { improper-leaders }
2502     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2503     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2504   }
2505 }

```

(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 letters v, h for vertical/horizontal leaders, and Z for normal boxes.

```

2506 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2507   {
2508     \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
2509     { \__unravel_box_hook:N \tex_everyhbox:D }
2510     { \__unravel_box_hook:N \tex_everyvbox:D }
2511     % ^^A todo: TeX calls |normal_paragraph| here.
2512     \__unravel_scan_spec:
2513     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2514     \__unravel_set_action_text:x
2515     { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2516     \seq_push:Nx \l__unravel_leaders_box_seq
2517     { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2518     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2519     \gtl_gconcat:NNN \g__unravel_output_gtl
2520     \g__unravel_output_gtl \c_group_begin_gtl
2521     \tl_use:N \l__unravel_head_tl
2522     \c_group_begin_token \__unravel_box_hook_end:
2523   }

```

(End definition for __unravel_do_box_explicit:N)

__unravel_box_hook:N
 __unravel_box_hook:w
 __unravel_box_hook_end:

Used to capture the contents of an \everyhbox or similar, without altering \everyhbox too much (just add one token at the start). The various o-expansions remove \prg_do_nothing:, used to avoid losing braces.

```

2524 \cs_new_protected:Npn \__unravel_box_hook:N #1
2525   {

```

```

2526 \tl_set:NV \l__unravel_tmpa_tl #1
2527 \str_if_eq_x:nnF
2528 { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2529 {
2530   \exp_args:Nx #1
2531   {
2532     \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2533     \exp_not:V #1
2534   }
2535 }
2536 \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2537 {
2538   \exp_args:No #1 {##1}
2539   \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2540   \gtl_clear:N \l__unravel_after_group_gtl
2541   \__unravel_print_action:
2542   \__unravel_back_input:o {##1}
2543   \__unravel_set_action_text:x
2544   { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2545   \tl_if_empty:oF {##1} { \__unravel_print_action: }
2546 }
2547 }
2548 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2549 \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`, `height` and `width` keywords, we are in the same situation as after finding a box. Fetch the required skip accordingly.

```

2550 \cs_new_protected_nopar:Npn \__unravel_do_leaders_rule:
2551 {
2552   \__unravel_prev_input:V \l__unravel_head_tl
2553   \__unravel_scan_alt_rule:
2554   \__unravel_do_leaders_fetch_skip:
2555 }

```

(End definition for __unravel_do_leaders_rule:.)

2.9 Paragraphs

`__unravel_charcode_if_safe:nTF`

```

2556 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2557 {
2558   \bool_if:nTF
2559   {
2560     \int_compare_p:n { #1 = '!' }
2561     || \int_compare_p:n { ' ' <= #1 <= '[' }
2562     || \int_compare_p:n { #1 = ']' }
2563     || \int_compare_p:n { ' ' <= #1 <= 'z' }

```

```

2564     }
2565     { \prg_return_true: }
2566     { \prg_return_false: }
2567 }

```

(End definition for `__unravel_charcode_if_safe:nTF`.)

```

\__unravel_char:n
\__unravel_char:V
\__unravel_char:x
2568 \cs_new_protected:Npn \__unravel_char:n #1
2569 {
2570   \tex_char:D #1 \scan_stop:
2571   \__unravel_charcode_if_safe:nTF {#1}
2572   { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2573   {
2574     \tl_set:Nx \l__unravel_tmpa_tl
2575     { \exp_not:N \char \int_eval:n {#1} ~ }
2576   }
2577   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2578   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2579 }
2580 \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
2581 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2582 {
2583   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2584   { % math active
2585     \gtl_set:Nx \l__unravel_head_gtl
2586     { \char_generate:nn {#1} { 12 } }
2587     \__unravel_back_input:
2588   }
2589   { \__unravel_char:n {#1} }
2590 }
2591 \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
2592 \cs_new_protected:Npn \__unravel_mathchar:n #1
2593 {
2594   \tex_mathchar:D #1 \scan_stop:
2595   \tl_set:Nx \l__unravel_tmpa_tl
2596   { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2597   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2598   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2599 }
2600 \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 T_EX itself. Our only task is to correctly position the `\everypar` tokens in the input that we will read, rather than letting T_EX run the code right away.

```
2601 \cs_new_protected:Npn \_unravel_new_graf:N #1
2602   {
2603     \tl_set:NV \l__unravel_tmpa_tl \_unravel_everypar:w
2604     \_unravel_everypar:w { }
2605     \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2606     \exp_args:NV \_unravel_everypar:w \l__unravel_tmpa_tl
2607     \_unravel_back_input:V \l__unravel_tmpa_tl
2608     \_unravel_print_action:x
2609     {
2610       \g__unravel_action_text_str \c_space_tl : ~
2611       \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2612     }
2613   }
```

(End definition for _unravel_new_graf:N.)

`_unravel_end_graf:`

```
2614 \cs_new_protected_nopar:Npn \_unravel_end_graf:
2615   { \mode_if_horizontal:T { \_unravel_normal_paragraph: } }
```

(End definition for _unravel_end_graf:.)

`_unravel_normal_paragraph:`

```
2616 \cs_new_protected_nopar:Npn \_unravel_normal_paragraph:
2617   {
2618     \tex_par:D
2619     \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2620     \_unravel_print_action:x { Paragraph~end. }
2621   }
```

(End definition for _unravel_normal_paragraph:.)

`_unravel_build_page:`

```
2622 \cs_new_protected_nopar:Npn \_unravel_build_page:
2623   {
2624   }
```

(End definition for _unravel_build_page:.)

2.10 Groups

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

```

2625 \cs_new_protected_nopar:Npn \__unravel_handle_right_brace:
2626   {
2627     \int_compare:nTF { 1 <= \__unravel_currentgroupstype: <= 13 }
2628     {
2629       \gtl_gconcat:NNN \g__unravel_output_gtl
2630       \g__unravel_output_gtl \c_group_end_gtl
2631       \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2632       \int_case:nn \__unravel_currentgroupstype:
2633         {
2634           { 1 } { \__unravel_end_simple_group: } % simple
2635           { 2 } { \__unravel_end_box_group: } % hbox
2636           { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2637           { 4 } { \__unravel_end_graf: \__unravel_end_box_group: } % vbox
2638           { 5 } { \__unravel_end_graf: \__unravel_end_box_group: } % vtop
2639           { 6 } { \__unravel_end_align_group: } % align
2640           { 7 } { \__unravel_end_no_align_group: } % no_align
2641           { 8 } { \__unravel_end_output_group: } % output
2642           { 9 } { \__unravel_end_simple_group: } % math
2643           { 10 } { \__unravel_end_disc_group: } % disc
2644           { 11 } { \__unravel_end_graf: \__unravel_end_simple_group: } % insert
2645           { 12 } { \__unravel_end_graf: \__unravel_end_simple_group: } % vcenter
2646           { 13 } { \__unravel_end_math_choice_group: } % math_choice
2647         }
2648       }
2649       { % bottom_level, semi_simple, math_shift, math_left
2650         \l__unravel_head_token
2651         \__unravel_print_action:
2652       }
2653     }

```

(End definition for __unravel_handle_right_brace:.)

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

```

2654 \cs_new_protected_nopar:Npn \__unravel_end_simple_group:
2655   {
2656     \l__unravel_head_token
2657     \__unravel_print_action:
2658   }

```

(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.

```

2659 \cs_new_protected_nopar:Npn \__unravel_end_box_group:
2660   {
2661     \seq_pop:NN \l__unravel_leaders_box_seq \l__unravel_tmpa_tl

```

```

2662   \exp_args:No \__unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
2663 }
2664 \cs_new_protected:Npn \__unravel_end_box_group_aux:n #1
2665 {
2666   \str_if_eq_x:nnTF {#1} { Z }
2667   { \__unravel_end_simple_group: }
2668   {
2669     \__unravel_get_x_non_relax:
2670     \__unravel_set_cmd:
2671     \int_compare:nNnTF \l__unravel_head_cmd_int
2672     = { \__unravel_tex_use:n { #1 skip } }
2673     {
2674       \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2675       \__unravel_do_append_glue:
2676     }
2677     {
2678       \__unravel_back_input:
2679       \c_group_end_token \group_begin: \group_end:
2680       \__unravel_print_action:
2681     }
2682   }
2683 }

```

(End definition for __unravel_end_box_group:.)

__unravel_off_save:

```

2684 \cs_new_protected_nopar:Npn \__unravel_off_save:
2685 {
2686   \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
2687   { % bottom-level
2688     \msg_error:nnx { unravel } { extra-close }
2689     { \token_to_meaning:N \l__unravel_head_token }
2690   }
2691   {
2692     \__unravel_back_input:
2693     \int_case:nnF \__unravel_currentgrouptype:
2694     {
2695       { 14 } % semi_simple_group
2696       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
2697       { 15 } % math_shift_group
2698       { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2699       { 16 } % math_left_group
2700       { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2701     }
2702     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2703     \__unravel_back_input:
2704     \msg_error:nnx { unravel } { off-save }
2705     { \gtl_to_str:N \l__unravel_head_gtl }
2706   }
2707 }

```

(End definition for `_unravel_off_save:`.)

2.11 Modes

```
\_unravel_mode_math:n
\_unravel_mode_non_math:n
\_unravel_mode_vertical:n
2708 \cs_new_protected:Npn \_unravel_mode_math:n #1
2709   { \mode_if_math:TF {#1} { \_unravel_insert_dollar_error: } }
2710 \cs_new_protected:Npn \_unravel_mode_non_math:n #1
2711   { \mode_if_math:TF { \_unravel_insert_dollar_error: } {#1} }
2712 \cs_new_protected:Npn \_unravel_mode_vertical:n #1
2713   {
2714     \mode_if_math:TF
2715       { \_unravel_insert_dollar_error: }
2716       { \mode_if_horizontal:TF { \_unravel_head_for_vmode: } {#1} }
2717   }
2718 \cs_new_protected:Npn \_unravel_mode_non_vertical:n #1
2719   {
2720     \mode_if_vertical:TF
2721       { \_unravel_back_input: \_unravel_new_graf:N \c_true_bool }
2722       {#1}
2723   }
```

(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.
2724 \cs_new_protected_nopar:Npn \_unravel_head_for_vmode:
2725   {
2726     \mode_if_inner:TF
2727     {
2728       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
2729       {
2730         \msg_error:nn { unravel } { hrulerule-bad-mode }
2731         \_unravel_print_action:
2732       }
2733       { \_unravel_off_save: }
2734     }
2735     {
2736       \_unravel_back_input:
2737       \gtl_set:Nn \l__unravel_head_gtl { \par }
2738       \_unravel_back_input:
2739     }
2740   }
```

(End definition for `_unravel_head_for_vmode:`.)

```
\_unravel_goto_inner_math:
2741 \cs_new_protected_nopar:Npn \_unravel_goto_inner_math:
2742   {
2743     \_unravel_box_hook:N \tex_everymath:D
```



```

2744   $ % $
2745   \__unravel_box_hook_end:
2746   }

```

(End definition for __unravel_goto_inner_math:.)

__unravel_goto_display_math:

```

2747 \cs_new_protected_nopar:Npn \__unravel_goto_display_math:
2748 {
2749   \__unravel_box_hook:N \tex_everydisplay:D
2750   $ $
2751   \__unravel_box_hook_end:
2752 }

```

(End definition for __unravel_goto_display_math:.)

__unravel_after_math:

```

2753 \cs_new_protected_nopar:Npn \__unravel_after_math:
2754 {
2755   \mode_if_inner:TF
2756   {
2757     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2758     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2759     $ % $
2760   }
2761   {
2762     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2763     \__unravel_get_x_next:
2764     \token_if_eq_catcode:NMF
2765     \l__unravel_head_token \c_math_toggle_token
2766     {
2767       \__unravel_back_input:
2768       \tl_set:Nn \l__unravel_head_tl { $ } % $
2769       \msg_error:nn { unravel } { missing-dollar }
2770     }
2771     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2772     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2773     $ $
2774   }
2775   \__unravel_print_action:
2776 }

```

(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.

```

2777 \cs_new_protected_nopar:Npn \__unravel_do_step:
2778 {

```

```

2779  \__unravel_set_action_text:
2780  \bool_if:NT \g__unravel_internal_debug_bool
2781    { \iow_term:x { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int } } }
2782  \cs_if_exist_use:cF
2783    { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2784    { \msg_error:nnx { unravel } { internal } { unknown-command } }
2785  }

```

(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 `\crcr`.
- `macro_char=6` for macro parameter characters (catcode 6).
- `superscript_char=7` for superscript characters (catcode 7).
- `subscript_char=8` for subscript characters (catcode 8).
- `endv=9` for ?.
- `blank_char=10` for blank spaces (catcode 10).
- `the_char=11` for letters (catcode 11).
- `other_char=12` for other characters (catcode 12).
- `par_end=13` for `\par`.
- `stop=14` for `\end` and `\dump`.
- `delim_num=15` for `\delimiter`.

Not implemented at all: `endv`.

`\relax` does nothing.

```
2786 \__unravel_new_tex_cmd:nn { relax } % 0
2787 { \__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.

```
2788 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
2789 {
2790   \gtl_gconcat:NNN \g__unravel_output_gtl
2791   \g__unravel_output_gtl \c_group_begin_gtl
2792   \__unravel_print_action:
2793   \l__unravel_head_token
2794   \gtl_clear:N \l__unravel_after_group_gtl
2795 }
```

```
2796 \__unravel_new_tex_cmd:nn { end-group_char } % 2
2797 { \__unravel_handle_right_brace: }
```

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

```
2798 \__unravel_new_tex_cmd:nn { math_char } % 3
2799 {
2800   \__unravel_mode_non_vertical:n
2801   {
2802     \mode_if_math:TF
2803     {
2804       \int_compare:nNnTF
2805         \__unravel_currentgrouptype: = { 15 } % math_shift_group
2806         { \__unravel_after_math: }
2807         { \__unravel_off_save: }
2808     }
2809     {
2810       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2811       \__unravel_get_next:
2812       \token_if_eq_catcode:NNTF
2813         \l__unravel_head_token \c_math_toggle_token
2814         {
2815           \mode_if_inner:TF
2816             { \__unravel_back_input: \__unravel_goto_inner_math: }
2817             {
2818               \gtl_gput_right:NV
2819                 \g__unravel_output_gtl \l__unravel_head_tl
2820                 \__unravel_goto_display_math:
2821             }
2822           }
2823         { \__unravel_back_input: \__unravel_goto_inner_math: }
2824       }
2825     }
2826 }
```

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.

```

2827 \__unravel_new_tex_cmd:nn { alignment_char } % 4
2828 { \l__unravel_head_token \__unravel_print_action: }
2829 \__unravel_new_tex_cmd:nn { car_ret } % 5
2830 { \l__unravel_head_token \__unravel_print_action: }
2831 \__unravel_new_tex_cmd:nn { macro_char } % 6
2832 { \l__unravel_head_token \__unravel_print_action: }

2833 \__unravel_new_tex_cmd:nn { superscript_char } % 7
2834 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2835 \__unravel_new_tex_cmd:nn { subscript_char } % 8
2836 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2837 \cs_new_protected_nopar:Npn \__unravel_sub_sup:
2838 {
2839 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2840 \__unravel_print_action:
2841 \__unravel_get_x_non_relax:
2842 \__unravel_set_cmd:
2843 \int_case:nnTF \l__unravel_head_cmd_int
2844 {
2845 { \__unravel_tex_use:n { the_char } }
2846 { \__unravel_prev_input:V \l__unravel_head_tl }
2847 { \__unravel_tex_use:n { other_char } }
2848 { \__unravel_prev_input:V \l__unravel_head_tl }
2849 { \__unravel_tex_use:n { char_given } }
2850 { \__unravel_prev_input:V \l__unravel_head_tl }
2851 { \__unravel_tex_use:n { char_num } }
2852 {
2853 \__unravel_prev_input:V \l__unravel_head_tl
2854 \__unravel_scan_int:
2855 }
2856 { \__unravel_tex_use:n { math_char_num } }
2857 {
2858 \__unravel_prev_input:V \l__unravel_head_tl
2859 \__unravel_scan_int:
2860 }
2861 { \__unravel_tex_use:n { math_given } }
2862 { \__unravel_prev_input:V \l__unravel_head_tl }
2863 { \__unravel_tex_use:n { delim_num } }
2864 { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
2865 }
2866 {
2867 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2868 \gtl_gput_right:NV \g__unravel_output_gt1 \l__unravel_head_tl
2869 \tl_use:N \l__unravel_head_tl \scan_stop:
2870 }
2871 {
2872 \__unravel_back_input:

```

```

2873     \_unravel_scan_left_brace:
2874     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2875     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2876     \gtl_gconcat:NNN \g__unravel_output_gtl
2877     \g__unravel_output_gtl \c_group_begin_gtl
2878     \tl_use:N \l__unravel_head_tl \c_group_begin_token
2879   }
2880   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2881 }

```

```

2882 \_unravel_new_tex_cmd:nn { endv } % 9
2883 { \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.

```

2884 \_unravel_new_tex_cmd:nn { blank_char } % 10
2885 {
2886   \mode_if_horizontal:T
2887   {
2888     \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
2889     \l__unravel_head_token
2890   }
2891   \_unravel_print_action:
2892 }

```

Letters and other characters leave vertical mode.

```

2893 \_unravel_new_tex_cmd:nn { the_char } % 11
2894 {
2895   \_unravel_mode_non_vertical:n
2896   {
2897     \tl_set:Nx \l__unravel_tmpa_tl
2898     { ' \_unravel_token_to_char:N \l__unravel_head_token }
2899     \mode_if_math:TF
2900     { \_unravel_char_in_mmode:V \l__unravel_tmpa_tl }
2901     { \_unravel_char:V \l__unravel_tmpa_tl }
2902   }
2903 }

```

```

2904 \_unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12

```

```

2905 \_unravel_new_tex_cmd:nn { par_end } % 13
2906 {
2907   \_unravel_mode_non_math:n
2908   {
2909     \mode_if_vertical:TF
2910     { \_unravel_normal_paragraph: }
2911     {
2912       % if align_state<0 then off_save;
2913       \_unravel_end_graf:
2914       \mode_if_vertical:T
2915       { \mode_if_inner:F { \_unravel_build_page: } }
2916     }

```

```

2917     }
2918 }
2919 \__unravel_new_tex_cmd:nn { stop } % 14
2920 {
2921   \__unravel_mode_vertical:n
2922   {
2923     \mode_if_inner:TF
2924     { \__unravel_forbidden_case: }
2925     {
2926       % ^^A todo: unless its_all_over
2927       \int_gdecr:N \g__unravel_ends_int
2928       \int_compare:nNnTF \g__unravel_ends_int > \c_zero
2929       {
2930         \__unravel_back_input:
2931         \__unravel_back_input:n
2932         {
2933           \__unravel_hbox:w to \tex_hsize:D { }
2934           \tex_vfill:D
2935           \tex_penalty:D - '10000000000 ~
2936         }
2937         \__unravel_build_page:
2938         \__unravel_print_action:x { End-everything! }
2939       }
2940       {
2941         \__unravel_print_outcome:
2942         \l__unravel_head_token
2943       }
2944     }
2945   }
2946 }
2947 \__unravel_new_tex_cmd:nn { delim_num } % 15
2948 {
2949   \__unravel_mode_math:n
2950   {
2951     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2952     \__unravel_print_action:
2953     \__unravel_scan_int:
2954     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2955     \tl_use:N \l__unravel_head_tl \scan_stop:
2956     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2957   }
2958 }

```

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.

```

2959 \__unravel_new_tex_cmd:nn { char_num } % 16
2960 {
2961   \__unravel_mode_non_vertical:n
2962   {
2963     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2964     \__unravel_print_action:
2965     \__unravel_scan_int:
2966     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2967     \mode_if_math:TF
2968     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
2969     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
2970   }
2971 }

```

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.

```

2972 \__unravel_new_tex_cmd:nn { math_char_num } % 17
2973 {
2974   \__unravel_mode_math:n
2975   {
2976     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl

```

```

2977     \__unravel_print_action:
2978     \__unravel_scan_int:
2979     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2980     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
2981   }
2982 }
2983 \__unravel_new_tex_cmd:nn { mark } % 18
2984 {
2985   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2986   \__unravel_print_action:
2987   \int_compare:nNnF \l__unravel_head_char_int = \c_zero
2988     { \__unravel_scan_int: }
2989   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2990   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2991   \seq_gpop_right:Nn \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2992   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2993   \__unravel_print_action:x
2994     { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
2995   \tl_put_right:Nx \l__unravel_head_tl
2996     { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2997   \tl_use:N \l__unravel_head_tl
2998 }

```

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.

```

2999 \__unravel_new_tex_cmd:nn { xray } % 19
3000 {
3001   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3002   \__unravel_print_action:
3003   \int_case:nnF \l__unravel_head_char_int
3004     {
3005       { 0 }
3006       { % show
3007         \__unravel_get_next:
3008         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3009         \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl
3010       }
3011       { 2 }
3012       { % showthe
3013         \__unravel_get_x_next:
3014         \__unravel_scan_something_internal:n { 5 }
3015         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3016         \exp_args:Nx \etex_showtokens:D
3017           { \tl_tail:N \l__unravel_head_tl }
3018       }

```



```

3019     }
3020     { % no operand for showlists, showgroups, showifs
3021       \int_compare:nNnT \l__unravel_head_char_int = \c_one % showbox
3022       { \__unravel_scan_int: }
3023       \int_compare:nNnT \l__unravel_head_char_int = \c_five % showtokens
3024       { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3025       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3026       \tl_use:N \l__unravel_head_tl \scan_stop:
3027     }
3028   }
    make_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
3029 \__unravel_new_tex_cmd:nn { make_box } % 20
3030 {
3031   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3032   \__unravel_back_input:
3033   \__unravel_do_box:N \c_false_bool
3034 }

```

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

```

3035 \cs_new_protected_nopar:Npn \__unravel_do_move:
3036 {
3037   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3038   \__unravel_print_action:
3039   \__unravel_scan_normal_dimen:
3040   \__unravel_do_box:N \c_false_bool
3041 }

```

(End definition for __unravel_do_move:.)

hmove=21 for \moveright and \moveleft.

```

3042 \__unravel_new_tex_cmd:nn { hmove } % 21
3043 {
3044   \mode_if_vertical:TF
3045   { \__unravel_do_move: } { \__unravel_forbidden_case: }
3046 }

```

vmove=22 for \lower and \raise.

```

3047 \__unravel_new_tex_cmd:nn { vmove } % 22
3048 {
3049   \mode_if_vertical:TF
3050   { \__unravel_forbidden_case: } { \__unravel_do_move: }
3051 }

```

`__unravel_do_unpackage:`

```

3052 \cs_new_protected_nopar:Npn \__unravel_do_unpackage:
3053 {
3054   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3055   \__unravel_print_action:
3056   \__unravel_scan_int:
3057   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl

```

```

3058 \tl_use:N \l__unravel_head_tl \scan_stop:
3059 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3060 }

(End definition for \__unravel_do_unpackage:.)
un_hbox=23 for \unhbox and \unhcopy.
3061 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3062 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

unvbox=24 for \unvbox, \unvcopy, \pagediscards, and \splitdiscards. The latter two take no operands, so we just let TeX do its thing, then we show the action.
3063 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3064 {
3065 \__unravel_mode_vertical:n
3066 {
3067 \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3068 { \l__unravel_head_token \__unravel_print_action: }
3069 { \__unravel_do_unpackage: }
3070 }
3071 }

remove_item=25 for \unpenalty (12), \unkern (11), \unskip (10). Those commands only act on TeX's box/glue data structures, which unravel does not (and cannot) care about.
3072 \__unravel_new_tex_cmd:nn { remove_item } % 25
3073 { \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.
3074 \cs_new_protected_nopar:Npn \__unravel_do_append_glue:
3075 {
3076 \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3077 { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3078 {
3079 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3080 \__unravel_print_action:
3081 \exp_args:Nf \__unravel_scan_glue:n
3082 { \int_eval:n { \l__unravel_head_char_int - 2 } }
3083 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3084 \tl_use:N \l__unravel_head_tl \scan_stop:
3085 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3086 }
3087 }

(End definition for \__unravel_do_append_glue:.)
hskip=26 for \hfil, \hfill, \hss, \hfilneg, \hskip.
3088 \__unravel_new_tex_cmd:nn { hskip } % 26
3089 { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }

```

```

vskip=27 for \vfil, \vfill, \vss, \vfilneg, \vskip.
3090 \__unravel_new_tex_cmd:nn { vskip } % 27
3091 { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }
vskip=28 for \mskip (5).
3092 \__unravel_new_tex_cmd:nn { mskip } % 28
3093 { \__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.

```

3094 \cs_new_protected_nopar:Npn \__unravel_do_append_kern:
3095 {
3096   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3097   \__unravel_print_action:
3098   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
3099     { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
3100     { \__unravel_scan_dimen:nN { 3 } \c_false_bool }
3101   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3102   \tl_use:N \l__unravel_head_tl \scan_stop:
3103   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3104 }

(End definition for \__unravel_do_append_kern:.)
kern=29 for \kern (1).
3105 \__unravel_new_tex_cmd:nn { kern } % 29
3106 { \__unravel_do_append_kern: }
mkern=30 for \mkern (99).
3107 \__unravel_new_tex_cmd:nn { mkern } % 30
3108 { \__unravel_mode_math:n { \__unravel_do_append_kern: } }
leader_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).
3109 \__unravel_new_tex_cmd:nn { leader_ship } % 31
3110 {
3111   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3112   \__unravel_print_action:
3113   \__unravel_do_box:N \c_true_bool
3114 }

```

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

```

3115 \__unravel_new_tex_cmd:nn { halign } % 32
3116 { \msg_fatal:nxx { unravel } { not-implemented } { halign } }
3117 \__unravel_new_tex_cmd:nn { valign } % 33
3118 { \msg_fatal:nxx { unravel } { not-implemented } { valign } }
3119 \__unravel_new_tex_cmd:nn { no_align } % 34
3120 { \msg_fatal:nxx { unravel } { not-implemented } { noalign } }
3121 \__unravel_new_tex_cmd:nn { vrule } % 35
3122 { \__unravel_mode_non_vertical:n { \__unravel_do_rule: } }
3123 \__unravel_new_tex_cmd:nn { hrule } % 36
3124 { \__unravel_mode_vertical:n { \__unravel_do_rule: } }
3125 \cs_new_protected_nopar:Npn \__unravel_do_rule:
3126 {
3127 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3128 \__unravel_print_action:
3129 \__unravel_scan_alt_rule:
3130 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3131 \tl_use:N \l__unravel_head_tl \scan_stop:
3132 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3133 }
3134 \__unravel_new_tex_cmd:nn { insert } % 37
3135 { \__unravel_begin_insert_or_adjust: }
3136 \__unravel_new_tex_cmd:nn { vadjust } % 38
3137 {
3138 \mode_if_vertical:TF
3139 { \__unravel_forbidden_case: } { \__unravel_begin_insert_or_adjust: }
3140 }

```

```

3141 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
3142 {
3143   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3144   {
3145     \__unravel_get_x_non_blank:
3146     \__unravel_set_cmd:
3147     \__unravel_do_step:
3148   }
3149   { \msg_error:nn { unravel } { not-implemented } { pdfprimitive } }
3150 }

3151 \__unravel_new_tex_cmd:nn { after_assignment } % 40
3152 {
3153   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3154   \__unravel_get_next:
3155   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3156   \__unravel_print_action:x
3157   {
3158     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3159     \gtl_to_str:N \l__unravel_head_gtl
3160   }
3161 }

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

3162 \__unravel_new_tex_cmd:nn { after_group } % 41
3163 {
3164   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3165   \__unravel_get_next:
3166   \int_compare:nNnTF \__unravel_currentgrouptype: = \c_zero
3167   {
3168     \__unravel_print_action:x
3169     {
3170       Aftergroup~(level~0~=>~dropped):~
3171       \tl_to_str:N \l__unravel_tmpa_tl
3172       \gtl_to_str:N \l__unravel_head_gtl
3173     }
3174   }
3175   {
3176     \gtl_concat:NNN \l__unravel_after_group_gtl
3177     \l__unravel_after_group_gtl \l__unravel_head_gtl
3178     \__unravel_print_action:x
3179     {
3180       Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3181       \gtl_to_str:N \l__unravel_head_gtl
3182     }
3183   }
3184 }

See \__unravel_do_append_glue:.

3185 \__unravel_new_tex_cmd:nn { break_penalty } % 42

```

```

3186 {
3187   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3188   \__unravel_print_action:
3189   \__unravel_scan_int:
3190   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3191   \tl_use:N \l__unravel_head_tl \scan_stop:
3192   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3193 }

3194 \__unravel_new_tex_cmd:nn { start_par } % 43
3195 {
3196   \mode_if_vertical:TF
3197   {
3198     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3199     { \__unravel_new_graf:N \c_false_bool }
3200     { \__unravel_new_graf:N \c_true_bool }
3201   }
3202   {
3203     \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3204     {
3205       \__unravel_hbox:w width \tex_parindent:D { }
3206       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3207     }
3208     \__unravel_print_action:
3209   }
3210 }

3211 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3212 {
3213   \mode_if_vertical:TF { \__unravel_forbidden_case: }
3214   { \l__unravel_head_token \__unravel_print_action: }
3215 }

\__unravel_do_accent:

3216 \cs_new_protected_nopar:Npn \__unravel_do_accent:
3217 {
3218   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3219   \__unravel_print_action:
3220   \__unravel_scan_int:
3221   \__unravel_do_assignments:
3222   \bool_if:nTF
3223   {
3224     \token_if_eq_catcode_p:NN
3225     \l__unravel_head_token \c_catcode_letter_token
3226     ||
3227     \token_if_eq_catcode_p:NN
3228     \l__unravel_head_token \c_catcode_other_token
3229     ||
3230     \int_compare_p:nNn
3231     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3232   }

```

```

3233     { \_unravel_prev_input:V \l\_unravel_head_tl }
3234     {
3235         \token_if_eq_meaning:NNTF \l\_unravel_head_token \tex_char:D
3236         {
3237             \_unravel_prev_input:V \l\_unravel_head_tl
3238             \_unravel_scan_int:
3239         }
3240         { \_unravel_break:w }
3241     }
3242     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
3243     \gtl_gput_right:NV \g\_unravel_output_gtl \l\_unravel_head_tl
3244     \tl_use:N \l\_unravel_head_tl \scan_stop:
3245     \_unravel_print_action:x { \tl_to_str:N \l\_unravel_head_tl }
3246     \_unravel_break_point:
3247 }

```

(End definition for _unravel_do_accent:.)

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

```

3248 \cs_new_protected_nopar:Npn \_unravel_do_math_accent:
3249 {
3250     \seq_gput_right:NV \g\_unravel_prev_input_seq \l\_unravel_head_tl
3251     \_unravel_print_action:
3252     \_unravel_scan_int:
3253     \_unravel_scan_math:
3254     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
3255     \gtl_gput_right:NV \g\_unravel_output_gtl \l\_unravel_head_tl
3256     \tl_use:N \l\_unravel_head_tl \scan_stop:
3257     \_unravel_print_action:x { \tl_to_str:N \l\_unravel_head_tl }
3258 }

```

(End definition for _unravel_do_math_accent:.)

```

3259 \_unravel_new_tex_cmd:nn { accent } % 45
3260 {
3261     \_unravel_mode_non_vertical:n
3262     {
3263         \mode_if_math:TF
3264         { \_unravel_do_math_accent: } { \_unravel_do_accent: }
3265     }
3266 }
3267 \_unravel_new_tex_cmd:nn { math_accent } % 46
3268 { \_unravel_mode_math:n { \_unravel_do_math_accent: } }
3269 \_unravel_new_tex_cmd:nn { discretionary } % 47
3270 { \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

```
3271 \__unravel_new_tex_cmd:nn { eq_no } % 48
3272 { \msg_error:nxx { unravel } { not-implemented } { eqno } }
3273 \__unravel_new_tex_cmd:nn { left_right } % 49
3274 { \msg_error:nxx { unravel } { not-implemented } { left/right } }
3275 \__unravel_new_tex_cmd:nn { math_comp } % 50
3276 { \msg_error:nxx { unravel } { not-implemented } { math-comp } }
3277 \__unravel_new_tex_cmd:nn { limit_switch } % 51
3278 { \msg_error:nxx { unravel } { not-implemented } { limits } }
3279 \__unravel_new_tex_cmd:nn { above } % 52
3280 { \msg_error:nxx { unravel } { not-implemented } { above } }
3281 \__unravel_new_tex_cmd:nn { math_style } % 53
3282 { \msg_error:nxx { unravel } { not-implemented } { math-style } }
3283 \__unravel_new_tex_cmd:nn { math_choice } % 54
3284 { \msg_error:nxx { unravel } { not-implemented } { math-choice } }
3285 \__unravel_new_tex_cmd:nn { non_script } % 55
3286 { \msg_error:nxx { unravel } { not-implemented } { non-script } }
3287 \__unravel_new_tex_cmd:nn { vcenter } % 56
3288 { \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

```

3289 \__unravel_new_tex_cmd:nn { case_shift } % 57
3290 {
3291   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3292   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3293   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3294   \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
3295 }
3296 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
3297 {
3298   #1 { \__unravel_back_input:n {#2} }
3299   \__unravel_print_action:x
3300   { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3301 }
3302 \__unravel_new_tex_cmd:nn { message } % 58
3303 {
3304   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3305   \__unravel_print_action:
3306   \__unravel_scan_toks_to_str:
3307   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3308   \tl_use:N \l__unravel_head_tl
3309   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3310 }

```

Extensions are implemented in a later section.

```

3311 \__unravel_new_tex_cmd:nn { extension } % 59
3312 {
3313   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3314   \__unravel_print_action:
3315   \__unravel_scan_extension_operands:
3316   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3317   \tl_use:N \l__unravel_head_tl \scan_stop:
3318   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3319 }

```

```

3320 \__unravel_new_tex_cmd:nn { in_stream } % 60
3321 {
3322   \seq_put_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3323   \__unravel_print_action:
3324   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3325   {
3326     \__unravel_scan_int:
3327     \__unravel_scan_optional_equals:
3328     \__unravel_scan_file_name:
3329   }
3330   { \__unravel_scan_int: }
3331   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3332   \tl_use:N \l__unravel_head_tl \scan_stop:
3333   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3334 }

3335 \__unravel_new_tex_cmd:nn { begin_group } % 61
3336 {
3337   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3338   \l__unravel_head_token
3339   \gtl_clear:N \l__unravel_after_group_gtl
3340   \__unravel_print_action:
3341 }

3342 \__unravel_new_tex_cmd:nn { end_group } % 62
3343 {
3344   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3345   \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3346   \l__unravel_head_token
3347   \__unravel_print_action:
3348 }

3349 \__unravel_new_tex_cmd:nn { omit } % 63
3350 { \msg_error:nn { unravel } { not-implemented } { omit } }

3351 \__unravel_new_tex_cmd:nn { ex_space } % 64
3352 {
3353   \__unravel_mode_non_vertical:n
3354   { \l__unravel_head_token \__unravel_print_action: }
3355 }

3356 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3357 {
3358   \__unravel_mode_non_vertical:n
3359   { \l__unravel_head_token \__unravel_print_action: }
3360 }

3361 \__unravel_new_tex_cmd:nn { radical } % 66
3362 {
3363   \__unravel_mode_math:n
3364   {
3365     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3366     \__unravel_print_action:
3367     \__unravel_scan_int:

```

```

3368     \_unravel_scan_math:
3369     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3370     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3371     \tl_use:N \l__unravel_head_tl \scan_stop:
3372     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3373   }
3374 }

Let TEX cause the error.
3375 \_unravel_new_tex_cmd:nn { end_cs_name } % 67
3376 { \l__unravel_head_token \_unravel_print_action: }

See the_char and other_char.
3377 \_unravel_new_tex_cmd:nn { char_given } % 68
3378 {
3379   \_unravel_mode_non_vertical:n
3380   {
3381     \mode_if_math:TF
3382     { \_unravel_char_in_mmode:V \l__unravel_head_char_int }
3383     { \_unravel_char:V \l__unravel_head_char_int }
3384   }
3385 }

See math_char_num.
3386 \_unravel_new_tex_cmd:nn { math_given } % 69
3387 {
3388   \_unravel_mode_math:n
3389   { \_unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3390 }

3391 \_unravel_new_tex_cmd:nn { last_item } % 70
3392 { \_unravel_forbidden_case: }

```

2.13.6 Extensions

_unravel_scan_extension_operands:

```

3393 \cs_new_protected_nopar:Npn \_unravel_scan_extension_operands:
3394 {
3395   \int_case:nnF \l__unravel_head_char_int
3396   {
3397     { 0 } % openout
3398     {
3399       \_unravel_scan_int:
3400       \_unravel_scan_optional_equals:
3401       \_unravel_scan_file_name:
3402     }
3403     { 1 } % write
3404     {
3405       \_unravel_scan_int:
3406       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3407     }

```

```

3408 { 2 } % closeout
3409 { \_unravel_scan_int: }
3410 { 3 } % special
3411 { \_unravel_scan_toks_to_str: }
3412 { 4 } % immediate
3413 { \_unravel_scan_immediate_operands: }
3414 { 5 } % setlanguage
3415 {
3416     \mode_if_horizontal:TF
3417     { \_unravel_scan_int: }
3418     { \msg_error:nn { unravel } { invalid-mode } }
3419 }
3420 { 6 } % pdfliteral
3421 {
3422     \_unravel_scan_keyword:nF { dDiIrReEcCtT }
3423     { \_unravel_scan_keyword:n { pPaAgGeE } }
3424     \_unravel_scan_pdf_ext_toks:
3425 }
3426 { 7 } % pdfobj
3427 {
3428     \_unravel_scan_keyword:nTF
3429     { rReEsSeErRvVeEoObBjJnNuUmM }
3430     { \_unravel_skip_optional_space: }
3431     {
3432         \_unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3433         { \_unravel_scan_int: }
3434         \_unravel_scan_keyword:nT { sStTrReEaAmM }
3435         {
3436             \_unravel_scan_keyword:nT { aAtTtTrR }
3437             { \_unravel_scan_pdf_ext_toks: }
3438         }
3439         \_unravel_scan_keyword:n { fFiIlLeE }
3440         \_unravel_scan_pdf_ext_toks:
3441     }
3442 }
3443 { 8 } % pdfrefobj
3444 { \_unravel_scan_int: }
3445 { 9 } % pdfxform
3446 {
3447     \_unravel_scan_keyword:nT { aAtTtTrR }
3448     { \_unravel_scan_pdf_ext_toks: }
3449     \_unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3450     { \_unravel_scan_pdf_ext_toks: }
3451     \_unravel_scan_int:
3452 }
3453 { 10 } % pdfrefxform
3454 { \_unravel_scan_int: }
3455 { 11 } % pdfximage
3456 { \_unravel_scan_image: }
3457 { 12 } % pdfrefximage

```

```

3458     { \_unravel_scan_int: }
3459 { 13 } % pdfannot
3460 {
3461     \_unravel_scan_keyword:nTF
3462     { rReEsSeErRvVeEoObBjJnNuUmM }
3463     { \_unravel_scan_optional_space: }
3464     {
3465         \_unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3466         { \_unravel_scan_int: }
3467         \_unravel_scan_alt_rule:
3468         \_unravel_scan_pdf_ext_toks:
3469     }
3470 }
3471 { 14 } % pdfstartlink
3472 {
3473     \mode_if_vertical:TF
3474     { \msg_error:nn { unravel } { invalid-mode } }
3475     {
3476         \_unravel_scan_rule_attr:
3477         \_unravel_scan_action:
3478     }
3479 }
3480 { 15 } % pdfendlink
3481 {
3482     \mode_if_vertical:T
3483     { \msg_error:nn { unravel } { invalid-mode } }
3484 }
3485 { 16 } % pdfoutline
3486 {
3487     \_unravel_scan_keyword:nT { aAtTtTrR }
3488     { \_unravel_scan_pdf_ext_toks: }
3489     \_unravel_scan_action:
3490     \_unravel_scan_keyword:nT { cCoOuUnNtT }
3491     { \_unravel_scan_int: }
3492     \_unravel_scan_pdf_ext_toks:
3493 }
3494 { 17 } % pdfdest
3495 { \_unravel_scan_pdfdest_operands: }
3496 { 18 } % pdfthread
3497 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3498 { 19 } % pdfstartthread
3499 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3500 { 20 } % pdfendthread
3501 { }
3502 { 21 } % pdfsavepos
3503 { }
3504 { 22 } % pdfinfo
3505 { \_unravel_scan_pdf_ext_toks: }
3506 { 23 } % pdfcatalog
3507 {

```

```

3508         \_unravel_scan_pdf_ext_toks:
3509         \_unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
3510         { \_unravel_scan_action: }
3511     }
3512 { 24 } % pdfnames
3513     { \_unravel_scan_pdf_ext_toks: }
3514 { 25 } % pdffontattr
3515     {
3516         \_unravel_scan_font_ident:
3517         \_unravel_scan_pdf_ext_toks:
3518     }
3519 { 26 } % pdfincludechars
3520     {
3521         \_unravel_scan_font_ident:
3522         \_unravel_scan_pdf_ext_toks:
3523     }
3524 { 27 } % pdfmapfile
3525     { \_unravel_scan_pdf_ext_toks: }
3526 { 28 } % pdfmapline
3527     { \_unravel_scan_pdf_ext_toks: }
3528 { 29 } % pdftrailer
3529     { \_unravel_scan_pdf_ext_toks: }
3530 { 30 } % pdfresettimer
3531     { }
3532 { 31 } % pdffontexpand
3533     {
3534         \_unravel_scan_font_ident:
3535         \_unravel_scan_optional_equals:
3536         \_unravel_scan_int:
3537         \_unravel_scan_int:
3538         \_unravel_scan_int:
3539         \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3540         { \_unravel_skip_optional_space: }
3541     }
3542 { 32 } % pdfsetrandomseed
3543     { \_unravel_scan_int: }
3544 { 33 } % pdfsnaprefpoint
3545     { }
3546 { 34 } % pdfsnapy
3547     { \_unravel_scan_normal_glue: }
3548 { 35 } % pdfsnapycomp
3549     { \_unravel_scan_int: }
3550 { 36 } % pdfglyphtounicode
3551     {
3552         \_unravel_scan_pdf_ext_toks:
3553         \_unravel_scan_pdf_ext_toks:
3554     }
3555 { 37 } % pdfcolorstack
3556     { \_unravel_scan_pdfcolorstack_operands: }
3557 { 38 } % pdfsetmatrix

```

```

3558         { \_unravel_scan_pdf_ext_toks: }
3559     { 39 } % pdfsave
3560     { }
3561     { 40 } % pdfrestore
3562     { }
3563     { 41 } % pdfnombuiltintounicode
3564     { \_unravel_scan_font_ident: }
3565 }
3566 { } % no other cases.
3567 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```

3568 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstack_operands:
3569 {
3570     \_unravel_scan_int:
3571     \_unravel_scan_keyword:nF { sSeEtT }
3572     {
3573         \_unravel_scan_keyword:nF { pPuUsShH }
3574         {
3575             \_unravel_scan_keyword:nF { pPoOpP }
3576             {
3577                 \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
3578                 {
3579                     \msg_error:nn { unravel }
3580                     { color-stack-action-missing }
3581                 }
3582             }
3583         }
3584     }
3585 }

```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```

3586 \cs_new_protected_nopar:Npn \_unravel_scan_rule_attr:
3587 {
3588     \_unravel_scan_alt_rule:
3589     \_unravel_scan_keyword:nT { aAtTtTrR }
3590     { \_unravel_scan_pdf_ext_toks: }
3591 }

```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```

3592 \cs_new_protected_nopar:Npn \_unravel_scan_action:
3593 {
3594     \_unravel_scan_keyword:nTF { uUsSeErR }
3595     { \_unravel_scan_pdf_ext_toks: }

```

```

3596 {
3597   \_unravel_scan_keyword:nF { gGoOtToO }
3598   {
3599     \_unravel_scan_keyword:nF { tThHrReEaAdD }
3600     { \msg_error:nn { unravel } { action-type-missing } }
3601   }
3602 }
3603 \_unravel_scan_keyword:nT { fFiIlLeE }
3604 { \_unravel_scan_pdf_ext_toks: }
3605 \_unravel_scan_keyword:nTF { pPaAgGeE }
3606 {
3607   \_unravel_scan_int:
3608   \_unravel_scan_pdf_ext_toks:
3609 }
3610 {
3611   \_unravel_scan_keyword:nTF { nNaAmMeE }
3612   { \_unravel_scan_pdf_ext_toks: }
3613   {
3614     \_unravel_scan_keyword:nTF { nNuUmM }
3615     { \_unravel_scan_int: }
3616     { \msg_error:nn { unravel } { identifier-type-missing } }
3617   }
3618 }
3619 \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3620 { \_unravel_skip_optional_space: }
3621 {
3622   \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3623   { \_unravel_skip_optional_space: }
3624 }
3625 }

```

(End definition for _unravel_scan_action:.)

_unravel_scan_image: Used by \pdfximage.

```

3626 \cs_new_protected_nopar:Npn \_unravel_scan_image:
3627 {
3628   \_unravel_scan_rule_attr:
3629   \_unravel_scan_keyword:nTF { nNaAmMeEdD }
3630   { \_unravel_scan_pdf_ext_toks: }
3631   {
3632     \_unravel_scan_keyword:nT { pPaAgGeE }
3633     { \_unravel_scan_int: }
3634   }
3635   \_unravel_scan_keyword:nT { cCoO1LoOrRsSpPaAcCeE }
3636   { \_unravel_scan_int: }
3637   \_unravel_scan_pdf_ext_toks:
3638 }

```

(End definition for _unravel_scan_image:.)

_unravel_scan_immediate_operands:

```
3639 \cs_new_protected_nopar:Npn \__unravel_scan_immediate_operands:
3640 {
3641   \__unravel_get_x_next:
3642   \__unravel_set_cmd:
3643   \int_compare:nNnTF
3644     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
3645     {
3646       \int_compare:nNnTF
3647         \l__unravel_head_char_int < { 3 } % openout, write, closeout
3648         { \__unravel_scan_immediate_operands_aux: }
3649         {
3650           \int_case:nnF \l__unravel_head_char_int
3651             {
3652               { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
3653               { 9 } { \__unravel_scan_extension_operands_aux: } % pdfxform
3654               { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
3655             }
3656             { \__unravel_scan_immediate_operands_bad: }
3657           }
3658         }
3659       { \__unravel_scan_immediate_operands_bad: }
3660     }
3661 \cs_new_protected_nopar:Npn \__unravel_scan_immediate_operands_aux:
3662 {
3663   \__unravel_prev_input:V \l__unravel_head_tl
3664   \__unravel_scan_extension_operands:
3665 }
3666 \cs_new_protected_nopar:Npn \__unravel_scan_immediate_operands_bad:
3667 {
3668   \__unravel_back_input:
3669   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3670   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3671   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3672 }
3673
```

(End definition for __unravel_scan_immediate_operands:.)

_unravel_scan_pdfdest_operands:

```
3674 \cs_new_protected_nopar:Npn \__unravel_scan_pdfdest_operands:
3675 {
3676   \__unravel_scan_keyword:nTF { nNuUmM }
3677     { \__unravel_scan_int: }
3678     {
3679       \__unravel_scan_keyword:nTF { nNaAmMeE }
3680         { \__unravel_scan_pdf_ext_toks: }
3681         { \msg_error:nn { unravel } { identifier-type-missing } }
3682       }
3683   \__unravel_scan_keyword:nTF { xXyYzZ }
```

```

3684 {
3685   \_unravel_scan_keyword:nT { zZoOoOmM }
3686   { \_unravel_scan_int: }
3687 }
3688 {
3689   \_unravel_scan_keyword:nF { fFiItTbBhH }
3690   {
3691     \_unravel_scan_keyword:nF { fFiItTbBvV }
3692     {
3693       \_unravel_scan_keyword:nF { fFiItTbB }
3694       {
3695         \_unravel_scan_keyword:nF { fFiItThHhH }
3696         {
3697           \_unravel_scan_keyword:nF { fFiItTvV }
3698           {
3699             \_unravel_scan_keyword:nTF
3700             { fFiItTrR }
3701             {
3702               \_unravel_skip_optional_space:
3703               \_unravel_scan_alt_rule:
3704               \use_none:n
3705             }
3706             {
3707               \_unravel_scan_keyword:nF
3708               { fFiItT }
3709               {
3710                 \msg_error:nn { unravel }
3711                 {
3712                   destination-type-missing
3713                 }
3714               }
3715             }
3716           }
3717         }
3718       }
3719     }
3720   }
3721 }
3722 \_unravel_skip_optional_space:
3723 }

```

(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`.

```

3724 \cs_set_protected_nopar:Npn \__unravel_tmp:w
3725 {
3726   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3727   \__unravel_prefixed_command:
3728 }
3729 \int_step_inline:nnnn
3730 { \__unravel_tex_use:n { max_non_prefixed_command } + 1 }
3731 { 1 }
3732 { \__unravel_tex_use:n { max_command } }
3733 { \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.

```

3734 \cs_new_protected_nopar:Npn \__unravel_prefixed_command:
3735 {
3736   \int_while_do:nNnn
3737     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
3738     {
3739       \__unravel_prev_input:V \l__unravel_head_tl
3740       \__unravel_get_x_non_relax:
3741       \__unravel_set_cmd:
3742       \int_compare:nNnF \l__unravel_head_cmd_int
3743         > { \__unravel_tex_use:n { max_non_prefixed_command } }
3744         {
3745           \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3746           \msg_error:nnxx { unravel } { erroneous-prefixes }
3747             { \tl_to_str:N \l__unravel_tmpa_tl }
3748             { \tl_to_str:N \l__unravel_head_tl }
3749           \__unravel_back_input:
3750           \__unravel_omit_after_assignment:w
3751         }
3752       }
3753       % ^^A todo: Discard non-\global prefixes if they are irrelevant
3754       % ^^A todo: Adjust for the setting of \globaldefs
3755       \cs_if_exist_use:cF
3756         { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3757         {
3758           \msg_error:nnx { unravel } { internal } { prefixed }
3759           \__unravel_omit_after_assignment:w
3760         }
3761       \__unravel_after_assignment:
3762     }

```

(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

```

```

3763 \cs_new_protected_nopar:Npn \__unravel_after_assignment:
3764 {
3765   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3766   \gtl_gclear:N \g__unravel_after_assignment_gtl
3767 }
3768 \cs_new_protected_nopar:Npn \__unravel_omit_after_assignment:w
3769 #1 \__unravel_after_assignment: { }

```

(End definition for __unravel_after_assignment:.)

__unravel_prefixed_new:nn

```

3770 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3771 {
3772   \cs_new_protected_nopar:cpn
3773   { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3774 }

```

(End definition for __unravel_prefixed_new:nn.)

__unravel_assign_token:n

```

3775 \cs_new_protected:Npn \__unravel_assign_token:n #1
3776 {
3777   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3778   #1
3779   \tl_use:N \l__unravel_head_tl \scan_stop:
3780   \__unravel_print_assigned_token:
3781 }

```

(End definition for __unravel_assign_token:n.)

__unravel_assign_register:

```

3782 \cs_new_protected_nopar:Npn \__unravel_assign_register:
3783 {
3784   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3785   \tl_use:N \l__unravel_head_tl \scan_stop:
3786   \__unravel_print_assigned_register:
3787 }

```

(End definition for __unravel_assign_register:.)

__unravel_assign_value:nn

```

3788 \cs_new_protected:Npn \__unravel_assign_value:nn #1#2
3789 {
3790   \tl_if_empty:nF {#1}
3791   {
3792     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3793     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3794     #1
3795     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3796   }
3797   \__unravel_prev_input:V \l__unravel_head_tl

```

```

3798 \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3799 \__unravel_scan_optional_equals:
3800 #2
3801 \__unravel_assign_register:
3802 }

```

(End definition for __unravel_assign_value:nn.)

__unravel_assign_toks:

```

3803 \__unravel_prefixed_new:nn { toks_register } % 71
3804 {
3805 \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3806 { % \toks
3807 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3808 \__unravel_print_action:
3809 \__unravel_scan_int:
3810 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3811 }
3812 \__unravel_assign_toks:
3813 }
3814 \__unravel_prefixed_new:nn { assign_toks } % 72
3815 { \__unravel_assign_toks: }
3816 \cs_new_protected_nopar:Npn \__unravel_assign_toks:
3817 {
3818 \__unravel_prev_input_silent:V \l__unravel_head_tl
3819 \__unravel_print_action:
3820 \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3821 \__unravel_scan_optional_equals:
3822 \__unravel_get_x_non_relax:
3823 \__unravel_set_cmd:
3824 \int_compare:nNnTF
3825 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
3826 {
3827 \__unravel_prev_input:V \l__unravel_head_tl
3828 \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3829 { \__unravel_scan_int: }
3830 }
3831 {
3832 \int_compare:nNnTF
3833 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
3834 { \__unravel_prev_input:V \l__unravel_head_tl }
3835 {
3836 \__unravel_back_input:
3837 \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3838 }
3839 }
3840 \__unravel_assign_register:
3841 }

```

(End definition for __unravel_assign_toks:.)

```

3842 \__unravel_prefixed_new:nn { assign_int } % 73
3843 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3844 \__unravel_prefixed_new:nn { assign_dimen } % 74
3845 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3846 \__unravel_prefixed_new:nn { assign_glue } % 75
3847 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
3848 \__unravel_prefixed_new:nn { assign_mu_glue } % 76
3849 { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
3850 \__unravel_prefixed_new:nn { assign_font_dimen } % 77
3851 {
3852 \__unravel_assign_value:nn
3853 { \__unravel_scan_int: \__unravel_scan_font_ident: }
3854 { \__unravel_scan_normal_dimen: }
3855 }
3856 \__unravel_prefixed_new:nn { assign_font_int } % 78
3857 {
3858 \__unravel_assign_value:nn
3859 { \__unravel_scan_font_int: } { \__unravel_scan_int: }
3860 }
3861 \__unravel_prefixed_new:nn { set_aux } % 79
3862 { % prevdepth = 1, spacefactor = 102
3863 \int_compare:nNnTF \l__unravel_head_char_int = \c_one
3864 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3865 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3866 }
3867 \__unravel_prefixed_new:nn { set_prev_graf } % 80
3868 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3869 \__unravel_prefixed_new:nn { set_page_dimen } % 81
3870 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3871 \__unravel_prefixed_new:nn { set_page_int } % 82
3872 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3873 \__unravel_prefixed_new:nn { set_box_dimen } % 83
3874 {
3875 \__unravel_assign_value:nn
3876 { \__unravel_scan_int: } { \__unravel_scan_normal_dimen: }
3877 }
3878 \__unravel_prefixed_new:nn { set_shape } % 84
3879 {
3880 \__unravel_assign_value:nn { \__unravel_scan_int: }
3881 {
3882 \prg_replicate:nn
3883 {
3884 \tl_if_head_eq_meaning:VNT
3885 \l__unravel_defined_tl \tex_parshape:D { \c_two * }
3886 \tl_tail:N \l__unravel_defined_tl
3887 }
3888 { \__unravel_scan_int: }
3889 }
3890 }

```

```

3891 \__unravel_prefixed_new:nn { def_code } % 85
3892 {
3893   \__unravel_assign_value:nn
3894   { \__unravel_scan_int: } { \__unravel_scan_int: }
3895 }
3896 \__unravel_prefixed_new:nn { def_family } % 86
3897 {
3898   \__unravel_assign_value:nn
3899   { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
3900 }
3901 \__unravel_prefixed_new:nn { set_font } % 87
3902 {
3903   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3904   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3905   \tl_use:N \l__unravel_head_tl \scan_stop:
3906   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3907   \__unravel_print_action:
3908 }
3909 \__unravel_prefixed_new:nn { def_font } % 88
3910 {
3911   \__unravel_prev_input_silent:V \l__unravel_head_tl
3912   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3913   \__unravel_scan_r_token:
3914   \__unravel_print_action:x
3915   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
3916   \__unravel_scan_optional_equals:
3917   \__unravel_scan_file_name:
3918   \bool_gset_true:N \g__unravel_name_in_progress_bool
3919   \__unravel_scan_keyword:nTF { aAtT }
3920   { \__unravel_scan_normal_dimen: }
3921   {
3922     \__unravel_scan_keyword:nT { sScCaAlLeEdD }
3923     { \__unravel_scan_int: }
3924   }
3925   \bool_gset_false:N \g__unravel_name_in_progress_bool
3926   \__unravel_assign_token:n { }
3927 }

```

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

let, futurelet

```

3928 \__unravel_prefixed_new:nn { let } % 94
3929 {
3930   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3931   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
3932   { % |let|
3933     \__unravel_scan_r_token:
3934     \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3935     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3936     \__unravel_get_next:

```

```

3937     \bool_while_do:nn
3938         { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
3939         { \__unravel_get_next: }
3940     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
3941         { \__unravel_get_next: }
3942     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
3943         { \__unravel_get_next: }
3944     }
3945     { % |futurelet|
3946         \__unravel_scan_r_token:
3947         \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3948         \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3949         \__unravel_get_next:
3950         \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
3951         \__unravel_get_next:
3952         \__unravel_back_input:
3953         \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
3954         \__unravel_back_input:
3955     }
3956     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3957     \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
3958     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3959     \use:x
3960     {
3961         \exp_not:V \l__unravel_head_tl
3962         \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
3963     }
3964     \__unravel_print_assigned_token:
3965 }
3966 \__unravel_prefixed_new:nn { shorthand_def } % 95
3967 {
3968     \__unravel_prev_input_silent:V \l__unravel_head_tl
3969     \tl_set:Nx \l__unravel_prev_action_tl
3970         { \tl_to_str:N \l__unravel_head_tl }
3971     \__unravel_scan_r_token:
3972     \__unravel_print_action:x
3973         { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
3974     \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
3975     \__unravel_scan_optional_equals:
3976     \__unravel_scan_int:
3977     \__unravel_assign_token:n { }
3978 }

```

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


```

3979 \__unravel_prefixed_new:nn { read_to_cs } % 96
3980 {
3981   \__unravel_prev_input_silent:V \l__unravel_head_tl
3982   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3983   \__unravel_scan_int:
3984   \__unravel_scan_to:
3985   \__unravel_scan_r_token:
3986   \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3987   \__unravel_read_to_cs_safe:FTF
3988   { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
3989   { \__unravel_assign_token:n { } }
3990   {
3991     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3992     \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
3993   }
3994 }
3995 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
3996 {
3997   \int_compare:nNnTF { \etex_interactionmode:D } > { 1 }
3998   { \prg_return_true: }
3999   {
4000     \int_compare:nNnTF {#1} < { 0 }
4001     { \prg_return_false: }
4002     {
4003       \int_compare:nNnTF {#1} > { 15 }
4004       { \prg_return_false: }
4005       {
4006         \tex_ifeof:D #1 \exp_stop_f:
4007         \prg_return_false:
4008         \else:
4009         \prg_return_true:
4010         \fi:
4011       }
4012     }
4013   }
4014 }
4015 \cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }
(End definition for \__unravel_read_to_cs_safe:nTF and \__unravel_read_to_cs_safe:FTF.)
4016 \__unravel_prefixed_new:nn { def } % 97
4017 {
4018   \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4019   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4020   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4021   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4022   \int_compare:nNnTF \l__unravel_head_char_int < \c_two
4023   { % def/gdef
4024     \__unravel_scan_r_token:
4025     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4026     \__unravel_scan_toks:NN \c_true_bool \c_false_bool

```

```

4027     }
4028     { % edef/xdef
4029       \__unravel_scan_r_token:
4030       \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4031       \__unravel_scan_toks:NN \c_true_bool \c_true_bool
4032     }
4033     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4034     \__unravel_prev_input:V \l__unravel_head_tl
4035     \__unravel_assign_token:n
4036     { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4037   }

```

\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.

```

4038 \__unravel_prefixed_new:nn { set_box } % 98
4039 {
4040   \__unravel_prev_input:V \l__unravel_head_tl
4041   \__unravel_scan_int:
4042   \__unravel_scan_optional_equals:
4043   \bool_if:NTF \g__unravel_set_box_allowed_bool
4044   { \__unravel_do_box:N \c_false_bool }
4045   {
4046     \msg_error:nn { unravel } { improper-setbox }
4047     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4048     \__unravel_omit_after_assignment:w
4049   }
4050 }

```

\hyphenation and \patterns

```

4051 \__unravel_prefixed_new:nn { hyph_data } % 99
4052 {
4053   \__unravel_prev_input:V \l__unravel_head_tl
4054   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4055   \__unravel_assign_token:n { }
4056 }
4057 \__unravel_prefixed_new:nn { set_interaction } % 100
4058 {
4059   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4060   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4061   \tl_use:N \l__unravel_head_tl \scan_stop:
4062   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4063 }
4064 \__unravel_prefixed_new:nn { letterspace_font } % 101
4065 {
4066   \__unravel_prev_input_silent:V \l__unravel_head_tl
4067   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4068   \__unravel_scan_r_token:
4069   \__unravel_print_action:x

```

```

4070     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4071 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4072 \__unravel_scan_optional_equals:
4073 \__unravel_scan_font_ident:
4074 \__unravel_scan_int:
4075 \__unravel_assign_token:n { }
4076 }
4077 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4078 {
4079 \__unravel_prev_input_silent:V \l__unravel_head_tl
4080 \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4081 \__unravel_scan_r_token:
4082 \__unravel_print_action:x
4083 { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4084 \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4085 \__unravel_scan_optional_equals:
4086 \__unravel_scan_font_ident:
4087 \__unravel_assign_token:n { }
4088 }

```

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

```

4089 \__unravel_prefixed_new:nn { register } % 89
4090 { \__unravel_do_register:N \c_zero }
4091 \__unravel_prefixed_new:nn { advance } % 90
4092 { \__unravel_do_operation:N \c_one }
4093 \__unravel_prefixed_new:nn { multiply } % 91
4094 { \__unravel_do_operation:N \c_two }
4095 \__unravel_prefixed_new:nn { divide } % 92
4096 { \__unravel_do_operation:N \c_three }

```

`__unravel_do_operation:N`

`__unravel_do_operation_fail:w`

```

4097 \cs_new_protected:Npn \__unravel_do_operation:N #1
4098 {
4099 \__unravel_prev_input_silent:V \l__unravel_head_tl
4100 \__unravel_print_action:
4101 \__unravel_get_x_next:
4102 \__unravel_set_cmd:
4103 \int_compare:nNnTF
4104 \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4105 {
4106 \int_compare:nNnTF
4107 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4108 { \__unravel_do_register:N #1 }
4109 { \__unravel_do_operation_fail:w }
4110 }
4111 {
4112 \int_compare:nNnTF
4113 \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }

```

```

4114     { \_unravel_do_operation_fail:w }
4115     {
4116       \_unravel_prev_input:V \l\_unravel_head_tl
4117       \exp_args:NNf \_unravel_do_register_set:Nn #1
4118       {
4119         \int_eval:n
4120         {
4121           \l\_unravel_head_cmd_int
4122           - \_unravel_tex_use:n { assign_toks }
4123         }
4124       }
4125     }
4126   }
4127 }
4128 \cs_new_protected_nopar:Npn \_unravel_do_operation_fail:w
4129 {
4130   \msg_error:nn { unravel } { after-advance }
4131   \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_tmpa_tl
4132   \_unravel_omit_after_assignment:w
4133 }

```

(End definition for _unravel_do_operation:N and _unravel_do_operation_fail:w.)

_unravel_do_register:N
 _unravel_do_register_aux:Nn

```

4134 \cs_new_protected:Npn \_unravel_do_register:N #1
4135 {
4136   \exp_args:NNV \_unravel_do_register_aux:Nn #1
4137   \l\_unravel_head_char_int
4138 }
4139 \cs_new_protected:Npn \_unravel_do_register_aux:Nn #1#2
4140 {
4141   \int_compare:nNnTF { \tl_tail:n {#2} } = \c_zero
4142   {
4143     \seq_gput_right:NV \g\_unravel_prev_input_seq \l\_unravel_head_tl
4144     \_unravel_print_action:
4145     \_unravel_scan_int:
4146     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
4147     \_unravel_prev_input_silent:V \l\_unravel_head_tl
4148   }
4149   {
4150     \_unravel_prev_input_silent:V \l\_unravel_head_tl
4151     \_unravel_print_action:
4152   }
4153   \tl_set_eq:NN \l\_unravel_defined_tl \l\_unravel_head_tl
4154   \exp_args:NNf \_unravel_do_register_set:Nn #1
4155   { \int_eval:n { #2 / 1 000 000 } }
4156 }

```

(End definition for _unravel_do_register:N and _unravel_do_register_aux:Nn.)

`_unravel_do_register_set:Nn`

```
4157 \cs_new_protected:Npn \_unravel_do_register_set:Nn #1#2
4158 {
4159   \int_compare:nNnTF {#1} = \c_zero
4160     { % truly register command
4161       \_unravel_scan_optional_equals:
4162     }
4163     { % \advance, \multiply, \divide
4164       \_unravel_scan_keyword:nF { bByY }
4165       { \_unravel_prev_input_silent:n { by } }
4166     }
4167   \int_compare:nNnTF {#1} < \c_two
4168     {
4169       \int_case:nnF {#2}
4170       {
4171         { 1 } { \_unravel_scan_int:          } % count
4172         { 2 } { \_unravel_scan_normal_dimen: } % dim
4173         { 3 } { \_unravel_scan_normal_glue:  } % glue
4174         { 4 } { \_unravel_scan_mu_glue:     } % muglue
4175       }
4176       { \msg_error:nx { unravel } { internal } { do-reg=#2 } }
4177     }
4178     { \_unravel_scan_int: }
4179   \_unravel_assign_register:
4180 }
```

(End definition for _unravel_do_register_set:Nn.)

The following is used for instance when making accents.

```
4181 \cs_new_protected_nopar:Npn \_unravel_do_assignments:
4182 {
4183   \_unravel_get_x_non_relax:
4184   \_unravel_set_cmd:
4185   \int_compare:nNnT
4186     \l__unravel_head_cmd_int
4187     > { \_unravel_tex_use:n { max_non_prefixed_command } }
4188   {
4189     \bool_gset_false:N \g__unravel_set_box_allowed_bool
4190     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
4191     \_unravel_prefixed_command:
4192     \bool_gset_true:N \g__unravel_set_box_allowed_bool
4193     \_unravel_do_assignments:
4194   }
4195 }
```

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`, `\pdfscapestring`, `\pdfescapename`, `\leftmarginkern`, `\rightmarginkern`, `\pdfstrcmp`, `\pdfcolorstackinit`, `\pdfscapehex`, `\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.

```

4196 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
4197 { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }

\__unravel_expandafter:
  \__unravel_unless:
\__unravel_unless_bad:
4198 \__unravel_new_tex_expandable:nn { expand_after } % 104
4199 {
4200   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4201   { \__unravel_expandafter: } { \__unravel_unless: }
4202 }
4203 \cs_new_protected_nopar:Npn \__unravel_expandafter:
4204 {
4205   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4206   \__unravel_get_next:
4207   \gtl_concat:NNN \l__unravel_head_gtl
4208   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4209   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_gtl
4210   \__unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
4211   \__unravel_get_next:

```

```

4212 \__unravel_token_if_expandable:NTF \l__unravel_head_token
4213 { \__unravel_expand: }
4214 { \__unravel_back_input: }
4215 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_gtl
4216 \__unravel_set_action_text:x
4217 { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4218 \gtl_pop_left:N \l__unravel_head_gtl
4219 \__unravel_back_input:
4220 \__unravel_print_action:
4221 }
4222 \cs_new_protected_nopar:Npn \__unravel_unless:
4223 {
4224 \__unravel_get_token:
4225 \int_compare:nNnTF
4226 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4227 {
4228 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4229 { \__unravel_unless_bad: }
4230 {
4231 \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4232 % \int_add:Nn \l__unravel_head_char_int { 32 }
4233 \__unravel_expand_nonmacro:
4234 }
4235 }
4236 { \__unravel_unless_bad: }
4237 }
4238 \cs_new_protected_nopar:Npn \__unravel_unless_bad:
4239 {
4240 \msg_error:nn { unravel } { bad-unless }
4241 \__unravel_back_input:
4242 }

```

(End definition for __unravel_expandafter:, __unravel_unless:, and __unravel_unless_bad:.)

```

\__unravel_noexpand:
\__unravel_pdfprimitive:
4243 \__unravel_new_tex_expandable:nn { no_expand } % 105
4244 {
4245 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4246 { \__unravel_noexpand: }
4247 { \__unravel_pdfprimitive: }
4248 }
4249 \cs_new_protected_nopar:Npn \__unravel_noexpand:
4250 {
4251 \__unravel_get_token:
4252 \__unravel_back_input:
4253 \__unravel_token_if_expandable:NT \l__unravel_head_token
4254 {
4255 \cs_gset_protected_nopar:Npx \__unravel_get_next:
4256 {
4257 \cs_gset_protected_nopar:Npn \__unravel_get_next:

```

```

4258         { \exp_not:o { \__unravel_get_next: } }
4259         \exp_not:o { \__unravel_get_next: }
4260         \exp_not:n { \cs_set_eq:NN \l__unravel_head_token \tex_relax:D }
4261     }
4262 }
4263 }
4264 \cs_new_protected_nopar:Npn \__unravel_pdfprimitive:
4265 { \msg_error:nxx { unravel } { not-implemented } { pdfprimitive } }

```

(End definition for __unravel_noexpand: and __unravel_pdfprimitive:.)

```

\__unravel_endinput:
\__unravel_scantokens: 4266 \__unravel_new_tex_expandable:nn { input } % 106
\__unravel_input:      4267 {
4268     \int_case:nnF \l__unravel_head_char_int
4269     {
4270         { 1 } { \__unravel_endinput: } % \endinput
4271         { 2 } { \__unravel_scantokens: } % \scantokens
4272     }
4273     { % 0=\input
4274         \bool_if:NTF \g__unravel_name_in_progress_bool
4275         { \__unravel_insert_relax: } { \__unravel_input: }
4276     }
4277 }
4278 \cs_new_protected_nopar:Npn \__unravel_endinput:
4279 {
4280     \msg_warning:nn { unravel } { endinput-ignored }
4281     \__unravel_print_action:
4282 }
4283 \cs_new_protected_nopar:Npn \__unravel_scantokens:
4284 {
4285     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
4286     \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4287     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4288     \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4289     \__unravel_back_input:V \l__unravel_head_tl
4290     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4291 }
4292 \cs_new_protected_nopar:Npn \__unravel_input:
4293 {
4294     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4295     \__unravel_scan_file_name:
4296     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4297     \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4298     \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4299     \__unravel_back_input:V \l__unravel_tmpa_tl
4300     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4301 }

```

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

`__unravel_csname_loop:`

```
4302 \__unravel_new_tex_expandable:nn { cs_name } % 109
4303 {
4304   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4305   \__unravel_print_action:
4306   \__unravel_csname_loop:
4307   \__unravel_prev_input:V \l__unravel_head_tl
4308   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4309   \__unravel_back_input_tl_o:
4310 }
4311 \cs_new_protected_nopar:Npn \__unravel_csname_loop:
4312 {
4313   \__unravel_get_x_next:
4314   \token_if_cs:NTF \l__unravel_head_token
4315   {
4316     \cs_if_eq:NMF \l__unravel_head_token \tex_endcsname:D
4317     {
4318       \msg_error:nn { unravel } { missing-endcsname }
4319       \__unravel_back_input:
4320       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4321     }
4322   }
4323   {
4324     \__unravel_prev_input_silent:x
4325     { \__unravel_token_to_char:N \l__unravel_head_token }
4326     \__unravel_csname_loop:
4327   }
4328 }
```

(End definition for __unravel_csname_loop:.)

```
4329 \__unravel_new_tex_expandable:nn { convert } % 110
4330 {
4331   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4332   \__unravel_print_action:
4333   \int_case:nn \l__unravel_head_char_int
4334   {
4335     0 \__unravel_scan_int:
4336     1 \__unravel_scan_int:
4337     2 { \__unravel_get_next: \__unravel_prev_input:V \l__unravel_head_tl }
4338     3 { \__unravel_get_next: \__unravel_prev_input:V \l__unravel_head_tl }
4339     4 \__unravel_scan_font_ident:
4340     8 \__unravel_scan_font_ident:
4341     9 \__unravel_scan_font_ident:
4342     { 10 } \__unravel_scan_font_ident:
4343     { 11 } \__unravel_scan_int:
4344     { 12 } \__unravel_scan_int:
4345     { 13 } \__unravel_scan_pdf_ext_toks:
4346     { 14 } \__unravel_scan_pdf_ext_toks:
4347     { 15 } \__unravel_scan_int:
```

```

4348     { 16 } \_unravel_scan_int:
4349     { 17 } \_unravel_scan_pdfstrcmp:
4350     { 18 } \_unravel_scan_pdfcolorstackinit:
4351     { 19 } \_unravel_scan_pdf_ext_toks:
4352     { 20 } \_unravel_scan_pdf_ext_toks:
4353     { 22 } \_unravel_scan_pdf_ext_toks:
4354     { 23 } \_unravel_scan_pdf_ext_toks:
4355     { 24 }
4356     {
4357         \_unravel_scan_keyword:n { fFiLLeE }
4358         \_unravel_scan_pdf_ext_toks:
4359     }
4360     { 25 } \_unravel_scan_pdffiledump:
4361     { 26 } \_unravel_scan_pdfmatch:
4362     { 27 } \_unravel_scan_int:
4363     { 28 } \_unravel_scan_int:
4364     { 30 } \_unravel_scan_int:
4365     { 31 } \_unravel_scan_pdfximagebbox:
4366 }
4367 \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_head_tl
4368 \_unravel_back_input_tl_o:
4369 }
4370 \cs_new_protected_nopar:Npn \_unravel_scan_pdfstrcmp:
4371 {
4372     \_unravel_scan_toks_to_str:
4373     \_unravel_scan_toks_to_str:
4374 }
4375 \cs_new_protected_nopar:Npn \_unravel_scan_pdfximagebbox:
4376 { \_unravel_scan_int: \_unravel_scan_int: }
4377 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstackinit:
4378 {
4379     \_unravel_scan_keyword:nTF { pPaAgGeE }
4380     { \bool_set_true:N \l_unravel_tmpa_bool }
4381     { \bool_set_false:N \l_unravel_tmpb_bool }
4382     \_unravel_scan_keyword:nF { dDiIrReEcCtT }
4383     { \_unravel_scan_keyword:n { pPaAgGeE } }
4384     \_unravel_scan_toks_to_str:
4385 }
4386 \cs_new_protected_nopar:Npn \_unravel_scan_pdffiledump:
4387 {
4388     \_unravel_scan_keyword:nT { oOfFfFsSeEtT } \_unravel_scan_int:
4389     \_unravel_scan_keyword:nT { lLeEnNgGtThH } \_unravel_scan_int:
4390     \_unravel_scan_pdf_ext_toks:
4391 }
4392 \cs_new_protected_nopar:Npn \_unravel_scan_pdfmatch:
4393 {
4394     \_unravel_scan_keyword:n { iIcCaAsSeE }
4395     \_unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
4396     { \_unravel_scan_int: }
4397     \_unravel_scan_pdf_ext_toks:

```

```

4398   \__unravel_scan_pdf_ext_toks:
4399   }

\__unravel_get_the:

4400 \__unravel_new_tex_expandable:nn { the }                % 111
4401   {
4402   \__unravel_get_the:
4403   \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4404   \__unravel_back_input:V \l__unravel_tmpa_tl
4405   \__unravel_print_action:
4406   }
4407 \cs_new_protected_nopar:Npn \__unravel_get_the:
4408   {
4409   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4410   \__unravel_print_action:
4411   \int_if_odd:nTF \l__unravel_head_char_int
4412     { % \unexpanded, \detokenize
4413       \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4414       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4415       \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4416     }
4417     { % \the
4418       \__unravel_get_x_next:
4419       \__unravel_scan_something_internal:n { 5 }
4420       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4421       \__unravel_set_action_text:x
4422         {
4423           \tl_head:N \l__unravel_head_tl
4424           => \tl_tail:N \l__unravel_head_tl
4425         }
4426       \tl_set:Nx \l__unravel_head_tl
4427         { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4428     }
4429   }

```

(End definition for __unravel_get_the:.)

```

4430 \__unravel_new_tex_expandable:nn { top_bot_mark }      % 112
4431   { \__unravel_back_input_tl_o: }

4432 \__unravel_new_tex_expandable:nn { end_template }     % 117
4433   {
4434   \msg_error:nn { unravel } { not-implemented } { end-template }
4435   \__unravel_back_input_tl_o:
4436   }

```

2.14.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w
4437 \cs_new_protected_nopar:Npn \__unravel_pass_text:
4438   {

```

```

4439 \__unravel_input_if_empty:TF
4440 { \__unravel_pass_text_empty: }
4441 {
4442   \__unravel_input_get:N \l__unravel_tmpb_gtl
4443   \if_true:
4444     \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one
4445     \exp_after:wN \__unravel_pass_text_done:w
4446     \fi:
4447     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4448     \exp_after:wN \__unravel_pass_text:
4449   \else:
4450     \use:c { fi: }
4451     \int_set_eq:NN \l__unravel_if_nesting_int \c_one
4452     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4453     \exp_after:wN \__unravel_pass_text_nested:
4454     \fi:
4455   }
4456 }
4457 \cs_new_protected_nopar:Npn \__unravel_pass_text_done:w
4458 {
4459   \__unravel_get_next:
4460   \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4461   \else:
4462   }

```

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

```

4463 \cs_new_protected_nopar:Npn \__unravel_pass_text_nested:
4464 {
4465   \__unravel_input_if_empty:TF
4466   { \__unravel_pass_text_empty: }
4467   {
4468     \__unravel_input_get:N \l__unravel_tmpb_gtl
4469     \if_true:
4470     \if_true:
4471       \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4472       \int_decr:N \l__unravel_if_nesting_int

```

```

4473         \use_none:nnnnn
4474         \fi:
4475         \use_none:nnn
4476         \fi:
4477         \int_incr:N \l__unravel_if_nesting_int
4478         \fi:
4479         \__unravel_input_gpop:N \l__unravel_unused_gtl
4480         \int_compare:nNnTF \l__unravel_if_nesting_int = \c_zero
4481         { \__unravel_pass_text: }
4482         { \__unravel_pass_text_nested: }
4483     }
4484 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

4485 \cs_new_protected_nopar:Npn \__unravel_pass_text_empty:
4486 {
4487     \msg_error:nn { unravel } { runaway-if }
4488     \__unravel_exit:w
4489 }

```

(End definition for __unravel_pass_text_empty:.)

__unravel_cond_push:

__unravel_cond_pop:

```

4490 \cs_new_protected:Npn \__unravel_cond_push:
4491 {
4492     \tl_gput_left:Nx \g__unravel_if_limit_tl
4493     { { \int_use:N \g__unravel_if_limit_int } }
4494     \int_gincr:N \g__unravel_if_depth_int
4495     \int_gzero:N \g__unravel_if_limit_int
4496 }
4497 \cs_new_protected_nopar:Npn \__unravel_cond_pop:
4498 {
4499     \int_gset:Nn \g__unravel_if_limit_int
4500     { \tl_head:N \g__unravel_if_limit_tl }
4501     \tl_gset:Nx \g__unravel_if_limit_tl
4502     { \tl_tail:N \g__unravel_if_limit_tl }
4503     \int_gdecr:N \g__unravel_if_depth_int
4504 }

```

(End definition for __unravel_cond_push: and __unravel_cond_pop:.)

__unravel_change_if_limit:nn

```

4505 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
4506 {
4507     \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4508     { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4509     {
4510         \tl_clear:N \l__unravel_tpa_tl

```

```

4511 \prg_replicate:nn { \g__unravel_if_depth_int - #2 - \c_one }
4512 {
4513   \tl_put_right:Nx \l__unravel_tmpa_tl
4514   { { \tl_head:N \g__unravel_if_limit_tl } }
4515   \tl_gset:Nx \g__unravel_if_limit_tl
4516   { \tl_tail:N \g__unravel_if_limit_tl }
4517 }
4518 \tl_gset:Nx \g__unravel_if_limit_tl
4519 { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4520 }
4521 }

```

(End definition for __unravel_change_if_limit:nn.)

```

4522 \__unravel_new_tex_expandable:nn { if_test } % 107
4523 {
4524   \__unravel_cond_push:
4525   \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
4526 }

```

__unravel_cond_aux:nn

```

4527 \cs_new_protected:Npn \__unravel_cond_aux:n #1
4528 {
4529   \int_case:nnF \l__unravel_head_char_int
4530   {
4531     { 12 } { \__unravel_test_ifx:n {#1} }
4532     { 16 } { \__unravel_test_case:n {#1} }
4533     { 21 } { \__unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4534   }
4535   {
4536     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4537     \__unravel_print_action:
4538     \int_case:nn \l__unravel_head_char_int
4539     {
4540       { 0 } { \__unravel_test_two_chars: } % if
4541       { 1 } { \__unravel_test_two_chars: } % ifcat
4542       { 2 } % ifnum
4543       { \__unravel_test_two_vals:N \__unravel_scan_int: }
4544       { 3 } % ifdim
4545       { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
4546       { 4 } { \__unravel_scan_int: } % ifodd
4547       % { 5 } { } % ifvmode
4548       % { 6 } { } % ifhmode
4549       % { 7 } { } % ifmmode
4550       % { 8 } { } % ifinner
4551       { 9 } { \__unravel_scan_int: } % ifvoid
4552       { 10 } { \__unravel_scan_int: } % ifhbox
4553       { 11 } { \__unravel_scan_int: } % ifvbox
4554       { 13 } { \__unravel_scan_int: } % ifeof
4555       % { 14 } { } % iftrue
4556       % { 15 } { } % iffalse

```

```

4557         { 17 } { \_unravel_test_ifdefined: } % ifdefined
4558         { 18 } { \_unravel_test_ifcsname: } % ifcsname
4559         { 19 } % iffontchar
4560         { \_unravel_scan_font_ident: \_unravel_scan_int: }
4561         % { 20 } { } % ifincsname % ^A todo: something?
4562         { 22 } % ifpdfabsnum
4563         { \_unravel_test_two_vals:N \_unravel_scan_int: }
4564         { 23 } % ifpdfabsdim
4565         { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4566     }
4567     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
4568     \_unravel_set_action_text:x { \tl_to_str:N \l\_unravel_head_tl }
4569     \l\_unravel_head_tl \scan_stop:
4570     \exp_after:wN \_unravel_cond_true:n
4571     \else:
4572     \exp_after:wN \_unravel_cond_false:n
4573     \fi:
4574     {#1}
4575 }
4576 }

```

(End definition for _unravel_cond_aux:nn.)

_unravel_cond_true:n

```

4577 \cs_new_protected:Npn \_unravel_cond_true:n #1
4578 {
4579     \_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4580     \_unravel_print_action:x { \g\_unravel_action_text_str = true }
4581 }

```

(End definition for _unravel_cond_true:n.)

_unravel_cond_false:n

_unravel_cond_false_loop:n

_unravel_cond_false_common:

```

4582 \cs_new_protected:Npn \_unravel_cond_false:n #1
4583 {
4584     \_unravel_cond_false_loop:n {#1}
4585     \_unravel_cond_false_common:
4586     \_unravel_print_action:x { \g\_unravel_action_text_str = false }
4587 }
4588 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4589 {
4590     \_unravel_pass_text:
4591     \int_compare:nNnTF \g\_unravel_if_depth_int = {#1}
4592     {
4593         \token_if_eq_meaning:NNT \l\_unravel_head_token \or:
4594         {
4595             \msg_error:nn { unravel } { extra-or }
4596             \_unravel_cond_false_loop:n {#1}
4597         }
4598     }

```

```

4599     {
4600         \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4601         { \__unravel_cond_pop: }
4602         \__unravel_cond_false_loop:n {#1}
4603     }
4604 }
4605 \cs_new_protected_nopar:Npn \__unravel_cond_false_common:
4606 {
4607     \token_if_eq_meaning:NNTF \l__unravel_head_token \fi:
4608     { \__unravel_cond_pop: }
4609     { \int_gset:Nn \g__unravel_if_limit_int { 2 } } % wait for fi
4610 }

```

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

__unravel_test_two_vals:N

```

4611 \cs_new_protected:Npn \__unravel_test_two_vals:N #1
4612 {
4613     #1
4614     \__unravel_get_x_non_blank:
4615     \tl_if_in:nVF { < = > } \l__unravel_head_tl
4616     {
4617         \msg_error:nn { unravel } { missing-equals }
4618         \__unravel_back_input:
4619         \tl_set:Nn \l__unravel_head_tl { = }
4620     }
4621     \__unravel_prev_input:V \l__unravel_head_tl
4622     #1
4623 }

```

(End definition for __unravel_test_two_vals:N.)

__unravel_test_two_chars:

__unravel_test_two_chars_aux:

```

4624 \cs_new_protected_nopar:Npn \__unravel_test_two_chars:
4625 {
4626     \__unravel_test_two_chars_aux:
4627     \__unravel_prev_input:V \l__unravel_head_tl
4628     \__unravel_test_two_chars_aux:
4629     \__unravel_prev_input:V \l__unravel_head_tl
4630 }
4631 \cs_new_protected_nopar:Npn \__unravel_test_two_chars_aux:
4632 {
4633     \__unravel_get_x_next:
4634     \gtl_if_tl:NF \l__unravel_head_gtl
4635     {
4636         \tl_set:Nx \l__unravel_head_tl
4637         {
4638             \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
4639             { \c_group_begin_token } { \c_group_end_token }

```



```

4640     }
4641   }
4642   \tl_put_left:Nn \l__unravel_head_tl { \exp_not:N } % ^^A todo: prettify.
4643 }

```

(End definition for __unravel_test_two_chars: and __unravel_test_two_chars_aux:.)

```

\__unravel_test_ifx:n
\__unravel_test_ifx_aux:w
4644 \cs_new_protected:Npn \__unravel_test_ifx:n #1
4645 {
4646   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4647   \__unravel_print_action:
4648   \__unravel_get_next:
4649   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4650   \__unravel_get_next:
4651   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4652   \__unravel_set_action_text:x
4653   {
4654     Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4655     \gtl_to_str:N \l__unravel_tmpb_gtl
4656     \gtl_to_str:N \l__unravel_head_gtl
4657   }
4658   \gtl_head_do:NN \l__unravel_tmpb_gtl \__unravel_test_ifx_aux:w
4659   \exp_after:wN \__unravel_cond_true:n
4660   \else:
4661     \exp_after:wN \__unravel_cond_false:n
4662   \fi:
4663   {#1}
4664 }
4665 \cs_new_nopar:Npn \__unravel_test_ifx_aux:w
4666 { \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
4667 \cs_new_protected:Npn \__unravel_test_case:n #1
4668 {
4669   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4670   \__unravel_print_action:
4671   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
4672   \__unravel_scan_int:
4673   \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4674   \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
4675   % ^^A does text_case_aux use prev_input_seq?
4676   \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
4677   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4678   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4679 }
4680 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
4681 {

```

```

4682 \int_compare:nNnTF {#1} = \c_zero
4683 { \__unravel_change_if_limit:nn { 4 } {#2} }
4684 {
4685   \__unravel_pass_text:
4686   \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
4687   {
4688     \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
4689     {
4690       \exp_args:Nf \__unravel_test_case_aux:nn
4691       { \int_eval:n { #1 - 1 } } {#2}
4692     }
4693     { \__unravel_cond_false_common: }
4694   }
4695   {
4696     \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4697     { \__unravel_cond_pop: }
4698     \__unravel_test_case_aux:nn {#1} {#2}
4699   }
4700 }
4701 }

```

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

__unravel_test_ifdefined:

```

4702 \cs_new_protected_nopar:Npn \__unravel_test_ifdefined:
4703 {
4704   \__unravel_input_if_empty:TF
4705   { \__unravel_pass_text_empty: }
4706   {
4707     \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4708     \__unravel_set_action_text:x
4709     {
4710       Conditional:~ \tl_to_str:N \l__unravel_head_tl
4711       \gtl_to_str:N \l__unravel_tmpb_gtl
4712     }
4713     \__unravel_prev_input:x
4714     {
4715       \gtl_if_tl:NNTF \l__unravel_tmpb_gtl
4716       { \gtl_head:N \l__unravel_tmpb_gtl }
4717       { \gtl_to_str:N \l__unravel_tmpb_gtl }
4718     }
4719   }
4720 }

```

(End definition for __unravel_test_ifdefined:.)

__unravel_test_ifcsname:

```

4721 \cs_new_protected_nopar:Npn \__unravel_test_ifcsname:
4722 {
4723   \__unravel_csname_loop:

```

```

4724   \_unravel_prev_input:V \l_unravel_head_tl
4725 }

(End definition for \_unravel_test_ifcname:.)

4726 \_unravel_new_tex_expandable:nn { fi_or_else }           % 108
4727 {
4728   \int_compare:nNnTF \l_unravel_head_char_int > \g_unravel_if_limit_int
4729   {
4730     \int_compare:nNnTF \g_unravel_if_limit_int = \c_zero
4731     {
4732       \int_compare:nNnTF \g_unravel_if_depth_int = \c_zero
4733       { \msg_error:nn { unravel } { extra-fi-or-else } }
4734       { \_unravel_insert_relax: }
4735     }
4736     { \msg_error:nn { unravel } { extra-fi-or-else } }
4737   }
4738   {
4739     \_unravel_set_action_text:
4740     \int_compare:nNnF \l_unravel_head_char_int = \c_two
4741     {
4742       \_unravel-fi-or-else-loop:
4743       \_unravel_set_action_text:x
4744       {
4745         \g_unravel_action_text_str \c_space_tl
4746         => ~ skipped ~ to ~ \tl_to_str:N \l_unravel_head_tl
4747       }
4748     }
4749     % ^^A todo: in this print_action the token itself is missing.
4750     \_unravel_print_action:
4751     \_unravel_cond_pop:
4752   }
4753 }
4754 \cs_new_protected_nopar:Npn \_unravel-fi-or-else-loop:
4755 {
4756   \int_compare:nNnF \l_unravel_head_char_int = \c_two
4757   {
4758     \_unravel_pass_text:
4759     \_unravel_set_cmd:
4760     \_unravel-fi-or-else-loop:
4761   }
4762 }

```

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

```
4763 \cs_new_eq:NN \_unravel_print:n \iow_term:n  
4764 \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.

```
4765 \cs_new_protected:Npn \_unravel_print_message:nn #1 #2  
4766 { \iow_wrap:nnnN { #1 #2 } { #1 } { } \_unravel_print:n }
```

(End definition for _unravel_print_message:nn.)

_unravel_set_action_text:x

```
4767 \cs_new_protected:Npn \_unravel_set_action_text:x #1  
4768 {  
4769   \group_begin:  
4770     \_unravel_set_escapechar:n { 92 }  
4771     \str_gset:Nx \g__unravel_action_text_str {#1}  
4772   \group_end:  
4773 }
```

(End definition for _unravel_set_action_text:x.)

_unravel_set_action_text:

```
4774 \cs_new_protected_nopar:Npn \_unravel_set_action_text:  
4775 {  
4776   \_unravel_set_action_text:x  
4777   {  
4778     \tl_to_str:N \l__unravel_head_tl  
4779     \tl_if_single_token:VT \l__unravel_head_tl  
4780     { = ~ \exp_after:wN \token_to_meaning:N \l__unravel_head_tl }  
4781   }  
4782 }
```

(End definition for _unravel_set_action_text:.)

_unravel_print_state:

```
4783 \cs_new_protected:Npn \_unravel_print_state:  
4784 {  
4785   \group_begin:  
4786     \_unravel_set_escapechar:n { 92 }  
4787     \tl_use:N \g__unravel_before_print_state_tl  
4788     \int_compare:nNnT \g__unravel_noise_int > \c_zero  
4789     {  
4790       \exp_args:Nx \_unravel_print_state_output:n  
4791       { \gtl_to_str:N \g__unravel_output_gtl }  
4792       \seq_set_map:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq  
4793       { \_unravel_to_str:n {##1} }  
4794       \seq_remove_all:Nn \l__unravel_tmpa_seq { }  
4795     }  
4796   \group_end:  
4797 }
```

```

4795     \exp_args:Nx \__unravel_print_state_prev:n
4796     { \seq_use:Nn \l__unravel_tmpa_seq { \ } }
4797     \exp_args:Nx \__unravel_print_state_input:n
4798     { \__unravel_input_to_str: }
4799   }
4800   \group_end:
4801   \__unravel_prompt:
4802 }

```

(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.

```

4803 \cs_new_protected:Npn \__unravel_print_state_output:n #1
4804 {
4805   \tl_if_empty:nF {#1}
4806   {
4807     \__unravel_print_message:nn { <| ~ }
4808     { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
4809   }
4810 }

```

(End definition for __unravel_print_state_output:n.)

`__unravel_print_state_prev:n` Never trim #1.

```

4811 \cs_new_protected:Npn \__unravel_print_state_prev:n #1
4812 { \__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.

```

4813 \cs_new_protected:Npn \__unravel_print_state_input:n #1
4814 {
4815   \__unravel_print_message:nn { |> ~ }
4816   { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
4817 }

```

(End definition for __unravel_print_state_input:n.)

`__unravel_print_meaning:`

```

4818 \cs_new_protected:Npn \__unravel_print_meaning:
4819 {
4820   \__unravel_input_if_empty:TF
4821   { \__unravel_print_message:nn { } { Empty-input! } }
4822   {
4823     \__unravel_input_get:N \l__unravel_tmpb_gtl
4824     \__unravel_print_message:nn { }

```

```

4825     {
4826     \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
4827     = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
4828     }
4829   }
4830 }

```

(End definition for __unravel_print_meaning:.)

__unravel_print_action:
__unravel_print_action:x

```

4831 \cs_new_protected:Npn \__unravel_print_action:
4832 {
4833   \int_gincr:N \g__unravel_step_int
4834   \__unravel_print:x
4835   {
4836     [====
4837     \bool_if:NT \g__unravel_number_steps_bool
4838     { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
4839     =====]~
4840     \int_compare:nNnTF
4841     { \str_count:N \g__unravel_action_text_str }
4842     > { \g__unravel_max_action_int }
4843     {
4844       \str_range:Nnn \g__unravel_action_text_str
4845       { 1 } { \g__unravel_max_action_int - 3 } ...
4846     }
4847     { \g__unravel_action_text_str }
4848   }
4849   \__unravel_print_state:
4850 }
4851 \cs_new_protected:Npn \__unravel_print_action:x #1
4852 {
4853   \__unravel_set_action_text:x {#1}
4854   \__unravel_print_action:
4855 }

```

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

__unravel_print_gtl_action:N

```

4856 \cs_new_protected:Npn \__unravel_print_gtl_action:N #1
4857 {
4858   \__unravel_print_action:x { \gtl_to_str:N #1 }
4859 }

```

(End definition for __unravel_print_gtl_action:N.)

__unravel_print_done:x

```

4860 \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:
4861 \cs_new_protected_nopar:Npn \__unravel_print_assigned_token:
4862 {
4863   \__unravel_after_assignment: % ^^A todo: simplify
4864   \__unravel_print_action:x
4865   {
4866     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4867     = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
4868   }
4869   \__unravel_omit_after_assignment:w
4870 }
4871 \cs_new_protected_nopar:Npn \__unravel_print_assigned_register:
4872 {
4873   \__unravel_after_assignment: % ^^A todo: simplify
4874   \exp_args:Nx \__unravel_print_action:x
4875   {
4876     \exp_not:n
4877     {
4878       Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4879       \tl_if_single:NT \l__unravel_defined_tl
4880       { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
4881     }
4882     = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
4883   }
4884   \__unravel_omit_after_assignment:w
4885 }

```

(End definition for __unravel_print_assigned_token: and __unravel_print_assigned_register:.)

```

\__unravel_print_welcome: Welcome message.
4886 \cs_new_protected_nopar:Npn \__unravel_print_welcome:
4887 {
4888   \__unravel_print_message:nn { }
4889   {
4890     \bool_if:NTF \g__unravel_welcome_message_bool
4891     {
4892       \\\
4893       =====~ Welcome~ to~ the~ unravel~ package~ =====\\
4894       \iow_indent:n
4895       {
4896         "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\
4897         "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\\
4898         "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\\
4899         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\\
4900       }
4901     }
4902     { [=====Start=====] }
4903   }
4904   \__unravel_print_state:
4905 }

```

(End definition for _unravel_print_welcome:.)

_unravel_print_outcome: Final message.

```
4906 \cs_new_protected_nopar:Npn \_unravel_print_outcome:
4907   { \_unravel_print:n { [====~End~====] } }
```

(End definition for _unravel_print_outcome:.)

2.15.2 Prompt

_unravel_prompt:

```
4908 \cs_new_protected_nopar:Npn \_unravel_prompt:
4909   {
4910     \int_gdecr:N \g__unravel_nonstop_int
4911     \int_compare:nNnF \g__unravel_nonstop_int > \c_zero
4912       {
4913         \group_begin:
4914           \_unravel_set_escapechar:n { -1 }
4915           \int_set_eq:NN \tex_endlinechar:D \c_minus_one
4916           \tl_use:N \g__unravel_before_prompt_tl
4917           \_unravel_prompt_aux:
4918         \group_end:
4919       }
4920   }
4921 \cs_new_protected_nopar:Npn \_unravel_prompt_aux:
4922   {
4923     \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
4924     {
4925       \bool_if:NTF \g__unravel_explicit_prompt_bool
4926         { \ior_get_str:Nc \c__unravel_prompt_ior }
4927         { \ior_get_str:Nc \c__unravel_noprompt_ior }
4928         { Your~input }
4929       \exp_args:Nv \_unravel_prompt_treat:n { Your~input }
4930     }
4931   }
4932 \cs_new_protected:Npn \_unravel_prompt_treat:n #1
4933   {
4934     \tl_if_empty:nF {#1}
4935     {
4936       \exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
4937       {
4938         { m } { \_unravel_print_meaning: \_unravel_prompt_aux: }
4939         { q }
4940         {
4941           \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4942           \int_gzero:N \g__unravel_nonstop_int
4943         }
4944         { x }
4945         {
4946           \group_end:
```



```

4947         \exp_after:wN \__unravel_exit:w \__unravel_exit:w
4948     }
4949     { X } { \tex_batchmode:D \tex_end:D }
4950     { s } { \__unravel_prompt_scan_int:nn {#1}
4951         \__unravel_prompt_silent_steps:n }
4952     { o } { \__unravel_prompt_scan_int:nn {#1}
4953         { \int_gset:Nn \g__unravel_noise_int } }
4954     { C }
4955     {
4956         \tl_gset_rescan:Nnx \g__unravel_tmpc_tl
4957         { \ExplSyntaxOn } { \tl_tail:n {#1} }
4958         \tl_gput_left:Nn \g__unravel_tmpc_tl
4959         { \tl_gclear:N \g__unravel_tmpc_tl }
4960         \group_insert_after:N \g__unravel_tmpc_tl
4961     }
4962     { | } { \__unravel_prompt_scan_int:nn {#1}
4963         \__unravel_prompt_vert:n }
4964     { a } { \__unravel_prompt_all: }
4965     }
4966     { \__unravel_prompt_help: }
4967 }
4968 }
4969 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
4970 {
4971     \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
4972     \l__unravel_prompt_tmpa_int =
4973     \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
4974     \use_ii:nn #1 \scan_stop:
4975 }
4976 \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
4977 {
4978     #2 \l__unravel_prompt_tmpa_int
4979     \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
4980 }
4981 \cs_new_protected:Npn \__unravel_prompt_help:
4982 {
4983     \__unravel_print:n { "m":~meaning-of~first-token }
4984     \__unravel_print:n { "q":~semi-quiet~(same-as~"o1") }
4985     \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
4986     \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
4987     \__unravel_print:n
4988     { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
4989     \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
4990     \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"|" }
4991     \__unravel_print:n { "a":~print~state~again,~without~truncating }
4992     \__unravel_prompt_aux:
4993 }
4994 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
4995 {
4996     \int_compare:nNnF {#1} < \c_zero

```

```

4997 {
4998   \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4999   \tl_gset:Nn \g__unravel_before_prompt_tl
5000   {
5001     \int_gset_eq:NN \g__unravel_noise_int \c_one
5002     \tl_gclear:N \g__unravel_before_prompt_tl
5003   }
5004   \int_gset:Nn \g__unravel_nonstop_int {#1}
5005 }
5006 }
5007 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5008 {
5009   \int_compare:nNnTF {#1} < { 0 }
5010   { \__unravel_prompt_vert:Nn > {#1} }
5011   { \__unravel_prompt_vert:Nn < {#1} }
5012 }
5013 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5014 {
5015   \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
5016   \tl_gset:Nf \g__unravel_before_print_state_tl
5017   {
5018     \exp_args:Nnf \exp_stop_f: \int_compare:nNnTF
5019     { \int_eval:n { \seq_count:N \g__unravel_prev_input_seq - #2 } }
5020     #1 { \seq_count:N \g__unravel_prev_input_seq }
5021     {
5022       \int_gset:Nn \g__unravel_nonstop_int
5023       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5024     }
5025     {
5026       \int_gset_eq:NN \g__unravel_noise_int \c_one
5027       \tl_gclear:N \g__unravel_before_print_state_tl
5028     }
5029   }
5030 }
5031 \cs_new_protected_nopar:Npn \__unravel_prompt_all:
5032 {
5033   \tl_gset:Nx \g__unravel_tmpc_tl
5034   {
5035     \exp_not:n
5036     {
5037       \tl_gclear:N \g__unravel_tmpc_tl
5038       \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5039       \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5040       \__unravel_print_state:
5041     }
5042     \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5043     \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5044   }
5045   \group_insert_after:N \g__unravel_tmpc_tl
5046 }

```

```

5047 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5048   { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }
(End definition for \__unravel_prompt:.)

```

2.15.3 Errors

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

```

5049 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5050   {
5051     \cs_new_nopar:cpn { __unravel_tex_msg_error_#1: } {#2}
5052     \cs_new_nopar:cpn { __unravel_tex_msg_help_#1: } {#3}
5053   }
(End definition for \__unravel_tex_msg_new:nnn.)

```

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

`__unravel_tex_error:nV`

```

5054 \cs_new_protected:Npn \__unravel_tex_error:nn #1#2
5055   {
5056     \msg_error:nnxxx { unravel } { tex-error }
5057     { \tl_to_str:n {#2} }
5058     { \use:c { __unravel_tex_msg_error_#1: } }
5059     { \use:c { __unravel_tex_msg_help_#1: } }
5060   }
5061 \cs_generate_variant:Nn \__unravel_tex_error:nn { nV }
(End definition for \__unravel_tex_error:nn and \__unravel_tex_error:nV.)

```

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

`__unravel_tex_fatal_error:nV`

```

5062 \cs_new_protected:Npn \__unravel_tex_fatal_error:nn #1#2
5063   {
5064     \msg_error:nnxxx { unravel } { tex-fatal }
5065     { \tl_to_str:n {#2} }
5066     { \use:c { __unravel_tex_msg_error_#1: } }
5067     { \use:c { __unravel_tex_msg_help_#1: } }
5068   }
5069 \cs_generate_variant:Nn \__unravel_tex_fatal_error:nn { nV }
(End definition for \__unravel_tex_fatal_error:nn and \__unravel_tex_fatal_error:nV.)

```

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.

```

5070 \keys_define:nn { unravel/defaults }
5071   {

```

```

5072     explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5073     internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5074     max-action .int_gset:N = \g__unravel_default_max_action_int ,
5075     max-output .int_gset:N = \g__unravel_default_max_output_int ,
5076     max-input .int_gset:N = \g__unravel_default_max_input_int ,
5077     number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
5078     welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5079   }
5080 \keys_define:nn { unravel }
5081 {
5082     explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5083     internal-debug .bool_gset:N = \g__unravel_internal_debug_bool ,
5084     max-action .int_gset:N = \g__unravel_max_action_int ,
5085     max-output .int_gset:N = \g__unravel_max_output_int ,
5086     max-input .int_gset:N = \g__unravel_max_input_int ,
5087     number-steps .bool_gset:N = \g__unravel_number_steps_bool ,
5088     welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,
5089 }

```

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.

```

5090 \tl_map_inline:nn { { /defaults } { } }
5091 {
5092   \keys_define:nn { unravel #1 }
5093   {
5094     machine .meta:nn =
5095     { unravel #1 }
5096     {
5097       explicit-prompt = false ,
5098       internal-debug = false ,
5099       max-action = \c_max_int ,
5100       max-output = \c_max_int ,
5101       max-input = \c_max_int ,
5102       number-steps = false ,
5103       welcome-message = false ,
5104     } ,
5105   }
5106 }

```

2.17 Main command

\unravel Simply call an underlying code-level command.

```
5107 \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.

```
5108 \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.

```
5109 \cs_new_protected:Npn \unravel_setup:n #1
5110   {
5111     \keys_set:nn { unravel/defaults } {#1}
5112     \keys_set:nn { unravel } {#1}
5113   }
```

(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.

```
5114 \cs_new_protected:Npn \unravel:nn #1#2
5115   {
5116     \__unravel_init_key_vars:
5117     \keys_set:nn { unravel } {#1}
5118     \__unravel_init_vars:
5119     \__unravel_input_gset:n {#2}
5120     \__unravel_print_welcome:
5121     \__unravel_main_loop:
5122     \__unravel_exit_point:
5123     \__unravel_print_outcome:
5124     \__unravel_final_test:
5125     \__unravel_exit_point:
5126   }
```

(End definition for `\unravel:nn`.)

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

```
5127 \cs_new_protected_nopar:Npn \__unravel_init_key_vars:
5128   {
5129     \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bool
5130     \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5131     \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5132     \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bool
5133     \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5134     \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5135     \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
5136   }
```

(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.

```
5137 \cs_new_protected_nopar:Npn \__unravel_init_vars:
5138 {
5139   \seq_gclear:N \g__unravel_prev_input_seq
5140   \gtl_gclear:N \g__unravel_output_gtl
5141   \int_gzero:N \g__unravel_step_int
5142   \tl_gclear:N \g__unravel_if_limit_tl
5143   \int_gzero:N \g__unravel_if_limit_int
5144   \int_gzero:N \g__unravel_if_depth_int
5145   \gtl_gclear:N \g__unravel_after_assignment_gtl
5146   \bool_gset_true:N \g__unravel_set_box_allowed_bool
5147   \bool_gset_false:N \g__unravel_name_in_progress_bool
5148   \gtl_clear:N \l__unravel_after_group_gtl
5149 }
```

(End definition for __unravel_init_vars:.)

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

```
5150 \cs_new_protected_nopar:Npn \__unravel_main_loop:
5151 {
5152   \__unravel_get_x_next:
5153   \__unravel_set_cmd:
5154   \__unravel_do_step:
5155   \__unravel_main_loop:
5156 }
```

(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:
5157 \cs_new_protected_nopar:Npn \__unravel_final_test:
5158 {
5159   \bool_if:nTF
5160     {
5161       \tl_if_empty_p:N \g__unravel_if_limit_tl
5162       && \int_compare_p:nNn \g__unravel_if_limit_int = \c_zero
5163       && \int_compare_p:nNn \g__unravel_if_depth_int = \c_zero
5164       && \seq_if_empty_p:N \g__unravel_prev_input_seq
5165     }
5166     { \__unravel_input_if_empty:TF { } { \__unravel_final_bad: } }
5167     { \__unravel_final_bad: }
5168 }
5169 \cs_new_protected_nopar:Npn \__unravel_final_bad:
5170 {
5171   \msg_error:nxx { unravel } { internal }
5172   { the-last-unravel-finished-badly }
5173 }
```

(End definition for __unravel_final_test: and __unravel_final_bad:.)

2.18 Messages

```
5174 \msg_new:nnn { unravel } { unknown-primitive }
5175   { Internal~error:-the-primitive-~'#1'~is-not-known. }
5176 \msg_new:nnn { unravel } { extra-fi-or-else }
5177   { Extra-fi,~or,~or~else. }
5178 \msg_new:nnn { unravel } { missing-lbrace }
5179   { Missing~left~brace~inserted. }
5180 \msg_new:nnn { unravel } { missing-dollar }
5181   { Missing~dollar~inserted. }
5182 \msg_new:nnn { unravel } { unknown-expandable }
5183   { Internal~error:-the-expandable-command-~'#1'~is-not-known. }
5184 \msg_new:nnn { unravel } { missing-font-id }
5185   { Missing~font~identifier.-~\iow_char:N\~nullfont~inserted. }
5186 \msg_new:nnn { unravel } { missing-rparen }
5187   { Missing~right~parenthesis~inserted~for~expression. }
5188 \msg_new:nnn { unravel } { missing-mu-dim }
5189   { Missing~mu~unit. }
5190 \msg_new:nnn { unravel } { missing-cs }
5191   { Missing~control~sequence.-~\iow_char:N\~inaccessible~inserted. }
5192 \msg_new:nnn { unravel } { missing-box }
5193   { Missing~box~inserted. }
5194 \msg_new:nnn { unravel } { missing-to }
5195   { Missing~keyword-~'to'~inserted. }
5196 \msg_new:nnn { unravel } { improper-leaders }
5197   { Leaders~not~followed~by~proper~glue. }
5198 \msg_new:nnn { unravel } { extra-close }
5199   { Extra~right~brace~or-~\iow_char:N\~endgroup. }
5200 \msg_new:nnn { unravel } { off-save }
5201   { Something~is~wrong~with~groups. }
5202 \msg_new:nnn { unravel } { hrule-bad-mode }
5203   { \iow_char\~hrule-used-in-wrong-mode. }
5204 \msg_new:nnn { unravel } { invalid-mode }
5205   { Invalid~mode~for~this~command. }
5206 \msg_new:nnn { unravel } { color-stack-action-missing }
5207   { Missing~color~stack~action. }
5208 \msg_new:nnn { unravel } { action-type-missing }
5209   { Missing~action~type. }
5210 \msg_new:nnn { unravel } { identifier-type-missing }
5211   { Missing~identifier~type. }
5212 \msg_new:nnn { unravel } { destination-type-missing }
5213   { Missing~destination~type. }
5214 \msg_new:nnn { unravel } { erroneous-prefixes }
5215   { Prefixes~applied~to~non-assignment~command. }
5216 \msg_new:nnn { unravel } { improper-setbox }
5217   { \iow_char:N\~setbox~while~fetching~base~of~an~accent. }
5218 \msg_new:nnn { unravel } { after-advance }
5219   {
5220     Missing~register~after~\iow_char:N\~advance,~
5221     \iow_char:N\~multiply,~or~\iow_char:N\~divide.
```

```

5222 }
5223 \msg_new:nnn { unravel } { bad-unless }
5224 { \iow_char:N\unless-not-followed-by-conditional. }
5225 \msg_new:nnn { unravel } { missing-endsname }
5226 { Missing~\iow_char:N\endsname-inserted. }
5227 \msg_new:nnn { unravel } { runaway-if }
5228 { Runaway~\iow_char:N\if... }
5229 \msg_new:nnn { unravel } { runaway-macro-parameter }
5230 {
5231   Runaway~macro-parameter~\# #2-after \\\
5232   \iow_indent:n {#1}
5233 }
5234 \msg_new:nnn { unravel } { extra-or }
5235 { Extra~\iow_char:N\or. }
5236 \msg_new:nnn { unravel } { missing-equals }
5237 { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
5238 \msg_new:nnn { unravel } { internal }
5239 { Internal~error:~'#1'.~ Please~report. }
5240 \msg_new:nnn { unravel } { not-implemented }
5241 { The~following~feature~is~not~implemented:~'#1'. }
5242 \msg_new:nnn { unravel } { endinput-ignored }
5243 { The~primitive~\iow_char:N\endinput~was~ignored. }
5244 \msg_new:nnn { unravel } { missing-something }
5245 { Something~is~missing,~sorry! }
5246 \msg_new:nnnn { unravel } { tex-error }
5247 { TeX~sees~"#1"~and~throws~an~error:\\\ \iow_indent:n {#2} }
5248 {
5249   \tl_if_empty:nTF {#3}
5250     { TeX~provides~no~further~help~for~this~error. }
5251     { TeX's~advice~is:\\\ \iow_indent:n {#3} }
5252 }
5253 \msg_new:nnnn { unravel } { tex-fatal }
5254 { TeX~sees~"#1"~and~throws~a~fatal~error:\\\ \iow_indent:n {#2} }
5255 {
5256   \tl_if_empty:nTF {#3}
5257     { TeX~provides~no~further~help~for~this~error. }
5258     { TeX's~advice~is:\\\ \iow_indent:n {#3} }
5259 }

Some error messages from TeX itself.
5260 \__unravel_tex_msg_new:nnn { incompatible-mag }
5261 {
5262   Incompatible~magnification~
5263   ( \int_to_arabic:n { \__unravel_mag: } ); \
5264   \ the~previous~value~will~be~retained
5265 }
5266 {
5267   I~can~handle~only~one~magnification~ratio~per~job.~So~I've\\\
5268   reverted~to~the~magnification~you~used~earlier~on~this~run.
5269 }

```



```

5270 \__unravel_tex_msg_new:nnn { illegal-mag }
5271   {
5272     Illegal~magnification~has~been~changed~to~1000~
5273     ( \int_to_arabic:n { \__unravel_mag: } )
5274   }
5275   { The~magnification~ratio~must~be~between~1~and~32768. }
5276 \__unravel_tex_msg_new:nnn { incompatible-units }
5277   { Incompatible~glue~units }
5278   { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }

Fatal TEX error messages.
5279 \__unravel_tex_msg_new:nnn { cannot-read }
5280   { ***~(cannot~\iow_char:N\read-from-terminal-in-nonstop-modes) }
5281   { }
5282 \__unravel_tex_msg_new:nnn { file-error }
5283   { ***~(job~aborted,~file~error~in~nonstop-mode) }
5284   { }
5285 \__unravel_tex_msg_new:nnn { interwoven-preambles }
5286   { (interwoven~alignment~preambles~are~not~allowed) }
5287   { }

Restore catcodes to their original values.
5288 \__unravel_setup_restore:
5289 </package>

```