

# The `unravel` package: watching TeX digest tokens\*

Bruno Le Floch

2018/04/04

## Contents

<b>1</b>	<b><code>unravel</code> documentation</b>	<b>2</b>
1.1	Options	4
1.2	Differences between <code>unravel</code> and TeX's processing	4
1.3	Future perhaps	5
<b>2</b>	<b><code>unravel</code> implementation</b>	<b>5</b>
2.1	Primitives, variants, and helpers	9
2.1.1	Renamed primitives	9
2.1.2	Variants	10
2.1.3	Miscellaneous helpers	10
2.1.4	String helpers	12
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	18
2.2.1	User interaction	18
2.2.2	Working with tokens	19
2.2.3	Numbers and conditionals	21
2.2.4	Boxes and groups	22
2.2.5	Constants	22
2.2.6	TeX parameters	22
2.3	Numeric codes	23
2.4	Get next token	36
2.5	Manipulating the input	40
2.5.1	Elementary operations	40
2.5.2	Insert token for error recovery	45
2.5.3	Macro calls	46
2.6	Expand next token	47
2.7	Basic scanning subroutines	49
2.8	Working with boxes	67
2.9	Paragraphs	71
2.10	Groups	73

---

\*This file has version number 0.2e, last revised 2018/04/04.

2.11 Modes . . . . .	75
2.12 One step . . . . .	77
2.13 Commands . . . . .	77
2.13.1 Characters: from 0 to 15 . . . . .	77
2.13.2 Boxes: from 16 to 31 . . . . .	81
2.13.3 From 32 to 47 . . . . .	86
2.13.4 Maths: from 48 to 56 . . . . .	90
2.13.5 From 57 to 70 . . . . .	90
2.13.6 Extensions . . . . .	93
2.13.7 Assignments . . . . .	100
2.14 Expandable primitives . . . . .	110
2.14.1 Conditionals . . . . .	115
2.15 User interaction . . . . .	123
2.15.1 Print . . . . .	123
2.15.2 Prompt . . . . .	127
2.15.3 Errors . . . . .	130
2.16 Keys . . . . .	131
2.17 Main command . . . . .	132
2.18 Messages . . . . .	134

## 1 unravel documentation

The aim of this L<sup>A</sup>T<sub>E</sub>X package is to help debug complicated macros. This is done by letting the user step through the execution of some T<sub>E</sub>X 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<sub>E</sub>X in a terminal.

---

<code>\unravel</code>	<code>\unravel [<i>&lt;key-value list&gt;</i>] {<i>&lt;code&gt;</i>}</code>
-----------------------	---

---

This command shows in the terminal the steps performed by T<sub>E</sub>X 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.

---

<code>\unravelsetup</code>	<code>\unravelsetup {<i>&lt;options&gt;</i>}</code>
----------------------------	---

---

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

---

<code>\unravel:nn</code>	<code>\unravel:nn {<i>&lt;options&gt;</i>} {<i>&lt;code&gt;</i>}</code>
--------------------------	---

---

See `\unravel`.

---

<code>\unravel_setup:n</code>	<code>\unravel_setup:n {<i>&lt;options&gt;</i>}</code>
-------------------------------	--

---

See `\unravelsetup`.

The `unravel` package is currently based on the behaviour of pdfT<sub>E</sub>X, but it should work in all engines supported by `expl3` (pdfT<sub>E</sub>X, Xe<sub>L</sub>T<sub>E</sub>X, LuaT<sub>E</sub>X, epT<sub>E</sub>X, eupT<sub>E</sub>X) as long as none of the primitives specific to those engines is used. Any difference between how `unravel` and (pdf)T<sub>E</sub>X process a given piece of code, unless described in the

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

As a simple example, one can run L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X, it is not fast on very large pieces of code. For instance, running it on `\documentclass{article}` takes about one minute on my machine, and finishes after somewhat less than 21000 steps.

```
\RequirePackage{unravel}
\unravel{\documentclass{article}\relax}
\usepackage{lipsum}
\begin{document}
\lipsum
\end{document}
```

The `\relax` command is needed after `\documentclass{article}` because this command tries to look for an optional argument: `\unravel` would not find any token, and would give up, as `TEX` would if your file ended just after `\documentclass{article}`. After running the above through pdf`TEX`, one can check that the result is identical to that without `unravel`. Note that `\unravel{\usepackage{lipsum}\relax}`, despite taking as many steps to complete, is much slower, because `\newcommand` uses delimited arguments, which prevent some optimizations that `unravel` can otherwise obtain. For comparison, `\unravel{\lipsum[1-30]}` also takes 20000 step.

## 1.1 Options

<u>explicit-prompt</u>	Boolean option (default <b>false</b> ) determining whether to give an explicit prompt. If <b>true</b> , the text “Your input=” will appear at the beginning of lines where user input is expected.
<u>internal-debug</u>	Boolean option (default <b>false</b> ) 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 <b>false</b> .
<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 <b>true</b> ) determining whether to number steps.
<u>welcome-message</u>	Boolean option (default <b>true</b> ) 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 `\currentifttype` may have wrong values. Perhaps `\currentgrouplevel` and `\currentgrouptype` too.
- Tokens passed to `\afterassignment` are not yet kept after `unravel` is done even if there has been no assignment.
- Tokens passed to `\aftergroup` are lost when `unravel` is done.

- In  $\text{X}\text{\TeX}$ , 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  $\text{T}\text{\TeX}$  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  $\text{T}\text{\TeX}$ '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  $\text{T}\text{\TeX}$ 's procedure `prepare_mag` is called.
- Find out why so many input levels are used (see the log of the `unravel1003` 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 `TEX` 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\newlinechar-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  $\varepsilon$ -`TEX`'s `\numexpr` or `\protected` are not available, bail out with an error.

```

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

```

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

```

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

```

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

```

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

```

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

```

20 \gdef \l__unravel_setup_restore_tl { }

```

(End definition for `\l__unravel_setup_restore_tl`.)

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

```

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

```

(End definition for `\__unravel_setup_restore:`.)

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

```

26 \protected \gdef \__unravel_setup_save:
27 {
28   \edef \l__unravel_setup_restore_tl
29   {
30     \__unravel_setup_save_aux:w 0 =
31     \newlinechar = \the \newlinechar
32     \escapechar = \the \escapechar

```

```

33         \newlinechar = \the \newlinechar
34         \relax
35     }
36 }
37 \long \gdef \__unravel_setup_save_aux:w #1 =
38 {
39     \catcode #1 = \the \catcode #1 ~
40     \ifnum 255 > #1 ~
41         \expandafter \__unravel_setup_save_aux:w
42         \the \numexpr #1 + 1 \expandafter =
43     \fi
44 }

```

(End definition for \\_\_unravel\_setup\_save: and \\_\_unravel\_setup\_save\_aux:n.)

\\_\_unravel\_setup\_catcodes:nnn This sets all characters from #1 to #2 (inclusive) to have catcode #3.

```

45 \protected \long \gdef \__unravel_setup_catcodes:nnn #1 #2 #3
46 {
47     \ifnum #1 > #2 ~ \else
48         \catcode #1 = #3 ~
49         \expandafter \__unravel_setup_catcodes:nnn \expandafter
50         { \the \numexpr #1 + 1 } {#2} {#3}
51     \fi
52 }

```

(End definition for \\_\_unravel\_setup\_catcodes:nnn.)

\\_\_unravel\_setup\_latex: This saves the catcodes and related parameters, then sets them to the value they normally have in a L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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`, `glt`. Load `l3str` if `expl3` is too old and does not define `\str_range:nnn`. Otherwise loading `l3str` would give an error.

```
128 \RequirePackage{expl3,xparse}[2018/02/21]
```

```
129 \RequirePackage{glt}[2018/04/04]
```

```
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} {2018/04/04} {0.2e} {Watching TeX digest tokens}%
```

```
136   \csname \_unravel_setup_unravel:\endcsname
```

```
137 }%
```

## 2.1 Primitives, variants, and helpers

### 2.1.1 Renamed primitives

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

```
138 \cs_new_eq:NN \_unravel_currentgrouptype: \etex_currentgrouptype:D
139 \cs_new_protected_nopar:Npn \_unravel_set_escapechar:n
140   { \int_set:Nn \tex_escapechar:D }
141 \cs_new_eq:NN \_unravel_everyeof:w \etex_everyeof:D
142 \cs_new_eq:NN \_unravel_everypar:D \tex_everypar:D
143 \cs_new_eq:NN \_unravel_hbox:w \tex_hbox:D
144 \cs_new_eq:NN \_unravel_mag: \tex_mag:D
145 \cs_new_eq:NN \_unravel_nullfont: \tex_nullfont:D
146 \cs_new_eq:NN \_unravel_the:w \tex_the:D
147 \cs_new_eq:NN \_unravel_number:w \tex_number:D
```

(End definition for `\_unravel_currentgrouptype:` and others.)

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

```
148 \cs_new_eq:NN \c__unravel_prompt_ior \c_sixteen
```

```
149 \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.

```

150 \cs_generate_variant:Nn \seq_push:Nn { Nf }
151 \cs_generate_variant:Nn \str_head:n { f }
152 \cs_generate_variant:Nn \tl_to_str:n { o }
153 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
154 \cs_generate_variant:Nn \tl_if_in:nnF { nV }
155 \cs_generate_variant:Nn \tl_if_in:nnTF { nV }
156 \cs_generate_variant:Nn \tl_if_in:NnTF { No , NV }
157 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
158 \cs_generate_variant:Nn \gtl_gput_right:Nn { NV }
159 \cs_generate_variant:Nn \ior_str_get:NN { Nc }
160 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
161 \cs_generate_variant:Nn \gtl_to_str:N { c }
162 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
163 \cs_generate_variant:Nn \gtl_get_left:NN { c }
164 \cs_generate_variant:Nn \gtl_gset:Nn { c }
165 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
166 \cs_generate_variant:Nn \gtl_gclear:N { c }
167 \cs_generate_variant:Nn \gtl_gclear_new:N { c }

```

```

\l__unravel_exp_tl
\__unravel_exp_args:Nx
\__unravel_exp_args:NNx

```

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

```

168 \tl_new:N \l__unravel_exp_tl
169 \cs_new_protected:Npn \__unravel_exp_args:Nx #1#2
170 {
171   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 {#2} }
172   \l__unravel_exp_tl
173 }
174 \cs_new_protected:Npn \__unravel_exp_args:NNx #1#2#3
175 {
176   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 \exp_not:N #2 {#3} }
177   \l__unravel_exp_tl
178 }

```

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

## 2.1.3 Miscellaneous helpers

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

```

179 \cs_new_protected_nopar:Npn \__unravel_tmp:w { }

```

(End definition for `\__unravel_tmp:w`.)

```

\__unravel_file_get:nN
\__unravel_file_get_aux:wN

```

```

180 \cs_set_protected:Npn \__unravel_tmp:w #1
181 {
182   \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
183   {
184     \group_begin:
185     \__unravel_everyeof:w { #1 ##2 }
186     \exp_after:wN \__unravel_file_get_aux:wN
187     \exp_after:wN \prg_do_nothing:

```

```

188         \tex_input:D ##1 \scan_stop:
189     }
190     \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
191     {
192         \group_end:
193         \tl_set:Nx ##2
194         { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
195     }
196 }
197 \exp_args:No \__unravel_tmp:w { \token_to_str:N : : }

```

(End definition for \\_\_unravel\_file\_get:nN and \\_\_unravel\_file\_get\_aux:wN.)

\\_\_unravel\_tl\_first\_int:N  
 \\_\_unravel\_tl\_first\_int\_aux:Nn

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

```

198 \cs_new:Npn \__unravel_tl_first_int:N #1
199 {
200     \int_eval:n
201     {
202         \exp_after:wN \__unravel_tl_first_int_aux:Nn
203         \exp_after:wN \__unravel_tl_first_int_aux:Nn
204         #1 ? 0 ? \q_stop
205     }
206 }
207 \cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
208 {
209     \tl_if_single:nT {#2}
210     {
211         \token_if_eq_catcode:NNT + #2
212         {
213             \if_int_compare:w 1 < 1 #2 \exp_stop_f:
214             #2
215             \exp_after:wN \use_i_ii:nnn
216             \exp_after:wN \__unravel_tl_first_int_aux:Nn
217             \exp_after:wN \use_none_delimit_by_q_stop:w
218             \fi:
219         }
220     }
221     #1
222 }

```

(End definition for \\_\_unravel\_tl\_first\_int:N and \\_\_unravel\_tl\_first\_int\_aux:Nn.)

\\_\_unravel\_prepare\_mag: Used whenever T<sub>E</sub>X needs the value of \mag.

```

223 \cs_new_protected_nopar:Npn \__unravel_prepare_mag:
224 {
225     \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
226     {

```

```

227 \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
228 {
229   \__unravel_tex_error:nn { incompatible-mag } { }
230   \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
231 }
232 }
233 \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
234 {
235   \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
236   \int_gset:Nn \__unravel_mag: { 1000 }
237 }
238 \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
239 }

```

(End definition for \\_\_unravel\_prepare\_mag:.)

### 2.1.4 String helpers

\\_\_unravel\_strip\_escape:w This is based on the 2013-07-19 (and earlier) version of \cs\_to\_str:N. There are three cases. If the escape character is printable, the charcode test is false, and \\_\_unravel\_strip\_escape\_aux:N removes one character. If the escape character is a space, the charcode test is true, and if there is no escape character, the test is unfinished after \token\_to\_str:N \ . In both of those cases, \\_\_unravel\_strip\_escape\_aux:w inserts -\@@\_number:w \fi: \c\_zero. 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.

```

240 \cs_new_nopar:Npn \__unravel_strip_escape:w
241 {
242   \tex_romannumeral:D
243   \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
244   \__unravel_strip_escape_aux:N
245 }
246 \cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero }
247 \cs_new:Npn \__unravel_strip_escape_aux:w #1#2
248 { - \__unravel_number:w #1 \c_zero }

```

(End definition for \\_\_unravel\_strip\_escape:w, \\_\_unravel\_strip\_escape\_aux:N, and \\_\_unravel\_strip\_escape\_aux:w.)

\\_\_unravel\_to\_str:n Use the type-appropriate conversion to string.

```

\__unravel_to_str_auxi:w
\__unravel_to_str_auxii:w
\__unravel_gtl_to_str:n
249 \cs_new:Npn \__unravel_to_str:n #1
250 {
251   \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
252   { \__unravel_to_str_auxi:w #1 ? \q_stop }
253   { \tl_to_str:n }
254   {#1}
255 }
256 \cs_set:Npn \__unravel_tmp:w #1
257 {
258   \cs_new:Npn \__unravel_to_str_auxi:w ##1##2 \q_stop
259   {
260     \exp_after:wN \__unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark

```

```

261         #1 t1 \q_mark \q_stop
262     }
263     \cs_new:Npn \__unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
264     { \cs_if_exist_use:cF { __unravel_ ##2 _to_str:n } { \tl_to_str:n } }
265 }
266 \exp_args:No \__unravel_tmp:w { \tl_to_str:n { s _ _ } }
267 \cs_new:Npn \__unravel_gtl_to_str:n { \gtl_to_str:n }

```

(End definition for \\_\_unravel\_to\_str:n and others.)

\\_\_unravel\_str\_truncate\_left:nn Truncate the string #1 to a maximum of #2 characters. If it is longer, replace some characters on the left of the string by (123~more~chars)~ with the appropriate number instead of 123. In any reasonable case, 25 is big enough to fit this extra text.

\\_\_unravel\_str\_truncate\_left\_aux:nnn

```

268 \cs_new:Npn \__unravel_str_truncate_left:nn #1#2
269 {
270     \exp_args:Nf \__unravel_str_truncate_left_aux:nnn
271     { \str_count:n {#1} } {#1} {#2}
272 }
273 \cs_new:Npn \__unravel_str_truncate_left_aux:nnn #1#2#3
274 {
275     \int_compare:nNnTF {#1} > {#3}
276     {
277         ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
278         \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
279     }
280     { \tl_to_str:n {#2} }
281 }

```

(End definition for \\_\_unravel\_str\_truncate\_left:nn and \\_\_unravel\_str\_truncate\_left\_aux:nnn.)

\\_\_unravel\_str\_truncate\_right:nn Truncate the string #1 to a maximum of #2 characters. If it is longer, replace some characters on the right of the string by ~(123~more~chars) with the appropriate number instead of 123. In any reasonable case, 25 is big enough to fit this extra text.

\\_\_unravel\_str\_truncate\_right\_aux:nnn

```

282 \cs_new:Npn \__unravel_str_truncate_right:nn #1#2
283 {
284     \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
285     { \str_count:n {#1} } {#1} {#2}
286 }
287 \cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
288 {
289     \int_compare:nNnTF {#1} > {#3}
290     {
291         \str_range:nnn {#2} { 1 } { #3 - 25 } ~
292         ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
293     }
294     { \tl_to_str:n {#2} }
295 }

```

(End definition for \\_\_unravel\_str\_truncate\_right:nn and \\_\_unravel\_str\_truncate\_right\_aux:nnn.)

## 2.1.5 Helpers for control flow

\\_\_unravel\_exit:w Jump to the very end of this instance of \unravel.

\\_\_unravel\_exit\_point:

```

296 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
297 \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: 298 \cs_new_eq:NN \_unravel_break_point: \prg_do_nothing:
299 \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).

```

300 \prg_new_conditional:Npnn \_unravel_cmd_if_internal: { TF }
301 {
302   \int_compare:nNnTF
303     \l\_unravel_head_cmd_int < { \_unravel_tex_use:n { min_internal } }
304     { \prg_return_false: }
305     {
306       \int_compare:nNnTF
307         \l\_unravel_head_cmd_int
308         > { \_unravel_tex_use:n { max_internal } }
309         { \prg_return_false: }
310         { \prg_return_true: }
311     }
312 }

```

(End definition for `\_unravel_cmd_if_internal:TF`.)

## 2.1.6 Helpers concerning tokens

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

```

\_unravel_meaning_to_char:n 313 \cs_new:Npn \_unravel_meaning_to_char:n #1
\_unravel_meaning_to_char:o 314 { \_unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
\_unravel_meaning_to_char:o 315 \cs_new:Npn \_unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
\_unravel_meaning_to_char:o 316 { \_unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
317 \cs_new:Npn \_unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
318 { \tl_if_empty:nTF {#2} { ~ } {#2} }
319 \cs_generate_variant:Nn \_unravel_meaning_to_char:n { o }
320 \cs_new:Npn \_unravel_token_to_char:N #1
321 { \_unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

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

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

```

\_unravel_token_if_expandable:N 322 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
323 { p , T , F , TF }
324 {
325   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
326   \prg_return_false:
327   \else:
328     \prg_return_true:
329   \fi:
330 }

```

(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.

```

331 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
332 { p , T , F , TF }
333 {
334   \_unravel_token_if_expandable:NTF #1
335   {
336     \token_if_protected_macro:NTF #1
337     { \prg_return_true: }
338     {
339       \token_if_protected_long_macro:NTF #1
340       { \prg_return_true: }
341       { \prg_return_false: }
342     }
343   }
344   { \prg_return_true: }
345 }

```

(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<sub>Y</sub>TeX (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.

```

346 \group_begin:
347   \char_set_catcode_active:n { 'Z }
348   \prg_new_protected_conditional:Npnn \_unravel_token_if_definable:N #1
349   { TF }
350   {
351     \group_begin:
352       \_unravel_set_escapechar:n { 92 }
353       \tl_set:Nx \l__unravel_tmpa_tl
354       { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
355       \tl_if_empty:NTF \l__unravel_tmpa_tl
356       {
357         \exp_args:Nx \char_set_lccode:nn
358         { ' \str_head:n {#1} } { 'Z }
359         \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
360         { \group_end: \prg_return_true: }
361         { \group_end: \prg_return_false: }
362       }
363       { \group_end: \prg_return_true: }
364     }
365   \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.

```

366 \prg_new_protected_conditional:Npnn \_unravel_gtl_if_head_is_definable:N #1
367 { TF , F }
368 {
369   \gtl_if_single_token:NTF #1
370   {
371     \gtl_if_head_is_N_type:NTF #1
372     {
373       \gtl_head_do:NN #1 \_unravel_token_if_definable:NTF
374       { \prg_return_true: }
375       { \prg_return_false: }
376     }
377     { \prg_return_false: }
378   }
379   { \prg_return_false: }
380 }

```

(End definition for `\_unravel_gtl_if_head_is_definable:NTF`.)

### 2.1.7 Helpers for previous input

```

\_unravel_prev_input_count:
  \_unravel_prev_input_count_aux:n
381 \cs_new_nopar:Npn \_unravel_prev_input_count:
382 {
383   \int_eval:n
384   {
385     0
386     \seq_map_function:NN \g__unravel_prev_input_seq
387     \_unravel_prev_input_count_aux:n
388   }
389 }
390 \cs_new:Npn \_unravel_prev_input_count_aux:n #1
391 { \tl_if_empty:nF {#1} { + 1 } }

```

(End definition for `\_unravel_prev_input_count:` and `\_unravel_prev_input_count_aux:n`.)

```

\_unravel_prev_input_get:N
\_unravel_prev_input_gpush:
  \_unravel_prev_input_gpush:N
\_unravel_prev_input_gpop:N
  \_unravel_prev_input_gpush_gtl:
  \_unravel_prev_input_gpush_gtl:N
  \_unravel_prev_input_gpop_gtl:N
392 \cs_new_protected_nopar:Npn \_unravel_prev_input_get:N
393 { \seq_get_right:NN \g__unravel_prev_input_seq }
394 \cs_new_protected_nopar:Npn \_unravel_prev_input_gpush:
395 { \seq_gput_right:Nn \g__unravel_prev_input_seq { } }
396 \cs_new_protected_nopar:Npn \_unravel_prev_input_gpush:N
397 { \seq_gput_right:NV \g__unravel_prev_input_seq }
398 \cs_new_protected_nopar:Npn \_unravel_prev_input_gpop:N
399 { \seq_gpop_right:NN \g__unravel_prev_input_seq }
400 \cs_new_protected_nopar:Npn \_unravel_prev_input_gpush_gtl:
401 { \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl }
402 \cs_new_protected_nopar:Npn \_unravel_prev_input_gpush_gtl:N
403 { \seq_gput_right:NV \g__unravel_prev_input_seq }

```



```

404 \cs_new_protected_nopar:Npn \__unravel_prev_input_gpop_gtl:N
405 { \seq_gpop_right:NN \g__unravel_prev_input_seq }

```

(End definition for \\_\_unravel\_prev\_input\_get:N and others.)

```

\__unravel_prev_input_silent:n
\__unravel_prev_input_silent:V
\__unravel_prev_input_silent:x
\__unravel_prev_input:n
\__unravel_prev_input:V
\__unravel_prev_input:x
406 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
407 {
408   \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
409   \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
410   \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
411 }
412 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V }
413 \cs_new_protected:Npn \__unravel_prev_input_silent:x
414 { \__unravel_exp_args:Nx \__unravel_prev_input_silent:n }
415 \cs_new_protected:Npn \__unravel_prev_input:n #1
416 {
417   \__unravel_prev_input_silent:n {#1}
418   \__unravel_print_action:x { \tl_to_str:n {#1} }
419 }
420 \cs_generate_variant:Nn \__unravel_prev_input:n { V }
421 \cs_new_protected:Npn \__unravel_prev_input:x
422 { \__unravel_exp_args:Nx \__unravel_prev_input:n }

```

(End definition for \\_\_unravel\_prev\_input\_silent:n and \\_\_unravel\_prev\_input:n.)

\\_\_unravel\_prev\_input\_gtl:N

```

423 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
424 {
425   \__unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
426   \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
427   \__unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
428 }

```

(End definition for \\_\_unravel\_prev\_input\_gtl:N.)

\\_\_unravel\_prev\_input\_join\_get:nN  
\\_\_unravel\_join\_get\_aux:NNN

Pops the previous-input sequence twice to get some value in \l\_\_unravel\_head\_tl and some sign or decimal number in \l\_\_unravel\_tmpa\_tl. Combines them into a value, using the appropriate evaluation function, determined based on #1.

```

429 \cs_new_protected:Npn \__unravel_prev_input_join_get:nN #1
430 {
431   \int_case:nnF {#1}
432   {
433     { 2 } { \__unravel_join_get_aux:NNN \skip_eval:n \etex_glueexpr:D }
434     { 3 } { \__unravel_join_get_aux:NNN \muskip_eval:n \etex_muexpr:D }
435   }
436   {
437     \__unravel_error:nnnnn { internal } { join-factor } { } { } { }
438     \__unravel_join_get_aux:NNN \use:n \prg_do_nothing:
439   }
440 }
441 \cs_new_protected:Npn \__unravel_join_get_aux:NNN #1#2#3
442 {
443   \__unravel_prev_input_gpop:N \l__unravel_head_tl
444   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl

```

```

445     \tl_set:Nx #3 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl } }
446 }

```

(End definition for \\_\_unravel\_prev\_input\_join\_get:nN and \\_\_unravel\_join\_get\_aux:NNN.)

## 2.2 Variables

### 2.2.1 User interaction

\g\_\_unravel\_before\_print\_state\_tl Code to run before printing the state or before the prompt.  
\g\_\_unravel\_before\_prompt\_tl

```

447 \tl_new:N \g__unravel_before_print_state_tl
448 \tl_new:N \g__unravel_before_prompt_tl

```

(End definition for \g\_\_unravel\_before\_print\_state\_tl and \g\_\_unravel\_before\_prompt\_tl.)

\l\_\_unravel\_prompt\_tmpa\_int

```

449 \int_new:N \l__unravel_prompt_tmpa_int

```

(End definition for \l\_\_unravel\_prompt\_tmpa\_int.)

\g\_\_unravel\_nonstop\_int The number of prompts to skip.  
\g\_\_unravel\_noise\_int

```

450 \int_new:N \g__unravel_nonstop_int
451 \int_new:N \g__unravel_noise_int
452 \int_gset_eq:NN \g__unravel_noise_int \c_one

```

(End definition for \g\_\_unravel\_nonstop\_int and \g\_\_unravel\_noise\_int.)

\g\_\_unravel\_default\_explicit\_prompt\_bool Variables for the options `explicit-prompt`, `internal-debug`, `number-steps`. The  
\g\_\_unravel\_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

```

453 \bool_new:N \g__unravel_default_explicit_prompt_bool
454 \bool_new:N \g__unravel_default_internal_debug_bool
455 \bool_new:N \g__unravel_default_number_steps_bool
456 \bool_gset_true:N \g__unravel_default_number_steps_bool
457 \bool_new:N \g__unravel_default_welcome_message_bool
458 \bool_gset_true:N \g__unravel_default_welcome_message_bool
459 \bool_new:N \g__unravel_explicit_prompt_bool
460 \bool_new:N \g__unravel_internal_debug_bool
461 \bool_new:N \g__unravel_number_steps_bool
462 \bool_new:N \g__unravel_welcome_message_bool

```

(End definition for \g\_\_unravel\_default\_explicit\_prompt\_bool and others.)

\g\_\_unravel\_step\_int Current expansion step.

```

463 \int_new:N \g__unravel_step_int

```

(End definition for \g\_\_unravel\_step\_int.)

\g\_\_unravel\_action\_text\_str Text describing the action, displayed at each step. This should only be altered through  
\\_\_unravel\_set\_action\_text:x, which sets the escape character as appropriate before  
converting the argument to a string.

```

464 \str_new:N \g__unravel_action_text_str

```

(End definition for \g\_\_unravel\_action\_text\_str.)

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

```

\g__unravel_default_max_output_int
\g__unravel_default_max_input_int
\g__unravel_max_action_int
\g__unravel_max_output_int
\g__unravel_max_input_int
465 \int_new:N \g__unravel_default_max_action_int
466 \int_new:N \g__unravel_default_max_output_int
467 \int_new:N \g__unravel_default_max_input_int
468 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
469 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
470 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
471 \int_new:N \g__unravel_max_action_int
472 \int_new:N \g__unravel_max_output_int
473 \int_new:N \g__unravel_max_input_int

```

*(End definition for \g\_\_unravel\_default\_max\_action\_int and others.)*

`\g__unravel_speedup_macros_bool` If this boolean is true, speed up macros which have a simple parameter text. This may not be safe if very weird macros appear.

```

474 \bool_new:N \g__unravel_speedup_macros_bool
475 \bool_gset_true:N \g__unravel_speedup_macros_bool

```

*(End definition for \g\_\_unravel\_speedup\_macros\_bool.)*

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

```

476 \int_new:N \l__unravel_print_int

```

*(End definition for \l\_\_unravel\_print\_int.)*

### 2.2.2 Working with tokens

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

```

477 \int_new:N \g__unravel_input_int

```

*(End definition for \g\_\_unravel\_input\_int.)*

`\g__unravel_input_tmpa_int`  
`\l__unravel_input_tmpa_tl`

```

478 \int_new:N \g__unravel_input_tmpa_int
479 \tl_new:N \l__unravel_input_tmpa_tl

```

*(End definition for \g\_\_unravel\_input\_tmpa\_int and \l\_\_unravel\_input\_tmpa\_tl.)*

`\g__unravel_prev_input_seq`  
`\l__unravel_prev_input_tl`  
`\l__unravel_prev_input_gtl` The different levels of expansion are stored in `\g__unravel_prev_input_seq`, with the innermost at the end of the sequence (otherwise the sequence would have to be reversed for display). When adding material to the last level of expansion, `\l__unravel_prev_input_tl` or `\l__unravel_prev_input_gtl` are used to temporarily store the last level of expansion.

```

480 \seq_new:N \g__unravel_prev_input_seq
481 \tl_new:N \l__unravel_prev_input_tl
482 \gtl_new:N \l__unravel_prev_input_gtl

```

*(End definition for \g\_\_unravel\_prev\_input\_seq, \l\_\_unravel\_prev\_input\_tl, and \l\_\_unravel\_prev\_input\_gtl.)*

`\g__unravel_output_gtl` Material that is “typeset” or otherwise sent further down T<sub>E</sub>X’s digestion.

483 `\gtl_new:N \g__unravel_output_gtl`

(End definition for `\g__unravel_output_gtl`.)

`\l__unravel_head_gtl` First token in the input, as a generalized token list (general case) or as a token list  
`\l__unravel_head_tl` whenever this is possible. Also, a token set equal to it, and its command code and  
`\l__unravel_head_token` character code, following T<sub>E</sub>X.

484 `\gtl_new:N \l__unravel_head_gtl`  
485 `\tl_new:N \l__unravel_head_tl`  
486 `\cs_new_eq:NN \l__unravel_head_token ?`  
487 `\int_new:N \l__unravel_head_cmd_int`  
488 `\int_new:N \l__unravel_head_char_int`

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

`\l__unravel_head_meaning_tl`

489 `\tl_new:N \l__unravel_head_meaning_tl`

(End definition for `\l__unravel_head_meaning_tl`.)

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

`\g__unravel_tmpc_tl` 490 `\tl_new:N \l__unravel_tmpa_tl`  
`\l__unravel_tmpa_seq` 491 `\gtl_new:N \l__unravel_unused_gtl`  
`\l__unravel_unused_gtl` 492 `\gtl_new:N \l__unravel_tmpb_gtl`  
493 `\tl_new:N \g__unravel_tmpc_tl`  
494 `\seq_new:N \l__unravel_tmpa_seq`

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

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

495 `\tl_new:N \l__unravel_defined_tl`  
496 `\tl_new:N \l__unravel_defining_tl`

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

`\__unravel_inaccessible:w`

497 `\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 T<sub>E</sub>X. Token to insert after the next assign-  
`\g__unravel_set_box_allowed_bool` ment. Is `\setbox` currently allowed? Should `\input` expand?  
`\g__unravel_name_in_progress_bool`

498 `\gtl_new:N \g__unravel_after_assignment_gtl`  
499 `\bool_new:N \g__unravel_set_box_allowed_bool`  
500 `\bool_new:N \g__unravel_name_in_progress_bool`

(End definition for `\g__unravel_after_assignment_gtl`, `\g__unravel_set_box_allowed_bool`, and  
`\g__unravel_name_in_progress_bool`.)

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

```
501 \gtl_new:N \l__unravel_after_group_gtl
```

(End definition for `\l__unravel_after_group_gtl`.)

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

```
502 \group_begin:
503   \cs_set:Npx \__unravel_tmp:w #1 { \c_hash_str #1 }
504   \tl_const:Nx \c__unravel_parameters_tl
505     { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
506 \group_end:
```

(End definition for `\c__unravel_parameters_tl`.)

### 2.2.3 Numbers and conditionals

`\g__unravel_val_level_int` See TeX's `cur_val_level` variable. This is set by `\__unravel_scan_something_internal:n` to

- 0 for integer values,
- 1 for dimension values,
- 2 for glue values,
- 3 for mu glue values,
- 4 for font identifiers,
- 5 for token lists.

```
507 \int_new:N \g__unravel_val_level_int
```

(End definition for `\g__unravel_val_level_int`.)

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

```
\g__unravel_if_limit_int 508 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 509 \int_new:N \g__unravel_if_limit_int
510 \int_new:N \g__unravel_if_depth_int
```

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

`\l__unravel_if_nesting_int`

```
511 \int_new:N \l__unravel_if_nesting_int
```

(End definition for `\l__unravel_if_nesting_int`.)

## 2.2.4 Boxes and groups

`\l__unravel_leaders_box_seq` A stack of letters: the first token in the token list is `h` if the innermost explicit box (created with `\vtop`, `\vbox`, or `\hbox`) appears in a horizontal (or math) mode leaders construction; it is `v` if the innermost explicit box appears in a vertical mode leaders construction; it is `Z` otherwise.

512 `\seq_new:N \l__unravel_leaders_box_seq`

(End definition for `\l__unravel_leaders_box_seq`.)

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

513 `\int_new:N \g__unravel_ends_int`

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

(End definition for `\g__unravel_ends_int`.)

## 2.2.5 Constants

`\c__unravel_plus_tl`  
`\c__unravel_minus_tl` 515 `\tl_const:Nn \c__unravel_plus_tl { + }`  
`\c__unravel_times_tl` 516 `\tl_const:Nn \c__unravel_minus_tl { - }`  
`\c__unravel_over_tl` 517 `\tl_const:Nn \c__unravel_times_tl { * }`  
`\c__unravel_lq_tl` 518 `\tl_const:Nn \c__unravel_over_tl { / }`  
`\c__unravel_rq_tl` 519 `\tl_const:Nn \c__unravel_lq_tl { ‘ }`  
`\c__unravel_dq_tl` 520 `\tl_const:Nn \c__unravel_rq_tl { ’ }`  
`\c__unravel_lp_tl` 521 `\tl_const:Nn \c__unravel_dq_tl { " }`  
`\c__unravel_rp_tl` 522 `\tl_const:Nn \c__unravel_lp_tl { ( }`  
`\c__unravel_eq_tl` 523 `\tl_const:Nn \c__unravel_rp_tl { ) }`  
`\c__unravel_comma_tl` 524 `\tl_const:Nn \c__unravel_eq_tl { = }`  
`\c__unravel_point_tl` 525 `\tl_const:Nn \c__unravel_comma_tl { , }`  
526 `\tl_const:Nn \c__unravel_point_tl { . }`

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

`\c__unravel_frozen_relax_gtl` T<sub>E</sub>X’s `frozen_relax`, inserted by `\__unravel_insert_relax:`.

527 `\gtl_const:Nx \c__unravel_frozen_relax_gtl { \if_int_compare:w 0 = 0 \fi: }`

(End definition for `\c__unravel_frozen_relax_gtl`.)

## 2.2.6 T<sub>E</sub>X parameters

`\g__unravel_mag_set_int` The first time T<sub>E</sub>X uses the value of `\mag`, it stores it in a global parameter `mag_set` (initially 0 to denote not being set). Any time T<sub>E</sub>X 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`.

528 `\int_new:N \g__unravel_mag_set_int`

(End definition for `\g__unravel_mag_set_int`.)

## 2.3 Numeric codes

First we define some numeric codes, following Section 15 of the T<sub>E</sub>X web code, then we associate a command code to each T<sub>E</sub>X primitive, and a character code, to decide what action to perform upon seeing them.

```

\__unravel_tex_const:nn
\__unravel_tex_use:n
529 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
530 { \int_const:cn { c__unravel_tex_#1_int } {#2} }
531 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }

```

(End definition for \\_\_unravel\_tex\_const:nn and \\_\_unravel\_tex\_use:n.)

```

\__unravel_tex_primitive:nnn
532 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
533 {
534   \tl_const:cx { c__unravel_tex_#1_tl }
535   { { \__unravel_tex_use:n {#2} } {#3} }
536 }

```

(End definition for \\_\_unravel\_tex\_primitive:nnn.)

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
537 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
538 {
539   \cs_new_protected_nopar:cpn
540   { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
541 }
542 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
543 {
544   \cs_new_eq:cc
545   { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
546   { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
547 }

```

(End definition for \\_\_unravel\_new\_tex\_cmd:nn and \\_\_unravel\_new\_eq\_tex\_cmd:nn.)

```

\__unravel_new_tex_expandable:nn
548 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
549 {
550   \cs_new_protected_nopar:cpn
551   { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
552 }

```

(End definition for \\_\_unravel\_new\_tex\_expandable:nn.)

Contrarily to T<sub>E</sub>X, all macros are call, no long\_call and the like.

```

553 \__unravel_tex_const:nn { relax } { 0 }
554 \__unravel_tex_const:nn { begin-group_char } { 1 }
555 \__unravel_tex_const:nn { end-group_char } { 2 }
556 \__unravel_tex_const:nn { math_char } { 3 }
557 \__unravel_tex_const:nn { tab_mark } { 4 }
558 \__unravel_tex_const:nn { alignment_char } { 4 }
559 \__unravel_tex_const:nn { car_ret } { 5 }
560 \__unravel_tex_const:nn { macro_char } { 6 }
561 \__unravel_tex_const:nn { superscript_char } { 7 }

```

```

562 \_unravel_tex_const:nn { subscript_char } { 8 }
563 \_unravel_tex_const:nn { endv } { 9 }
564 \_unravel_tex_const:nn { blank_char } { 10 }
565 \_unravel_tex_const:nn { the_char } { 11 }
566 \_unravel_tex_const:nn { other_char } { 12 }
567 \_unravel_tex_const:nn { par_end } { 13 }
568 \_unravel_tex_const:nn { stop } { 14 }
569 \_unravel_tex_const:nn { delim_num } { 15 }
570 \_unravel_tex_const:nn { max_char_code } { 15 }
571 \_unravel_tex_const:nn { char_num } { 16 }
572 \_unravel_tex_const:nn { math_char_num } { 17 }
573 \_unravel_tex_const:nn { mark } { 18 }
574 \_unravel_tex_const:nn { xray } { 19 }
575 \_unravel_tex_const:nn { make_box } { 20 }
576 \_unravel_tex_const:nn { hmove } { 21 }
577 \_unravel_tex_const:nn { vmove } { 22 }
578 \_unravel_tex_const:nn { un_hbox } { 23 }
579 \_unravel_tex_const:nn { un_vbox } { 24 }
580 \_unravel_tex_const:nn { remove_item } { 25 }
581 \_unravel_tex_const:nn { hskip } { 26 }
582 \_unravel_tex_const:nn { vskip } { 27 }
583 \_unravel_tex_const:nn { mskip } { 28 }
584 \_unravel_tex_const:nn { kern } { 29 }
585 \_unravel_tex_const:nn { mkern } { 30 }
586 \_unravel_tex_const:nn { leader_ship } { 31 }
587 \_unravel_tex_const:nn { halign } { 32 }
588 \_unravel_tex_const:nn { valign } { 33 }
589 \_unravel_tex_const:nn { no_align } { 34 }
590 \_unravel_tex_const:nn { vrule } { 35 }
591 \_unravel_tex_const:nn { hrule } { 36 }
592 \_unravel_tex_const:nn { insert } { 37 }
593 \_unravel_tex_const:nn { vadjust } { 38 }
594 \_unravel_tex_const:nn { ignore_spaces } { 39 }
595 \_unravel_tex_const:nn { after_assignment } { 40 }
596 \_unravel_tex_const:nn { after_group } { 41 }
597 \_unravel_tex_const:nn { break_penalty } { 42 }
598 \_unravel_tex_const:nn { start_par } { 43 }
599 \_unravel_tex_const:nn { ital_corr } { 44 }
600 \_unravel_tex_const:nn { accent } { 45 }
601 \_unravel_tex_const:nn { math_accent } { 46 }
602 \_unravel_tex_const:nn { discretionary } { 47 }
603 \_unravel_tex_const:nn { eq_no } { 48 }
604 \_unravel_tex_const:nn { left_right } { 49 }
605 \_unravel_tex_const:nn { math_comp } { 50 }
606 \_unravel_tex_const:nn { limit_switch } { 51 }
607 \_unravel_tex_const:nn { above } { 52 }
608 \_unravel_tex_const:nn { math_style } { 53 }
609 \_unravel_tex_const:nn { math_choice } { 54 }
610 \_unravel_tex_const:nn { non_script } { 55 }
611 \_unravel_tex_const:nn { vcenter } { 56 }
612 \_unravel_tex_const:nn { case_shift } { 57 }
613 \_unravel_tex_const:nn { message } { 58 }
614 \_unravel_tex_const:nn { extension } { 59 }
615 \_unravel_tex_const:nn { in_stream } { 60 }

```



```

616 \__unravel_tex_const:nn { begin_group } { 61 }
617 \__unravel_tex_const:nn { end_group } { 62 }
618 \__unravel_tex_const:nn { omit } { 63 }
619 \__unravel_tex_const:nn { ex_space } { 64 }
620 \__unravel_tex_const:nn { no_boundary } { 65 }
621 \__unravel_tex_const:nn { radical } { 66 }
622 \__unravel_tex_const:nn { end_cs_name } { 67 }
623 \__unravel_tex_const:nn { min_internal } { 68 }
624 \__unravel_tex_const:nn { char_given } { 68 }
625 \__unravel_tex_const:nn { math_given } { 69 }
626 \__unravel_tex_const:nn { last_item } { 70 }
627 \__unravel_tex_const:nn { max_non_prefixed_command } { 70 }
628 \__unravel_tex_const:nn { toks_register } { 71 }
629 \__unravel_tex_const:nn { assign_toks } { 72 }
630 \__unravel_tex_const:nn { assign_int } { 73 }
631 \__unravel_tex_const:nn { assign_dimen } { 74 }
632 \__unravel_tex_const:nn { assign_glue } { 75 }
633 \__unravel_tex_const:nn { assign_mu_glue } { 76 }
634 \__unravel_tex_const:nn { assign_font_dimen } { 77 }
635 \__unravel_tex_const:nn { assign_font_int } { 78 }
636 \__unravel_tex_const:nn { set_aux } { 79 }
637 \__unravel_tex_const:nn { set_prev_graf } { 80 }
638 \__unravel_tex_const:nn { set_page_dimen } { 81 }
639 \__unravel_tex_const:nn { set_page_int } { 82 }
640 \__unravel_tex_const:nn { set_box_dimen } { 83 }
641 \__unravel_tex_const:nn { set_shape } { 84 }
642 \__unravel_tex_const:nn { def_code } { 85 }
643 \__unravel_tex_const:nn { def_family } { 86 }
644 \__unravel_tex_const:nn { set_font } { 87 }
645 \__unravel_tex_const:nn { def_font } { 88 }
646 \__unravel_tex_const:nn { register } { 89 }
647 \__unravel_tex_const:nn { max_internal } { 89 }
648 \__unravel_tex_const:nn { advance } { 90 }
649 \__unravel_tex_const:nn { multiply } { 91 }
650 \__unravel_tex_const:nn { divide } { 92 }
651 \__unravel_tex_const:nn { prefix } { 93 }
652 \__unravel_tex_const:nn { let } { 94 }
653 \__unravel_tex_const:nn { shorthand_def } { 95 }
654 \__unravel_tex_const:nn { read_to_cs } { 96 }
655 \__unravel_tex_const:nn { def } { 97 }
656 \__unravel_tex_const:nn { set_box } { 98 }
657 \__unravel_tex_const:nn { hyph_data } { 99 }
658 \__unravel_tex_const:nn { set_interaction } { 100 }
659 \__unravel_tex_const:nn { letterspace_font } { 101 }
660 \__unravel_tex_const:nn { pdf_copy_font } { 102 }
661 \__unravel_tex_const:nn { max_command } { 102 }
662 \__unravel_tex_const:nn { undefined_cs } { 103 }
663 \__unravel_tex_const:nn { expand_after } { 104 }
664 \__unravel_tex_const:nn { no_expand } { 105 }
665 \__unravel_tex_const:nn { input } { 106 }
666 \__unravel_tex_const:nn { if_test } { 107 }
667 \__unravel_tex_const:nn { fi_or_else } { 108 }
668 \__unravel_tex_const:nn { cs_name } { 109 }
669 \__unravel_tex_const:nn { convert } { 110 }

```

```

670 \__unravel_tex_const:nn { the } { 111 }
671 \__unravel_tex_const:nn { top_bot_mark } { 112 }
672 \__unravel_tex_const:nn { call } { 113 }
673 \__unravel_tex_const:nn { end_template } { 117 }

```

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

A few minor differences with pdfTeX's internal numbers are as follows.

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

```

674 \__unravel_tex_primitive:nnn { relax } { relax } { 256 }
675 \__unravel_tex_primitive:nnn { span } { tab_mark } { 256 }
676 \__unravel_tex_primitive:nnn { cr } { car_ret } { 257 }
677 \__unravel_tex_primitive:nnn { crcr } { car_ret } { 258 }
678 \__unravel_tex_primitive:nnn { par } { par_end } { 256 }
679 \__unravel_tex_primitive:nnn { end } { stop } { 0 }
680 \__unravel_tex_primitive:nnn { dump } { stop } { 1 }
681 \__unravel_tex_primitive:nnn { delimiter } { delim_num } { 0 }
682 \__unravel_tex_primitive:nnn { char } { char_num } { 0 }
683 \__unravel_tex_primitive:nnn { mathchar } { math_char_num } { 0 }
684 \__unravel_tex_primitive:nnn { mark } { mark } { 0 }
685 \__unravel_tex_primitive:nnn { marks } { mark } { 5 }
686 \__unravel_tex_primitive:nnn { show } { xray } { 0 }
687 \__unravel_tex_primitive:nnn { showbox } { xray } { 1 }
688 \__unravel_tex_primitive:nnn { showthe } { xray } { 2 }
689 \__unravel_tex_primitive:nnn { showlists } { xray } { 3 }
690 \__unravel_tex_primitive:nnn { showgroups } { xray } { 4 }
691 \__unravel_tex_primitive:nnn { showtokens } { xray } { 5 }
692 \__unravel_tex_primitive:nnn { showifs } { xray } { 6 }
693 \__unravel_tex_primitive:nnn { box } { make_box } { 0 }
694 \__unravel_tex_primitive:nnn { copy } { make_box } { 1 }
695 \__unravel_tex_primitive:nnn { lastbox } { make_box } { 2 }
696 \__unravel_tex_primitive:nnn { vsplit } { make_box } { 3 }
697 \__unravel_tex_primitive:nnn { vtop } { make_box } { 4 }
698 \__unravel_tex_primitive:nnn { vbox } { make_box } { 5 }
699 \__unravel_tex_primitive:nnn { hbox } { make_box } { 106 }
700 \__unravel_tex_primitive:nnn { moveright } { hmove } { 0 }
701 \__unravel_tex_primitive:nnn { moveleft } { hmove } { 1 }
702 \__unravel_tex_primitive:nnn { lower } { vmove } { 0 }
703 \__unravel_tex_primitive:nnn { raise } { vmove } { 1 }
704 \__unravel_tex_primitive:nnn { unhbox } { un_hbox } { 0 }
705 \__unravel_tex_primitive:nnn { unhcopy } { un_hbox } { 1 }
706 \__unravel_tex_primitive:nnn { unvbox } { un_vbox } { 0 }

```

707	<code>\_unravel\_tex\_primitive:nnn { unvcopy</code>	<code>} { un\_vbox } { 1 }</code>
708	<code>\_unravel\_tex\_primitive:nnn { pagediscards</code>	<code>} { un\_vbox } { 2 }</code>
709	<code>\_unravel\_tex\_primitive:nnn { splitdiscards</code>	<code>} { un\_vbox } { 3 }</code>
710	<code>\_unravel\_tex\_primitive:nnn { unpenalty</code>	<code>} { remove\_item } { 12 }</code>
711	<code>\_unravel\_tex\_primitive:nnn { unkern</code>	<code>} { remove\_item } { 11 }</code>
712	<code>\_unravel\_tex\_primitive:nnn { unskip</code>	<code>} { remove\_item } { 10 }</code>
713	<code>\_unravel\_tex\_primitive:nnn { hfil</code>	<code>} { hskip } { 0 }</code>
714	<code>\_unravel\_tex\_primitive:nnn { hfill</code>	<code>} { hskip } { 1 }</code>
715	<code>\_unravel\_tex\_primitive:nnn { hss</code>	<code>} { hskip } { 2 }</code>
716	<code>\_unravel\_tex\_primitive:nnn { hfilneg</code>	<code>} { hskip } { 3 }</code>
717	<code>\_unravel\_tex\_primitive:nnn { hskip</code>	<code>} { hskip } { 4 }</code>
718	<code>\_unravel\_tex\_primitive:nnn { vfil</code>	<code>} { vskip } { 0 }</code>
719	<code>\_unravel\_tex\_primitive:nnn { vfill</code>	<code>} { vskip } { 1 }</code>
720	<code>\_unravel\_tex\_primitive:nnn { vss</code>	<code>} { vskip } { 2 }</code>
721	<code>\_unravel\_tex\_primitive:nnn { vfilneg</code>	<code>} { vskip } { 3 }</code>
722	<code>\_unravel\_tex\_primitive:nnn { vskip</code>	<code>} { vskip } { 4 }</code>
723	<code>\_unravel\_tex\_primitive:nnn { mskip</code>	<code>} { mskip } { 5 }</code>
724	<code>\_unravel\_tex\_primitive:nnn { kern</code>	<code>} { kern } { 1 }</code>
725	<code>\_unravel\_tex\_primitive:nnn { mkern</code>	<code>} { mkern } { 99 }</code>
726	<code>\_unravel\_tex\_primitive:nnn { shipout</code>	<code>} { leader\_ship } { 99 }</code>
727	<code>\_unravel\_tex\_primitive:nnn { leaders</code>	<code>} { leader\_ship } { 100 }</code>
728	<code>\_unravel\_tex\_primitive:nnn { cleaders</code>	<code>} { leader\_ship } { 101 }</code>
729	<code>\_unravel\_tex\_primitive:nnn { xleaders</code>	<code>} { leader\_ship } { 102 }</code>
730	<code>\_unravel\_tex\_primitive:nnn { halign</code>	<code>} { halign } { 0 }</code>
731	<code>\_unravel\_tex\_primitive:nnn { valign</code>	<code>} { valign } { 0 }</code>
732	<code>\_unravel\_tex\_primitive:nnn { beginL</code>	<code>} { valign } { 4 }</code>
733	<code>\_unravel\_tex\_primitive:nnn { endL</code>	<code>} { valign } { 5 }</code>
734	<code>\_unravel\_tex\_primitive:nnn { beginR</code>	<code>} { valign } { 8 }</code>
735	<code>\_unravel\_tex\_primitive:nnn { endR</code>	<code>} { valign } { 9 }</code>
736	<code>\_unravel\_tex\_primitive:nnn { noalign</code>	<code>} { no\_align } { 0 }</code>
737	<code>\_unravel\_tex\_primitive:nnn { vrule</code>	<code>} { vrule } { 0 }</code>
738	<code>\_unravel\_tex\_primitive:nnn { hrule</code>	<code>} { hrule } { 0 }</code>
739	<code>\_unravel\_tex\_primitive:nnn { insert</code>	<code>} { insert } { 0 }</code>
740	<code>\_unravel\_tex\_primitive:nnn { vadjust</code>	<code>} { vadjust } { 0 }</code>
741	<code>\_unravel\_tex\_primitive:nnn { ignorespaces</code>	<code>} { ignore\_spaces } { 0 }</code>
742	<code>\_unravel\_tex\_primitive:nnn { afterassignment</code>	<code>} { after\_assignment } { 0 }</code>
743	<code>\_unravel\_tex\_primitive:nnn { aftergroup</code>	<code>} { after\_group } { 0 }</code>
744	<code>\_unravel\_tex\_primitive:nnn { penalty</code>	<code>} { break\_penalty } { 0 }</code>
745	<code>\_unravel\_tex\_primitive:nnn { indent</code>	<code>} { start\_par } { 1 }</code>
746	<code>\_unravel\_tex\_primitive:nnn { noindent</code>	<code>} { start\_par } { 0 }</code>
747	<code>\_unravel\_tex\_primitive:nnn { quitvmode</code>	<code>} { start\_par } { 2 }</code>
748	<code>\_unravel\_tex\_primitive:nnn { /</code>	<code>} { ital\_corr } { 0 }</code>
749	<code>\_unravel\_tex\_primitive:nnn { accent</code>	<code>} { accent } { 0 }</code>
750	<code>\_unravel\_tex\_primitive:nnn { mathaccent</code>	<code>} { math\_accent } { 0 }</code>
751	<code>\_unravel\_tex\_primitive:nnn { -</code>	<code>} { discretionary } { 1 }</code>
752	<code>\_unravel\_tex\_primitive:nnn { discretionary</code>	<code>} { discretionary } { 0 }</code>
753	<code>\_unravel\_tex\_primitive:nnn { eqno</code>	<code>} { eq\_no } { 0 }</code>
754	<code>\_unravel\_tex\_primitive:nnn { leqno</code>	<code>} { eq\_no } { 1 }</code>
755	<code>\_unravel\_tex\_primitive:nnn { left</code>	<code>} { left\_right } { 30 }</code>
756	<code>\_unravel\_tex\_primitive:nnn { right</code>	<code>} { left\_right } { 31 }</code>
757	<code>\_unravel\_tex\_primitive:nnn { middle</code>	<code>} { left\_right } { 17 }</code>
758	<code>\_unravel\_tex\_primitive:nnn { mathord</code>	<code>} { math\_comp } { 16 }</code>
759	<code>\_unravel\_tex\_primitive:nnn { mathop</code>	<code>} { math\_comp } { 17 }</code>
760	<code>\_unravel\_tex\_primitive:nnn { mathbin</code>	<code>} { math\_comp } { 18 }</code>

```

761 \_unravel_tex_primitive:nnn { mathrel } { math_comp } { 19 }
762 \_unravel_tex_primitive:nnn { mathopen } { math_comp } { 20 }
763 \_unravel_tex_primitive:nnn { mathclose } { math_comp } { 21 }
764 \_unravel_tex_primitive:nnn { mathpunct } { math_comp } { 22 }
765 \_unravel_tex_primitive:nnn { mathinner } { math_comp } { 23 }
766 \_unravel_tex_primitive:nnn { underline } { math_comp } { 26 }
767 \_unravel_tex_primitive:nnn { overline } { math_comp } { 27 }
768 \_unravel_tex_primitive:nnn { displaylimits } { limit_switch } { 0 }
769 \_unravel_tex_primitive:nnn { limits } { limit_switch } { 1 }
770 \_unravel_tex_primitive:nnn { nolimits } { limit_switch } { 2 }
771 \_unravel_tex_primitive:nnn { above } { above } { 0 }
772 \_unravel_tex_primitive:nnn { over } { above } { 1 }
773 \_unravel_tex_primitive:nnn { atop } { above } { 2 }
774 \_unravel_tex_primitive:nnn { abovewithdelims } { above } { 3 }
775 \_unravel_tex_primitive:nnn { overwithdelims } { above } { 4 }
776 \_unravel_tex_primitive:nnn { atopwithdelims } { above } { 5 }
777 \_unravel_tex_primitive:nnn { displaystyle } { math_style } { 0 }
778 \_unravel_tex_primitive:nnn { textstyle } { math_style } { 2 }
779 \_unravel_tex_primitive:nnn { scriptstyle } { math_style } { 4 }
780 \_unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
781 \_unravel_tex_primitive:nnn { mathchoice } { math_choice } { 0 }
782 \_unravel_tex_primitive:nnn { nonscript } { non_script } { 0 }
783 \_unravel_tex_primitive:nnn { vcenter } { vcenter } { 0 }
784 \_unravel_tex_primitive:nnn { lowercase } { case_shift } { 256 }
785 \_unravel_tex_primitive:nnn { uppercase } { case_shift } { 512 }
786 \_unravel_tex_primitive:nnn { message } { message } { 0 }
787 \_unravel_tex_primitive:nnn { errmessage } { message } { 1 }
788 \_unravel_tex_primitive:nnn { openout } { extension } { 0 }
789 \_unravel_tex_primitive:nnn { write } { extension } { 1 }
790 \_unravel_tex_primitive:nnn { closeout } { extension } { 2 }
791 \_unravel_tex_primitive:nnn { special } { extension } { 3 }
792 \_unravel_tex_primitive:nnn { immediate } { extension } { 4 }
793 \_unravel_tex_primitive:nnn { setlanguage } { extension } { 5 }
794 \_unravel_tex_primitive:nnn { pdfliteral } { extension } { 6 }
795 \_unravel_tex_primitive:nnn { pdfobj } { extension } { 7 }
796 \_unravel_tex_primitive:nnn { pdfrefobj } { extension } { 8 }
797 \_unravel_tex_primitive:nnn { pdfxform } { extension } { 9 }
798 \_unravel_tex_primitive:nnn { pdfrefxform } { extension } { 10 }
799 \_unravel_tex_primitive:nnn { pdfximage } { extension } { 11 }
800 \_unravel_tex_primitive:nnn { pdfrefximage } { extension } { 12 }
801 \_unravel_tex_primitive:nnn { pdfannot } { extension } { 13 }
802 \_unravel_tex_primitive:nnn { pdfstartlink } { extension } { 14 }
803 \_unravel_tex_primitive:nnn { pdfendlink } { extension } { 15 }
804 \_unravel_tex_primitive:nnn { pdfoutline } { extension } { 16 }
805 \_unravel_tex_primitive:nnn { pdfdest } { extension } { 17 }
806 \_unravel_tex_primitive:nnn { pdfthread } { extension } { 18 }
807 \_unravel_tex_primitive:nnn { pdfstartthread } { extension } { 19 }
808 \_unravel_tex_primitive:nnn { pdfendthread } { extension } { 20 }
809 \_unravel_tex_primitive:nnn { pdfsavepos } { extension } { 21 }
810 \_unravel_tex_primitive:nnn { pdfinfo } { extension } { 22 }
811 \_unravel_tex_primitive:nnn { pdfcatalog } { extension } { 23 }
812 \_unravel_tex_primitive:nnn { pdfnames } { extension } { 24 }
813 \_unravel_tex_primitive:nnn { pdffontattr } { extension } { 25 }
814 \_unravel_tex_primitive:nnn { pdfincludechars } { extension } { 26 }

```

```

815 \__unravel_tex_primitive:nnn { pdfmapfile } { extension } { 27 }
816 \__unravel_tex_primitive:nnn { pdfmapline } { extension } { 28 }
817 \__unravel_tex_primitive:nnn { pdftrailer } { extension } { 29 }
818 \__unravel_tex_primitive:nnn { pdfresettimer } { extension } { 30 }
819 \__unravel_tex_primitive:nnn { pdffontexpand } { extension } { 31 }
820 \__unravel_tex_primitive:nnn { pdfsetrandomseed } { extension } { 32 }
821 \__unravel_tex_primitive:nnn { pdfsnaprefpoint } { extension } { 33 }
822 \__unravel_tex_primitive:nnn { pdfsnapy } { extension } { 34 }
823 \__unravel_tex_primitive:nnn { pdfsnapycomp } { extension } { 35 }
824 \__unravel_tex_primitive:nnn { pdfglyphtounicode } { extension } { 36 }
825 \__unravel_tex_primitive:nnn { pdfcolorstack } { extension } { 37 }
826 \__unravel_tex_primitive:nnn { pdfsetmatrix } { extension } { 38 }
827 \__unravel_tex_primitive:nnn { pdfsave } { extension } { 39 }
828 \__unravel_tex_primitive:nnn { pdfrestore } { extension } { 40 }
829 \__unravel_tex_primitive:nnn { pdfnobuiltintounicode } { extension } { 41 }
830 \__unravel_tex_primitive:nnn { openin } { in_stream } { 1 }
831 \__unravel_tex_primitive:nnn { closein } { in_stream } { 0 }
832 \__unravel_tex_primitive:nnn { begingroup } { begin_group } { 0 }
833 \__unravel_tex_primitive:nnn { endgroup } { end_group } { 0 }
834 \__unravel_tex_primitive:nnn { omit } { omit } { 0 }
835 \__unravel_tex_primitive:nnn { ~ } { ex_space } { 0 }
836 \__unravel_tex_primitive:nnn { noboundary } { no_boundary } { 0 }
837 \__unravel_tex_primitive:nnn { radical } { radical } { 0 }
838 \__unravel_tex_primitive:nnn { endcsname } { end_cs_name } { 0 }
839 \__unravel_tex_primitive:nnn { lastpenalty } { last_item } { 0 }
840 \__unravel_tex_primitive:nnn { lastkern } { last_item } { 1 }
841 \__unravel_tex_primitive:nnn { lastskip } { last_item } { 2 }
842 \__unravel_tex_primitive:nnn { lastnodetype } { last_item } { 3 }
843 \__unravel_tex_primitive:nnn { inputlineno } { last_item } { 4 }
844 \__unravel_tex_primitive:nnn { badness } { last_item } { 5 }
845 \__unravel_tex_primitive:nnn { pdftexversion } { last_item } { 6 }
846 \__unravel_tex_primitive:nnn { pdflastobj } { last_item } { 7 }
847 \__unravel_tex_primitive:nnn { pdflastxform } { last_item } { 8 }
848 \__unravel_tex_primitive:nnn { pdflastximage } { last_item } { 9 }
849 \__unravel_tex_primitive:nnn { pdflastximagepages } { last_item } { 10 }
850 \__unravel_tex_primitive:nnn { pdflastannot } { last_item } { 11 }
851 \__unravel_tex_primitive:nnn { pdflastxpos } { last_item } { 12 }
852 \__unravel_tex_primitive:nnn { pdflastypos } { last_item } { 13 }
853 \__unravel_tex_primitive:nnn { pdfretval } { last_item } { 14 }
854 \__unravel_tex_primitive:nnn { pdflastximagecolordepth } { last_item } { 15 }
855 \__unravel_tex_primitive:nnn { pdfelapstime } { last_item } { 16 }
856 \__unravel_tex_primitive:nnn { pdfshellescape } { last_item } { 17 }
857 \__unravel_tex_primitive:nnn { pdfrandomseed } { last_item } { 18 }
858 \__unravel_tex_primitive:nnn { pdflastlink } { last_item } { 19 }
859 \__unravel_tex_primitive:nnn { eTeXversion } { last_item } { 20 }
860 \__unravel_tex_primitive:nnn { currentgrouplevel } { last_item } { 21 }
861 \__unravel_tex_primitive:nnn { currentgrouptype } { last_item } { 22 }
862 \__unravel_tex_primitive:nnn { currentiflevel } { last_item } { 23 }
863 \__unravel_tex_primitive:nnn { currentifttype } { last_item } { 24 }
864 \__unravel_tex_primitive:nnn { currentifbranch } { last_item } { 25 }
865 \__unravel_tex_primitive:nnn { gluestretchorder } { last_item } { 26 }
866 \__unravel_tex_primitive:nnn { glueshrinkorder } { last_item } { 27 }
867 \__unravel_tex_primitive:nnn { fontcharwd } { last_item } { 28 }
868 \__unravel_tex_primitive:nnn { fontcharht } { last_item } { 29 }

```

```

869 \_unravel\_tex\_primitive:nnn { fontchardp } { last\_item } { 30 }
870 \_unravel\_tex\_primitive:nnn { fontcharic } { last\_item } { 31 }
871 \_unravel\_tex\_primitive:nnn { parshaplength } { last\_item } { 32 }
872 \_unravel\_tex\_primitive:nnn { parshapeindent } { last\_item } { 33 }
873 \_unravel\_tex\_primitive:nnn { parshapedimen } { last\_item } { 34 }
874 \_unravel\_tex\_primitive:nnn { gluestretch } { last\_item } { 35 }
875 \_unravel\_tex\_primitive:nnn { glueshrink } { last\_item } { 36 }
876 \_unravel\_tex\_primitive:nnn { mutoglue } { last\_item } { 37 }
877 \_unravel\_tex\_primitive:nnn { gluetomu } { last\_item } { 38 }
878 \_unravel\_tex\_primitive:nnn { numexpr } { last\_item } { 39 }
879 \_unravel\_tex\_primitive:nnn { dimexpr } { last\_item } { 40 }
880 \_unravel\_tex\_primitive:nnn { glueexpr } { last\_item } { 41 }
881 \_unravel\_tex\_primitive:nnn { muexpr } { last\_item } { 42 }
882 \_unravel\_tex\_primitive:nnn { toks } { toks\_register } { 0 }
883 \_unravel\_tex\_primitive:nnn { output } { assign\_toks } { 1 }
884 \_unravel\_tex\_primitive:nnn { everypar } { assign\_toks } { 2 }
885 \_unravel\_tex\_primitive:nnn { everymath } { assign\_toks } { 3 }
886 \_unravel\_tex\_primitive:nnn { everydisplay } { assign\_toks } { 4 }
887 \_unravel\_tex\_primitive:nnn { everyhbox } { assign\_toks } { 5 }
888 \_unravel\_tex\_primitive:nnn { everyvbox } { assign\_toks } { 6 }
889 \_unravel\_tex\_primitive:nnn { everyjob } { assign\_toks } { 7 }
890 \_unravel\_tex\_primitive:nnn { everycr } { assign\_toks } { 8 }
891 \_unravel\_tex\_primitive:nnn { errhelp } { assign\_toks } { 9 }
892 \_unravel\_tex\_primitive:nnn { pdfpagesattr } { assign\_toks } { 10 }
893 \_unravel\_tex\_primitive:nnn { pdfpageattr } { assign\_toks } { 11 }
894 \_unravel\_tex\_primitive:nnn { pdfpageresources } { assign\_toks } { 12 }
895 \_unravel\_tex\_primitive:nnn { pdfpkmode } { assign\_toks } { 13 }
896 \_unravel\_tex\_primitive:nnn { everyeof } { assign\_toks } { 14 }
897 \_unravel\_tex\_primitive:nnn { pretolerance } { assign\_int } { 0 }
898 \_unravel\_tex\_primitive:nnn { tolerance } { assign\_int } { 1 }
899 \_unravel\_tex\_primitive:nnn { linepenalty } { assign\_int } { 2 }
900 \_unravel\_tex\_primitive:nnn { hyphenpenalty } { assign\_int } { 3 }
901 \_unravel\_tex\_primitive:nnn { exhyphenpenalty } { assign\_int } { 4 }
902 \_unravel\_tex\_primitive:nnn { clubpenalty } { assign\_int } { 5 }
903 \_unravel\_tex\_primitive:nnn { widowpenalty } { assign\_int } { 6 }
904 \_unravel\_tex\_primitive:nnn { displaywidowpenalty } { assign\_int } { 7 }
905 \_unravel\_tex\_primitive:nnn { brokenpenalty } { assign\_int } { 8 }
906 \_unravel\_tex\_primitive:nnn { binoppenalty } { assign\_int } { 9 }
907 \_unravel\_tex\_primitive:nnn { relpenalty } { assign\_int } { 10 }
908 \_unravel\_tex\_primitive:nnn { predisdisplaypenalty } { assign\_int } { 11 }
909 \_unravel\_tex\_primitive:nnn { postdisplaypenalty } { assign\_int } { 12 }
910 \_unravel\_tex\_primitive:nnn { interlinepenalty } { assign\_int } { 13 }
911 \_unravel\_tex\_primitive:nnn { doublehyphendemerits } { assign\_int } { 14 }
912 \_unravel\_tex\_primitive:nnn { finalhyphendemerits } { assign\_int } { 15 }
913 \_unravel\_tex\_primitive:nnn { adjdemerits } { assign\_int } { 16 }
914 \_unravel\_tex\_primitive:nnn { mag } { assign\_int } { 17 }
915 \_unravel\_tex\_primitive:nnn { delimiterfactor } { assign\_int } { 18 }
916 \_unravel\_tex\_primitive:nnn { looseness } { assign\_int } { 19 }
917 \_unravel\_tex\_primitive:nnn { time } { assign\_int } { 20 }
918 \_unravel\_tex\_primitive:nnn { day } { assign\_int } { 21 }
919 \_unravel\_tex\_primitive:nnn { month } { assign\_int } { 22 }
920 \_unravel\_tex\_primitive:nnn { year } { assign\_int } { 23 }
921 \_unravel\_tex\_primitive:nnn { showboxbreadth } { assign\_int } { 24 }
922 \_unravel\_tex\_primitive:nnn { showboxdepth } { assign\_int } { 25 }

```

```

923 \_unravel_tex_primitive:nnn { hbadness } { assign_int } { 26 }
924 \_unravel_tex_primitive:nnn { vbadness } { assign_int } { 27 }
925 \_unravel_tex_primitive:nnn { pausing } { assign_int } { 28 }
926 \_unravel_tex_primitive:nnn { tracingonline } { assign_int } { 29 }
927 \_unravel_tex_primitive:nnn { tracingmacros } { assign_int } { 30 }
928 \_unravel_tex_primitive:nnn { tracingstats } { assign_int } { 31 }
929 \_unravel_tex_primitive:nnn { tracingparagraphs } { assign_int } { 32 }
930 \_unravel_tex_primitive:nnn { tracingpages } { assign_int } { 33 }
931 \_unravel_tex_primitive:nnn { tracingoutput } { assign_int } { 34 }
932 \_unravel_tex_primitive:nnn { tracinglostchars } { assign_int } { 35 }
933 \_unravel_tex_primitive:nnn { tracingcommands } { assign_int } { 36 }
934 \_unravel_tex_primitive:nnn { tracingrestores } { assign_int } { 37 }
935 \_unravel_tex_primitive:nnn { uchyph } { assign_int } { 38 }
936 \_unravel_tex_primitive:nnn { outputpenalty } { assign_int } { 39 }
937 \_unravel_tex_primitive:nnn { maxdeadcycles } { assign_int } { 40 }
938 \_unravel_tex_primitive:nnn { hangafter } { assign_int } { 41 }
939 \_unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
940 \_unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
941 \_unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
942 \_unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
943 \_unravel_tex_primitive:nnn { defaultthyphenchar } { assign_int } { 46 }
944 \_unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
945 \_unravel_tex_primitive:nnn { endlinesechar } { assign_int } { 48 }
946 \_unravel_tex_primitive:nnn { newlinesechar } { assign_int } { 49 }
947 \_unravel_tex_primitive:nnn { language } { assign_int } { 50 }
948 \_unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
949 \_unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
950 \_unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
951 \_unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
952 \_unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
953 \_unravel_tex_primitive:nnn { pdfcompresslevel } { assign_int } { 56 }
954 \_unravel_tex_primitive:nnn { pdfdecimaldigits } { assign_int } { 57 }
955 \_unravel_tex_primitive:nnn { pdfmovechars } { assign_int } { 58 }
956 \_unravel_tex_primitive:nnn { pdfimageresolution } { assign_int } { 59 }
957 \_unravel_tex_primitive:nnn { pdfpkresolution } { assign_int } { 60 }
958 \_unravel_tex_primitive:nnn { pdfuniqueresname } { assign_int } { 61 }
959 \_unravel_tex_primitive:nnn
960 { pdfoptionalalwaysusepdfpagebox } { assign_int } { 62 }
961 \_unravel_tex_primitive:nnn
962 { pdfoptionpdfinclusionerrorlevel } { assign_int } { 63 }
963 \_unravel_tex_primitive:nnn
964 { pdfoptionpdfminorversion } { assign_int } { 64 }
965 \_unravel_tex_primitive:nnn { pdfminorversion } { assign_int } { 64 }
966 \_unravel_tex_primitive:nnn { pdfforcepagebox } { assign_int } { 65 }
967 \_unravel_tex_primitive:nnn { pdfpagebox } { assign_int } { 66 }
968 \_unravel_tex_primitive:nnn
969 { pdfinclusionerrorlevel } { assign_int } { 67 }
970 \_unravel_tex_primitive:nnn { pdfgamma } { assign_int } { 68 }
971 \_unravel_tex_primitive:nnn { pdfimagegamma } { assign_int } { 69 }
972 \_unravel_tex_primitive:nnn { pdfimagehicolor } { assign_int } { 70 }
973 \_unravel_tex_primitive:nnn { pdfimageapplygamma } { assign_int } { 71 }
974 \_unravel_tex_primitive:nnn { pdfadjustspacing } { assign_int } { 72 }
975 \_unravel_tex_primitive:nnn { pdfprotrudechars } { assign_int } { 73 }
976 \_unravel_tex_primitive:nnn { pdftracingfonts } { assign_int } { 74 }

```

```

977 \_unravel\_tex\_primitive:nnn { pdfobjcompresslevel } { assign\_int } { 75 }
978 \_unravel\_tex\_primitive:nnn
979 { pdfadjustinterwordglue } { assign\_int } { 76 }
980 \_unravel\_tex\_primitive:nnn { pdfprependkern } { assign\_int } { 77 }
981 \_unravel\_tex\_primitive:nnn { pdfappendkern } { assign\_int } { 78 }
982 \_unravel\_tex\_primitive:nnn { pdfgentounicode } { assign\_int } { 79 }
983 \_unravel\_tex\_primitive:nnn { pdfdraftmode } { assign\_int } { 80 }
984 \_unravel\_tex\_primitive:nnn { pdfinclusioncopyfonts } { assign\_int } { 81 }
985 \_unravel\_tex\_primitive:nnn { tracingassigns } { assign\_int } { 82 }
986 \_unravel\_tex\_primitive:nnn { tracinggroups } { assign\_int } { 83 }
987 \_unravel\_tex\_primitive:nnn { tracingifs } { assign\_int } { 84 }
988 \_unravel\_tex\_primitive:nnn { tracingscantokens } { assign\_int } { 85 }
989 \_unravel\_tex\_primitive:nnn { tracingnesting } { assign\_int } { 86 }
990 \_unravel\_tex\_primitive:nnn { predisplaysdirection } { assign\_int } { 87 }
991 \_unravel\_tex\_primitive:nnn { lastlinefit } { assign\_int } { 88 }
992 \_unravel\_tex\_primitive:nnn { savingvdiscards } { assign\_int } { 89 }
993 \_unravel\_tex\_primitive:nnn { savinghyphcodes } { assign\_int } { 90 }
994 \_unravel\_tex\_primitive:nnn { TeXxETstate } { assign\_int } { 91 }
995 \_unravel\_tex\_primitive:nnn { parindent } { assign\_dimen } { 0 }
996 \_unravel\_tex\_primitive:nnn { mathsurround } { assign\_dimen } { 1 }
997 \_unravel\_tex\_primitive:nnn { lineskiplimit } { assign\_dimen } { 2 }
998 \_unravel\_tex\_primitive:nnn { hsize } { assign\_dimen } { 3 }
999 \_unravel\_tex\_primitive:nnn { vsize } { assign\_dimen } { 4 }
1000 \_unravel\_tex\_primitive:nnn { maxdepth } { assign\_dimen } { 5 }
1001 \_unravel\_tex\_primitive:nnn { splitmaxdepth } { assign\_dimen } { 6 }
1002 \_unravel\_tex\_primitive:nnn { boxmaxdepth } { assign\_dimen } { 7 }
1003 \_unravel\_tex\_primitive:nnn { hfuzz } { assign\_dimen } { 8 }
1004 \_unravel\_tex\_primitive:nnn { vfuzz } { assign\_dimen } { 9 }
1005 \_unravel\_tex\_primitive:nnn { delimitershortfall } { assign\_dimen } { 10 }
1006 \_unravel\_tex\_primitive:nnn { nulldelimiterspace } { assign\_dimen } { 11 }
1007 \_unravel\_tex\_primitive:nnn { scriptspace } { assign\_dimen } { 12 }
1008 \_unravel\_tex\_primitive:nnn { predisplaysize } { assign\_dimen } { 13 }
1009 \_unravel\_tex\_primitive:nnn { displaywidth } { assign\_dimen } { 14 }
1010 \_unravel\_tex\_primitive:nnn { displayindent } { assign\_dimen } { 15 }
1011 \_unravel\_tex\_primitive:nnn { overfullrule } { assign\_dimen } { 16 }
1012 \_unravel\_tex\_primitive:nnn { hangindent } { assign\_dimen } { 17 }
1013 \_unravel\_tex\_primitive:nnn { hoffset } { assign\_dimen } { 18 }
1014 \_unravel\_tex\_primitive:nnn { voffset } { assign\_dimen } { 19 }
1015 \_unravel\_tex\_primitive:nnn { emergencystretch } { assign\_dimen } { 20 }
1016 \_unravel\_tex\_primitive:nnn { pdfhorigin } { assign\_dimen } { 21 }
1017 \_unravel\_tex\_primitive:nnn { pdfvorigin } { assign\_dimen } { 22 }
1018 \_unravel\_tex\_primitive:nnn { pdfpagewidth } { assign\_dimen } { 23 }
1019 \_unravel\_tex\_primitive:nnn { pdfpageheight } { assign\_dimen } { 24 }
1020 \_unravel\_tex\_primitive:nnn { pdflinkmargin } { assign\_dimen } { 25 }
1021 \_unravel\_tex\_primitive:nnn { pdfdestmargin } { assign\_dimen } { 26 }
1022 \_unravel\_tex\_primitive:nnn { pdfthreadmargin } { assign\_dimen } { 27 }
1023 \_unravel\_tex\_primitive:nnn { pdffirstlineheight } { assign\_dimen } { 28 }
1024 \_unravel\_tex\_primitive:nnn { pdfastlinedepth } { assign\_dimen } { 29 }
1025 \_unravel\_tex\_primitive:nnn { pdfeachlineheight } { assign\_dimen } { 30 }
1026 \_unravel\_tex\_primitive:nnn { pdfeachlinedepth } { assign\_dimen } { 31 }
1027 \_unravel\_tex\_primitive:nnn { pdfignoredimen } { assign\_dimen } { 32 }
1028 \_unravel\_tex\_primitive:nnn { pdfpxdimen } { assign\_dimen } { 33 }
1029 \_unravel\_tex\_primitive:nnn { lineskip } { assign\_glue } { 0 }
1030 \_unravel\_tex\_primitive:nnn { baselineskip } { assign\_glue } { 1 }

```



```

1031 \_unravel_tex_primitive:nnn { parskip           } { assign_glue } { 2 }
1032 \_unravel_tex_primitive:nnn { abovedisplayskip  } { assign_glue } { 3 }
1033 \_unravel_tex_primitive:nnn { belowdisplayskip  } { assign_glue } { 4 }
1034 \_unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
1035 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1036 \_unravel_tex_primitive:nnn { leftskip          } { assign_glue } { 7 }
1037 \_unravel_tex_primitive:nnn { rightskip         } { assign_glue } { 8 }
1038 \_unravel_tex_primitive:nnn { topskip           } { assign_glue } { 9 }
1039 \_unravel_tex_primitive:nnn { splittopskip      } { assign_glue } { 10 }
1040 \_unravel_tex_primitive:nnn { tabskip           } { assign_glue } { 11 }
1041 \_unravel_tex_primitive:nnn { spaceskip         } { assign_glue } { 12 }
1042 \_unravel_tex_primitive:nnn { xspaceskip        } { assign_glue } { 13 }
1043 \_unravel_tex_primitive:nnn { parfillskip       } { assign_glue } { 14 }
1044 \_unravel_tex_primitive:nnn { thinmuskip        } { assign_mu_glue } { 15 }
1045 \_unravel_tex_primitive:nnn { medmuskip         } { assign_mu_glue } { 16 }
1046 \_unravel_tex_primitive:nnn { thickmuskip       } { assign_mu_glue } { 17 }
1047 \_unravel_tex_primitive:nnn { fontdimen         } { assign_font_dimen } { 0 }
1048 \_unravel_tex_primitive:nnn { hyphenchar        } { assign_font_int } { 0 }
1049 \_unravel_tex_primitive:nnn { skewchar          } { assign_font_int } { 1 }
1050 \_unravel_tex_primitive:nnn { lpcode                } { assign_font_int } { 2 }
1051 \_unravel_tex_primitive:nnn { rpcode                } { assign_font_int } { 3 }
1052 \_unravel_tex_primitive:nnn { efcodes                } { assign_font_int } { 4 }
1053 \_unravel_tex_primitive:nnn { tagcode                } { assign_font_int } { 5 }
1054 \_unravel_tex_primitive:nnn { pdfnoligatures      } { assign_font_int } { 6 }
1055 \_unravel_tex_primitive:nnn { knbscode                } { assign_font_int } { 7 }
1056 \_unravel_tex_primitive:nnn { stbscode                } { assign_font_int } { 8 }
1057 \_unravel_tex_primitive:nnn { shbscode                } { assign_font_int } { 9 }
1058 \_unravel_tex_primitive:nnn { knbccode                } { assign_font_int } { 10 }
1059 \_unravel_tex_primitive:nnn { knaccode                } { assign_font_int } { 11 }
1060 \_unravel_tex_primitive:nnn { spacefactor          } { set_aux } { 102 }
1061 \_unravel_tex_primitive:nnn { prevdepth            } { set_aux } { 1 }
1062 \_unravel_tex_primitive:nnn { prevgraf             } { set_prev_graf } { 0 }
1063 \_unravel_tex_primitive:nnn { pagegoal             } { set_page_dimen } { 0 }
1064 \_unravel_tex_primitive:nnn { pagetotal             } { set_page_dimen } { 1 }
1065 \_unravel_tex_primitive:nnn { pagestretch          } { set_page_dimen } { 2 }
1066 \_unravel_tex_primitive:nnn { pagefilstretch        } { set_page_dimen } { 3 }
1067 \_unravel_tex_primitive:nnn { pagefillstretch       } { set_page_dimen } { 4 }
1068 \_unravel_tex_primitive:nnn { pagefilllstretch     } { set_page_dimen } { 5 }
1069 \_unravel_tex_primitive:nnn { pageshrink           } { set_page_dimen } { 6 }
1070 \_unravel_tex_primitive:nnn { pagedepth            } { set_page_dimen } { 7 }
1071 \_unravel_tex_primitive:nnn { deadcycles           } { set_page_int } { 0 }
1072 \_unravel_tex_primitive:nnn { insertpenalties      } { set_page_int } { 1 }
1073 \_unravel_tex_primitive:nnn { interactionmode      } { set_page_int } { 2 }
1074 \_unravel_tex_primitive:nnn { wd                    } { set_box_dimen } { 1 }
1075 \_unravel_tex_primitive:nnn { dp                    } { set_box_dimen } { 2 }
1076 \_unravel_tex_primitive:nnn { ht                    } { set_box_dimen } { 3 }
1077 \_unravel_tex_primitive:nnn { parshape             } { set_shape } { 0 }
1078 \_unravel_tex_primitive:nnn { interlinepenalties   } { set_shape } { 1 }
1079 \_unravel_tex_primitive:nnn { clubpenalties        } { set_shape } { 2 }
1080 \_unravel_tex_primitive:nnn { widowpenalties        } { set_shape } { 3 }
1081 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
1082 \_unravel_tex_primitive:nnn { catcode              } { def_code } { 0 }
1083 \_unravel_tex_primitive:nnn { lccode              } { def_code } { 256 }
1084 \_unravel_tex_primitive:nnn { uccode              } { def_code } { 512 }

```

```

1085 \_unravel_tex_primitive:nnn { sfcode } { def_code } { 768 }
1086 \_unravel_tex_primitive:nnn { mathcode } { def_code } { 1024 }
1087 \_unravel_tex_primitive:nnn { delcode } { def_code } { 1591 }
1088 \_unravel_tex_primitive:nnn { textfont } { def_family } { -48 }
1089 \_unravel_tex_primitive:nnn { scriptfont } { def_family } { -32 }
1090 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family } { -16 }
1091 \_unravel_tex_primitive:nnn { nullfont } { set_font } { 0 }
1092 \_unravel_tex_primitive:nnn { font } { def_font } { 0 }
1093 \_unravel_tex_primitive:nnn { count } { register } { 1 000 000 }
1094 \_unravel_tex_primitive:nnn { dimen } { register } { 2 000 000 }
1095 \_unravel_tex_primitive:nnn { skip } { register } { 3 000 000 }
1096 \_unravel_tex_primitive:nnn { muskip } { register } { 4 000 000 }
1097 \_unravel_tex_primitive:nnn { advance } { advance } { 0 }
1098 \_unravel_tex_primitive:nnn { multiply } { multiply } { 0 }
1099 \_unravel_tex_primitive:nnn { divide } { divide } { 0 }
1100 \_unravel_tex_primitive:nnn { long } { prefix } { 1 }
1101 \_unravel_tex_primitive:nnn { outer } { prefix } { 2 }
1102 \_unravel_tex_primitive:nnn { global } { prefix } { 4 }
1103 \_unravel_tex_primitive:nnn { protected } { prefix } { 8 }
1104 \_unravel_tex_primitive:nnn { let } { let } { 0 }
1105 \_unravel_tex_primitive:nnn { futurelet } { let } { 1 }
1106 \_unravel_tex_primitive:nnn { chardef } { shorthand_def } { 0 }
1107 \_unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
1108 \_unravel_tex_primitive:nnn { countdef } { shorthand_def } { 2 }
1109 \_unravel_tex_primitive:nnn { dimendef } { shorthand_def } { 3 }
1110 \_unravel_tex_primitive:nnn { skipdef } { shorthand_def } { 4 }
1111 \_unravel_tex_primitive:nnn { muskipdef } { shorthand_def } { 5 }
1112 \_unravel_tex_primitive:nnn { toksdef } { shorthand_def } { 6 }
1113 \_unravel_tex_primitive:nnn { read } { read_to_cs } { 0 }
1114 \_unravel_tex_primitive:nnn { readline } { read_to_cs } { 1 }
1115 \_unravel_tex_primitive:nnn { def } { def } { 0 }
1116 \_unravel_tex_primitive:nnn { gdef } { def } { 1 }
1117 \_unravel_tex_primitive:nnn { edef } { def } { 2 }
1118 \_unravel_tex_primitive:nnn { xdef } { def } { 3 }
1119 \_unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1120 \_unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1121 \_unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1122 \_unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1123 \_unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1124 \_unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1125 \_unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1126 \_unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1127 \_unravel_tex_primitive:nnn { pdfcopyfont } { pdf_copy_font } { 0 }
1128 \_unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1129 \_unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1130 \_unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1131 \_unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1132 \_unravel_tex_primitive:nnn { pdfprimitive } { no_expand } { 1 }
1133 \_unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1134 \_unravel_tex_primitive:nnn { input } { input } { 0 }
1135 \_unravel_tex_primitive:nnn { endinput } { input } { 1 }
1136 \_unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1137 \_unravel_tex_primitive:nnn { if } { if_test } { 0 }
1138 \_unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }

```

```

1139 \_unravel_tex_primitive:nnn { ifnum           } { if_test } { 2 }
1140 \_unravel_tex_primitive:nnn { ifdim           } { if_test } { 3 }
1141 \_unravel_tex_primitive:nnn { ifodd           } { if_test } { 4 }
1142 \_unravel_tex_primitive:nnn { ifvmode        } { if_test } { 5 }
1143 \_unravel_tex_primitive:nnn { ifhmode        } { if_test } { 6 }
1144 \_unravel_tex_primitive:nnn { ifmmode        } { if_test } { 7 }
1145 \_unravel_tex_primitive:nnn { ifinner       } { if_test } { 8 }
1146 \_unravel_tex_primitive:nnn { ifvoid        } { if_test } { 9 }
1147 \_unravel_tex_primitive:nnn { ifhbox         } { if_test } { 10 }
1148 \_unravel_tex_primitive:nnn { ifvbox         } { if_test } { 11 }
1149 \_unravel_tex_primitive:nnn { ifx             } { if_test } { 12 }
1150 \_unravel_tex_primitive:nnn { ifeof           } { if_test } { 13 }
1151 \_unravel_tex_primitive:nnn { iftrue          } { if_test } { 14 }
1152 \_unravel_tex_primitive:nnn { iffalse         } { if_test } { 15 }
1153 \_unravel_tex_primitive:nnn { ifcase          } { if_test } { 16 }
1154 \_unravel_tex_primitive:nnn { ifdefined       } { if_test } { 17 }
1155 \_unravel_tex_primitive:nnn { ifcsname        } { if_test } { 18 }
1156 \_unravel_tex_primitive:nnn { iffontchar       } { if_test } { 19 }
1157 \_unravel_tex_primitive:nnn { ifincsname      } { if_test } { 20 }
1158 \_unravel_tex_primitive:nnn { ifpdfprimitive  } { if_test } { 21 }
1159 \_unravel_tex_primitive:nnn { ifpdfabsnum     } { if_test } { 22 }
1160 \_unravel_tex_primitive:nnn { ifpdfabsdim     } { if_test } { 23 }
1161 \_unravel_tex_primitive:nnn { fi              } { fi_or_else } { 2 }
1162 \_unravel_tex_primitive:nnn { else            } { fi_or_else } { 3 }
1163 \_unravel_tex_primitive:nnn { or              } { fi_or_else } { 4 }
1164 \_unravel_tex_primitive:nnn { csname          } { cs_name } { 0 }
1165 \_unravel_tex_primitive:nnn { number          } { convert } { 0 }
1166 \_unravel_tex_primitive:nnn { romannumeral     } { convert } { 1 }
1167 \_unravel_tex_primitive:nnn { string           } { convert } { 2 }
1168 \_unravel_tex_primitive:nnn { meaning         } { convert } { 3 }
1169 \_unravel_tex_primitive:nnn { fontname         } { convert } { 4 }
1170 \_unravel_tex_primitive:nnn { eTeXrevision    } { convert } { 5 }
1171 \_unravel_tex_primitive:nnn { pdftexrevision  } { convert } { 6 }
1172 \_unravel_tex_primitive:nnn { pdftexbanner    } { convert } { 7 }
1173 \_unravel_tex_primitive:nnn { pdffontname     } { convert } { 8 }
1174 \_unravel_tex_primitive:nnn { pdffontobjnum   } { convert } { 9 }
1175 \_unravel_tex_primitive:nnn { pdffontsize     } { convert } { 10 }
1176 \_unravel_tex_primitive:nnn { pdfpageref      } { convert } { 11 }
1177 \_unravel_tex_primitive:nnn { pdfxformname    } { convert } { 12 }
1178 \_unravel_tex_primitive:nnn { pdfescapestring } { convert } { 13 }
1179 \_unravel_tex_primitive:nnn { pdfescapename   } { convert } { 14 }
1180 \_unravel_tex_primitive:nnn { leftmarginkern  } { convert } { 15 }
1181 \_unravel_tex_primitive:nnn { rightmarginkern } { convert } { 16 }
1182 \_unravel_tex_primitive:nnn { pdfstrcmp       } { convert } { 17 }
1183 \_unravel_tex_primitive:nnn { pdfcolorstackinit } { convert } { 18 }
1184 \_unravel_tex_primitive:nnn { pdfescapehex    } { convert } { 19 }
1185 \_unravel_tex_primitive:nnn { pdfunescapehex  } { convert } { 20 }
1186 \_unravel_tex_primitive:nnn { pdfcreationdate } { convert } { 21 }
1187 \_unravel_tex_primitive:nnn { pdffilemoddate  } { convert } { 22 }
1188 \_unravel_tex_primitive:nnn { pdffilesize    } { convert } { 23 }
1189 \_unravel_tex_primitive:nnn { pdfmdfivesum   } { convert } { 24 }
1190 \_unravel_tex_primitive:nnn { pdffiledump     } { convert } { 25 }
1191 \_unravel_tex_primitive:nnn { pdfmatch        } { convert } { 26 }
1192 \_unravel_tex_primitive:nnn { pdflastmatch    } { convert } { 27 }

```

```

1193 \_unravel\_tex\_primitive:nnn { pdfuniformdeviate } { convert } { 28 }
1194 \_unravel\_tex\_primitive:nnn { pdfnormaldeviate } { convert } { 29 }
1195 \_unravel\_tex\_primitive:nnn { pdfinsertht } { convert } { 30 }
1196 \_unravel\_tex\_primitive:nnn { pdfximagebbox } { convert } { 31 }
1197 \_unravel\_tex\_primitive:nnn { jobname } { convert } { 32 }
1198 \_unravel\_tex\_primitive:nnn { the } { the } { 0 }
1199 \_unravel\_tex\_primitive:nnn { unexpanded } { the } { 1 }
1200 \_unravel\_tex\_primitive:nnn { detokenize } { the } { 5 }
1201 \_unravel\_tex\_primitive:nnn { topmark } { top\_bot\_mark } { 0 }
1202 \_unravel\_tex\_primitive:nnn { firstmark } { top\_bot\_mark } { 1 }
1203 \_unravel\_tex\_primitive:nnn { botmark } { top\_bot\_mark } { 2 }
1204 \_unravel\_tex\_primitive:nnn { splitfirstmark } { top\_bot\_mark } { 3 }
1205 \_unravel\_tex\_primitive:nnn { splitbotmark } { top\_bot\_mark } { 4 }
1206 \_unravel\_tex\_primitive:nnn { topmarks } { top\_bot\_mark } { 5 }
1207 \_unravel\_tex\_primitive:nnn { firstmarks } { top\_bot\_mark } { 6 }
1208 \_unravel\_tex\_primitive:nnn { botmarks } { top\_bot\_mark } { 7 }
1209 \_unravel\_tex\_primitive:nnn { splitfirstmarks } { top\_bot\_mark } { 8 }
1210 \_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.

```

1211 \cs_new_protected_nopar:Npn \_unravel\_get\_next:
1212 {
1213   \_unravel\_input\_if\_empty:TF
1214   { \_unravel\_exit:w }
1215   {
1216     \_unravel\_input\_gpop:N \l\_unravel\_head\_gtl
1217     \gtl\_head\_do:NN \l\_unravel\_head\_gtl \_unravel\_get\_next\_aux:w
1218     \gtl\_if\_tl:NTF \l\_unravel\_head\_gtl
1219     {
1220       \tl\_set:Nx \l\_unravel\_head\_tl
1221       { \gtl\_head:N \l\_unravel\_head\_gtl }
1222     }
1223     { \tl\_clear:N \l\_unravel\_head\_tl }
1224   }
1225 }
1226 \cs_new_protected_nopar:Npn \_unravel\_get\_next\_aux:w
1227 { \cs\_set\_eq:NN \l\_unravel\_head\_token }

```

(End definition for `\__unravel_get_next:` and `\__unravel_get_next_aux:w`.)

`\__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`.

```

1228 \cs_new_protected_nopar:Npn \__unravel_get_token:
1229 {
1230   \__unravel_get_next:
1231   \__unravel_set_cmd:
1232 }

```

(End definition for `\__unravel_get_token:.`)

`\__unravel_set_cmd:` After the call to `\__unravel_get_next:`, we find the command code `\l__unravel_head_cmd_int` and the character code `\l__unravel_head_char_int`, based only on `\l__unravel_head_token`. First set `\l__unravel_head_meaning_tl` from the `\meaning` of the first token. If the corresponding primitive exists, use the information to set the two integers. If the token is expandable, it can either be a macro or be a primitive that we somehow do not know (e.g., an expandable `XYTeX` or `LuaTeX` primitive perhaps). Otherwise, it can be a control sequence or a character.

```

1233 \cs_new_protected_nopar:Npn \__unravel_set_cmd:
1234 {
1235   \__unravel_set_cmd_aux_meaning:
1236   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1237   { }
1238   {
1239     \__unravel_token_if_expandable:NTF \l__unravel_head_token
1240     {
1241       \token_if_macro:NTF \l__unravel_head_token
1242       { \__unravel_set_cmd_aux_macro: }
1243       { \__unravel_set_cmd_aux_unknown: }
1244     }
1245     {
1246       \token_if_cs:NTF \l__unravel_head_token
1247       { \__unravel_set_cmd_aux_cs: }
1248       { \__unravel_set_cmd_aux_char: }
1249     }
1250   }
1251 }

```

(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.

`\__unravel_set_cmd_aux_meaning:w`

```

1252 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_meaning:
1253 {
1254   \tl_set:Nx \l__unravel_head_meaning_tl
1255   {
1256     \exp_after:wN \__unravel_strip_escape:w
1257     \token_to_meaning:N \l__unravel_head_token
1258     \tl_to_str:n { : }

```

```

1259     }
1260     \tl_set:Nx \l__unravel_head_meaning_tl
1261     {
1262         \exp_after:wN \__unravel_set_cmd_aux_meaning:w
1263         \l__unravel_head_meaning_tl \q_stop
1264     }
1265 }
1266 \use:x
1267 {
1268     \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
1269     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1270 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_meaning: and \\_\_unravel\_set\_cmd\_aux\_meaning:w.)

\\_\_unravel\_set\_cmd\_aux\_primitive:nTF Test if there is any information about the given (cleaned-up) \meaning. If there is, use  
\\_\_unravel\_set\_cmd\_aux\_primitive:oTF that as the command and character integers.

```

\__unravel_set_cmd_aux_primitive:nn
1271 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
1272 {
1273     \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1274     {
1275         \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1276         { c__unravel_tex_#1_tl }
1277         #2
1278     }
1279 }
1280 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
1281 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
1282 {
1283     \int_set:Nn \l__unravel_head_cmd_int {#1}
1284     \int_set:Nn \l__unravel_head_char_int {#2}
1285 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_primitive:nTF and \\_\_unravel\_set\_cmd\_aux\_primitive:nn.)

\\_\_unravel\_set\_cmd\_aux\_macro: The token is a macro. There is no need to determine whether the macro is long/outer.

```

1286 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_macro:
1287 {
1288     \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1289     \int_zero:N \l__unravel_head_char_int
1290 }

```

(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.

```

1291 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_unknown:
1292 {
1293     \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1294     \c__unravel_tex_relax_tl
1295     \__unravel_error:nxxxx { unknown-primitive }
1296     { \l__unravel_head_meaning_tl } { } { } { }
1297 }

```

(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.

```

1298 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_cs:
1299 {
1300   \tl_if_in:NoTF \l__unravel_head_meaning_tl
1301     { \tl_to_str:n { elect-font } }
1302     {
1303       \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1304       \c__unravel_tex_nullfont_tl
1305     }
1306     { \__unravel_set_cmd_aux_numeric: }
1307 }

```

(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.

```

1308 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_numeric:
1309 {
1310   \tl_set:Nx \l__unravel_tmpa_tl
1311   {
1312     \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1313     \l__unravel_head_meaning_tl + 0
1314   }
1315   \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1316   \l__unravel_tmpa_tl \q_stop
1317 }
1318 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1319 {
1320   \if_int_compare:w '#1 < 'A \exp_stop_f:
1321     \exp_not:N \q_mark
1322     \exp_after:wN \use_i:nn
1323   \fi:
1324   #1 \__unravel_set_cmd_aux_numeric:N
1325 }
1326 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1327 {
1328   \str_case:nnF {#1}
1329   {
1330     { char }      { \__unravel_set_cmd_aux_given:n { char_given } }
1331     { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1332   }
1333   {
1334     \__unravel_set_cmd_aux_primitive:nTF {#1}

```

```

1335         { }
1336         { \__unravel_set_cmd_aux_unknown: }
1337         \int_add:Nn \l__unravel_head_char_int { 100 000 }
1338     }
1339     \int_add:Nn \l__unravel_head_char_int {#2}
1340 }
1341 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1342 {
1343     \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1344     \int_zero:N \l__unravel_head_char_int
1345 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_numeric: and others.)

\\_\_unravel\_set\_cmd\_aux\_char: At this point, the \meaning token list has been shortened by the code meant to remove the escape character. We thus set it again to the \meaning of the leading token. The command is then the first word (delimited by a space) of the \meaning, followed by \_char, except for category other, where we use other\_char. For the character code, there is a need to expand \\_\_unravel\_token\_to\_char:N before placing ‘.

```

1346 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_char:
1347 {
1348     \tl_set:Nx \l__unravel_head_meaning_tl
1349     { \token_to_meaning:N \l__unravel_head_token }
1350     \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1351     { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1352     \exp_after:wN \__unravel_set_cmd_aux_char:w
1353     \l__unravel_head_meaning_tl \q_stop
1354     \__unravel_exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1355     { ‘ \__unravel_token_to_char:N \l__unravel_head_token }
1356 }
1357 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1358 {
1359     \int_set:Nn \l__unravel_head_cmd_int
1360     { \__unravel_tex_use:n { #1_char } }
1361 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_char: and \\_\_unravel\_set\_cmd\_aux\_char:w.)

## 2.5 Manipulating the input

### 2.5.1 Elementary operations

\\_\_unravel\_input\_to\_str: Map \gtl\_to\_str:c through the input stack.

```

1362 \cs_new_nopar:Npn \__unravel_input_to_str:
1363 {
1364     \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1365     \__unravel_input_to_str_aux:n
1366 }
1367 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1368 { \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.

```

1369 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1370 {
1371   \int_compare:nNnTF \g__unravel_input_int = \c_zero
1372   { \use_i:nn }
1373   {
1374     \gtl_if_empty:cTF
1375     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1376     {
1377       \int_gdecr:N \g__unravel_input_int
1378       \__unravel_input_if_empty:TF
1379     }
1380     {
1381       \__unravel_input_split:
1382       \use_ii:nn
1383     }
1384   }
1385 }

```

(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

```

1386 \cs_new_protected_nopar:Npn \__unravel_input_split:
1387 {
1388   \int_compare:nNnT \g__unravel_input_int = \c_one
1389   {
1390     \exp_args:Nc \__unravel_input_split_aux:N
1391     { g__unravel_input_1_gtl }
1392   }
1393 }
1394 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1395 {
1396   \gtl_if_tl:NT #1
1397   {
1398     \gtl_if_head_is_N_type:NT #1
1399     {
1400       \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1401       \__unravel_exp_args:NNx \use:nn
1402       \__unravel_input_split_auxii:N
1403       { \tl_head:N \l__unravel_input_tmpa_tl }
1404     }
1405   }
1406 }
1407 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1408 {
1409   \token_if_parameter:NF #1
1410   {
1411     \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1412     { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1413     \group_begin:
1414     \cs_set:Npn \__unravel_input_split_auxiii:w

```

```

1415         ##1 \__unravel_input_split_end: { + 1 }
1416         \int_gset:Nn \g__unravel_input_int
1417         { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1418         \group_end:
1419         \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1420         \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1421     }
1422 }
1423 \cs_new_nopar:Npn \__unravel_input_split_end: { }
1424 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1425     #1 \__unravel_input_split_end:
1426     {
1427         \gtl_gclear_new:c
1428         { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1429         \gtl_gset:cn
1430         { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1431         \int_gdecr:N \g__unravel_input_tmpa_int
1432     }

```

(End definition for \\_\_unravel\_input\_split:.)

\\_\_unravel\_input\_gset:n At first, all of the input is in the same gtl.

```

1433 \cs_new_protected_nopar:Npn \__unravel_input_gset:n
1434     {
1435         \int_gzero:N \g__unravel_input_int
1436         \__unravel_back_input:n
1437     }

```

(End definition for \\_\_unravel\_input\_gset:n.)

\\_\_unravel\_input\_get:N

```

1438 \cs_new_protected:Npn \__unravel_input_get:N #1
1439     {
1440         \__unravel_input_if_empty:TF
1441         { \gtl_set:Nn #1 { \q_no_value } }
1442         {
1443             \gtl_get_left:cN
1444             { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1445         }
1446     }

```

(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.

```

1447 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1448     {
1449         \__unravel_input_if_empty:TF
1450         { \gtl_set:Nn #1 { \q_no_value } }
1451         {
1452             \gtl_gpop_left:cN
1453             { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1454         }
1455     }

```

(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.

```

1456 \cs_new_protected_nopar:Npn \_unravel_input_merge:
1457 {
1458   \int_gdecr:N \g\_unravel_input_int
1459   \gtl_gconcat:ccc
1460   { \g\_unravel_input_ \int_use:N \g\_unravel_input_int _gtl }
1461   { \g\_unravel_input_ \int_eval:n { \g\_unravel_input_int + 1 } _gtl }
1462   { \g\_unravel_input_ \int_use:N \g\_unravel_input_int _gtl }
1463   \gtl_gclear:c
1464   { \g\_unravel_input_ \int_eval:n { \g\_unravel_input_int + 1 } _gtl }
1465 }

```

(End definition for \\_unravel\_input\_merge:.)

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

\\_unravel\_input\_gpop\_item\_aux:NN

```

1466 \prg_new_protected_conditional:Npnn \_unravel_input_gpop_item:N #1 { F }
1467 {
1468   \int_compare:nNnTF \g\_unravel_input_int = \c_zero
1469   { \prg_return_false: }
1470   {
1471     \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1472     { \g\_unravel_input_ \int_use:N \g\_unravel_input_int _gtl } #1
1473   }
1474 }
1475 \cs_new_protected:Npn \_unravel_input_gpop_item_aux:NN #1#2
1476 {
1477   \gtl_gpop_left_item:NNTF #1#2
1478   { \prg_return_true: }
1479   {
1480     \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero
1481     { \prg_return_false: }
1482     {
1483       \int_compare:nNnTF \g\_unravel_input_int = \c_one
1484       { \prg_return_false: }
1485       {
1486         \_unravel_input_merge:
1487         \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1488         {
1489           \g\_unravel_input_
1490           \int_use:N \g\_unravel_input_int _gtl
1491         }
1492         #2
1493       }
1494     }
1495   }
1496 }

```

(End definition for `\_unravel_input_gpop_item:N`TF and `\_unravel_input_gpop_item_aux:NN`.)

`\_unravel_input_gpop_tl:N`

```

1497 \cs_new_protected:Npn \_unravel_input_gpop_tl:N #1
1498 { \tl_clear:N #1 \_unravel_input_gpop_tl_aux:N #1 }
1499 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:N #1
1500 {
1501   \int_compare:nNnF \g__unravel_input_int = \c_zero
1502   {
1503     \exp_args:Nc \_unravel_input_gpop_tl_aux:NN
1504     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1505   }
1506 }
1507 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:NN #1#2
1508 {
1509   \gtl_if_tl:NTF #1
1510   {
1511     \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1512     \gtl_gclear:N #1
1513     \int_gdecr:N \g__unravel_input_int
1514     \_unravel_input_gpop_tl_aux:N #2
1515   }
1516   {
1517     \int_compare:nNnTF \g__unravel_input_int > \c_one
1518     { \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero }
1519     { \use_i:nn }
1520     {
1521       \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1522       \gtl_gpop_left_tl:N #1
1523     }
1524     {
1525       \_unravel_input_merge:
1526       \_unravel_input_gpop_tl_aux:N #2
1527     }
1528   }
1529 }

```

(End definition for `\_unravel_input_gpop_tl:N`.)

`\_unravel_back_input:n`  
`\_unravel_back_input:x`

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

```

1530 \cs_new_protected_nopar:Npn \_unravel_back_input:n
1531 {
1532   \int_gincr:N \g__unravel_input_int
1533   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1534   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1535 }
1536 \cs_generate_variant:Nn \_unravel_back_input:n { V , o }
1537 \cs_new_protected:Npn \_unravel_back_input:x
1538 { \_unravel_exp_args:Nx \_unravel_back_input:n }

```

(End definition for `\_unravel_back_input:n`.)

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

```

1539 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1540 {
1541   \gtl_if_tl:NTF #1
1542   { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1543   {
1544     \gtl_gconcat:cNc
1545     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1546     #1
1547     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1548   }
1549 }
```

(End definition for `\__unravel_back_input_gtl:N`.)

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

```

1550 \cs_new_protected_nopar:Npn \__unravel_back_input:
1551 { \__unravel_back_input_gtl:N \l__unravel_head_gtl }
```

(End definition for `\__unravel_back_input:.`)

`\__unravel_back_input_tl_o:` Insert the `\l__unravel_head_tl` (may or may not be the last token read) back into the input stream, after expanding it once. Then print some diagnostic information.

```

1552 \cs_new_protected_nopar:Npn \__unravel_back_input_tl_o:
1553 {
1554   \tl_set:Nx \l__unravel_tmpa_tl
1555   { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1556   \__unravel_back_input:V \l__unravel_tmpa_tl
1557   \__unravel_print_done:x
1558   { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1559 }
```

(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.

```

1560 \cs_new_protected_nopar:Npn \__unravel_insert_relax:
1561 {
1562   \__unravel_back_input:
1563   \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1564   \__unravel_back_input:
1565   \__unravel_print_action:
1566 }
```

(End definition for `\__unravel_insert_relax:.`)

`\__unravel_insert_group_begin_error:`

```

1567 \cs_new_protected_nopar:Npn \__unravel_insert_group_begin_error:
1568 {
1569   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
1570   \__unravel_back_input:
```

```

1571 \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1572 \__unravel_back_input:
1573 \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
1574 \__unravel_print_action:
1575 }

```

(End definition for \\_\_unravel\_insert\_group\_begin\_error:.)

\\_\_unravel\_insert\_dollar\_error:

```

1576 \cs_new_protected_nopar:Npn \__unravel_insert_dollar_error:
1577 {
1578   \__unravel_back_input:
1579   \__unravel_back_input:n { $ } % $
1580   \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
1581   \__unravel_print_action:
1582 }

```

(End definition for \\_\_unravel\_insert\_dollar\_error:.)

### 2.5.3 Macro calls

```

\__unravel_macro_prefix:N
\__unravel_macro_parameter:N
\__unravel_macro_replacement:N
1583 \use:x
1584 {
1585   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
1586   {
1587     \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
1588     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1589     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1590     \exp_not:N \q_stop
1591   }
1592   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
1593   \exp_not:n { #1 } \tl_to_str:n { : } \exp_not:n { #2 -> }
1594   \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1595   { \exp_not:n { #4 #6 { #1 } { #2 } { #3 } } }
1596 }
1597 \cs_new:Npn \__unravel_macro_prefix:N #1
1598 { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1599 \cs_new:Npn \__unravel_macro_parameter:N #1
1600 { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1601 \cs_new:Npn \__unravel_macro_replacement:N #1
1602 { \__unravel_macro_split_do:NN #1 \use_iii:nnn }

```

(End definition for \\_\_unravel\_macro\_prefix:N, \\_\_unravel\_macro\_parameter:N, and \\_\_unravel\_macro\_replacement:N.)

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

```

\__unravel_macro_call_safe:
\__unravel_macro_call_quick:
\__unravel_macro_call_quick_loop:NNN
\__unravel_macro_call_quick_runaway:Nw
1603 \cs_new_protected_nopar:Npn \__unravel_macro_call:
1604 {
1605   \bool_if:NTF \g__unravel_speedup_macros_bool
1606   {

```

```

1607         \tl_set:Nx \l__unravel_tmpa_tl
1608         {~ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1609         \tl_if_in:NVTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1610         { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1611     }
1612     { \__unravel_macro_call_safe: }
1613     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1614     \__unravel_print_done:x { \g__unravel_action_text_str }
1615 }
1616 \cs_new_protected_nopar:Npn \__unravel_macro_call_safe:
1617 {
1618     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1619     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1620 }
1621 \cs_new_protected_nopar:Npn \__unravel_macro_call_quick:
1622 {
1623     \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1624     { ? \use_none_delimit_by_q_stop:w } \q_stop
1625 }
1626 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1627 {
1628     \use_none:n #2
1629     \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1630     { \__unravel_macro_call_quick_runaway:Nw #3 }
1631     \tl_put_right:Nx \l__unravel_head_tl
1632     { { \exp_not:V \l__unravel_tmpa_tl } }
1633     \__unravel_macro_call_quick_loop:NNN
1634     #3
1635 }
1636 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1637 {
1638     \__unravel_error:nxxxx { runaway-macro-parameter }
1639     { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1640 }

```

(End definition for `\__unravel_macro_call:` and others.)

## 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<sub>E</sub>X's structure, distinguishing macros from other commands (not quite sure why).

```

1641 \cs_new_protected_nopar:Npn \__unravel_expand:
1642 {
1643     \__unravel_set_action_text:
1644     \bool_if:NT \g__unravel_internal_debug_bool
1645     {
1646         \__unravel_set_cmd:
1647         \__unravel_exp_args:Nx \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_in
1648     }
1649     \token_if_macro:NTF \l__unravel_head_token
1650     { \__unravel_macro_call: }
1651     { \__unravel_expand_nonmacro: }
1652 }

```

(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.

```

1653 \cs_new_protected_nopar:Npn \_unravel_expand_nonmacro:
1654 {
1655   \_unravel_set_cmd_aux_meaning:
1656   \_unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1657   {
1658     \cs_if_exist_use:cF
1659     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1660     { \_unravel_error:nxxxx { internal } { expandable } { } { } { } }
1661   }
1662   {
1663     \_unravel_error:nxxxx { unknown-primitive }
1664     { \l__unravel_head_meaning_tl } { } { } { } { }
1665     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
1666     \_unravel_print_action:
1667   }
1668 }

```

(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.

```

1669 \cs_new_protected_nopar:Npn \_unravel_get_x_next:
1670 {
1671   \_unravel_get_next:
1672   \_unravel_token_if_expandable:NT \l__unravel_head_token
1673   {
1674     \_unravel_expand:
1675     \_unravel_get_x_next:
1676   }
1677 }

```

(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.

```

1678 \cs_new_protected_nopar:Npn \_unravel_get_x_or_protected:
1679 {
1680   \_unravel_get_next:
1681   \_unravel_token_if_protected:NF \l__unravel_head_token
1682   {
1683     \_unravel_expand:
1684     \_unravel_get_x_or_protected:
1685   }
1686 }

```

(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.

```

1687 \cs_new_protected_nopar:Npn \__unravel_get_x_non_blank:
1688 {
1689   \__unravel_get_x_next:
1690   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1691     { \__unravel_get_x_non_blank: }
1692 }
```

*(End definition for \\_\_unravel\_get\_x\_non\_blank:.)*

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

```

1693 \cs_new_protected_nopar:Npn \__unravel_get_x_non_relax:
1694 {
1695   \__unravel_get_x_next:
1696   \token_if_eq_meaning:NNT \l__unravel_head_token \scan_stop:
1697     { \__unravel_get_x_non_relax: }
1698   {
1699     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1700     { \__unravel_get_x_non_relax: }
1701   }
1702 }
```

*(End definition for \\_\_unravel\_get\_x\_non\_relax:.)*

`\__unravel_skip_optional_space:`

```

1703 \cs_new_protected_nopar:Npn \__unravel_skip_optional_space:
1704 {
1705   \__unravel_get_x_next:
1706   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1707     { \__unravel_back_input: }
1708 }
```

*(End definition for \\_\_unravel\_skip\_optional\_space:.)*

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

```

1709 \cs_new_protected_nopar:Npn \__unravel_scan_optional_equals:
1710 {
1711   \__unravel_get_x_non_blank:
1712   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1713     { \__unravel_prev_input:n { = } }
1714     {
1715       \__unravel_prev_input_silent:n { = }
1716       \__unravel_back_input:
1717     }
1718 }
```

*(End definition for \\_\_unravel\_scan\_optional\_equals:.)*

`\__unravel_scan_left_brace:` The presence of `\relax` is allowed before a begin-group token. If there is no begin-group token, insert one, produce an error, and scan that begin-group using `\__unravel_get_next:`.

```

1719 \cs_new_protected_nopar:Npn \__unravel_scan_left_brace:
1720 {
1721   \__unravel_get_x_non_relax:
1722   \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
1723   {
1724     \__unravel_insert_group_begin_error:
1725     \__unravel_get_next:
1726   }
1727 }

```

(End definition for `\__unravel_scan_left_brace:`.)

`\__unravel_scan_keyword:n` The details of how T<sub>E</sub>X looks for keywords are quite tricky to get right, in particular with respect to expansion, case-insensitivity, and spaces. We get rid of the case issue by requiring the keyword to be given in both cases, intertwined: for instance, `\__unravel_scan_keyword:n { pPtT }`. Then loop through pairs of letters (which should be matching lowercase and uppercase letters). The looping auxiliary takes three arguments, the first of which is a boolean, `true` if spaces are allowed (no letter of the keyword has been found yet). At each iteration, get a token, with expansion, and test whether it is a non-active character equal (in character code) to either letter of the pair: this happens if the token is not “definable” (neither a control sequence nor an active character) and it has the right string representation. . . well, it could also be doubled (macro parameter character), hence we look at the first character only; spaces become an empty string, but this works out because no keyword contains a space. So, at each iteration, if the token is the correct non-active character, add it to the previous-input sequence (as a generalized token list since keywords may match begin-group or end-group characters), and otherwise break with `\__unravel_scan_keyword_false:w`, unless we are still at the beginning of the keyword and the token is a space. When the loop reaches the end of the keyword letter pairs, complain if there were an odd number of letters, and otherwise conclude the loop with `\__unravel_scan_keyword_true:`, which stores the keyword, converted to a string. Note that T<sub>E</sub>X’s skipping of leading spaces here must be intertwined with the search for keyword, as is shown by the (plain T<sub>E</sub>X) example

```

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

1728 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1729 { \__unravel_scan_keyword:nTF {#1} { } { } }
1730 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1731 { T , F , TF }
1732 {
1733   \__unravel_prev_input_gpush_gtl:
1734   \__unravel_scan_keyword_loop:NNN \c_true_bool
1735   #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1736 }
1737 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1738 {
1739   \quark_if_recursion_tail_stop_do:nn {#2}
1740   { \__unravel_scan_keyword_true: }

```

```

1741 \quark_if_recursion_tail_stop_do:nn {#3}
1742 { \__unravel_error:nxxxx { internal } { odd-keyword-length } { } { } { } }
1743 \__unravel_get_x_next:
1744 \__unravel_scan_keyword_test:NNTF #2#3
1745 {
1746   \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1747   \__unravel_scan_keyword_loop:NNN \c_false_bool
1748 }
1749 {
1750   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_space_token
1751   { \__unravel_scan_keyword_false:w }
1752   \bool_if:NF #1
1753   { \__unravel_scan_keyword_false:w }
1754   \__unravel_scan_keyword_loop:NNN #1#2#3
1755 }
1756 }
1757 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1758 { TF }
1759 {
1760   \__unravel_gtl_if_head_is_definable:NNTF \l__unravel_head_gtl
1761   { \prg_return_false: }
1762   {
1763     \str_if_eq_x:nnTF
1764     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1765     { \prg_return_true: }
1766     {
1767       \str_if_eq_x:nnTF
1768       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
1769       { \prg_return_true: }
1770       { \prg_return_false: }
1771     }
1772   }
1773 }
1774 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_true:
1775 {
1776   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1777   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1778   \prg_return_true:
1779 }
1780 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_false:w
1781 #1 \q_recursion_stop
1782 {
1783   \__unravel_back_input:
1784   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
1785   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1786   \prg_return_false:
1787 }

```

(End definition for \\_\_unravel\_scan\_keyword:n and others.)

\\_\_unravel\_scan\_to: Used when to is mandatory: after \read or \readline and after \vsplit.

```

1788 \cs_new_protected_nopar:Npn \__unravel_scan_to:
1789 {
1790   \__unravel_scan_keyword:nF { tTo0 }

```

```

1791     {
1792       \__unravel_error:nnnnn { missing-to } { } { } { } { }
1793       \__unravel_prev_input:n { to }
1794     }
1795   }

```

(End definition for \\_\_unravel\_scan\_to:.)

\\_\_unravel\_scan\_font\_ident: Find a font identifier.

```

1796 \cs_new_protected_nopar:Npn \__unravel_scan_font_ident:
1797 {
1798   \__unravel_get_x_non_blank:
1799   \__unravel_set_cmd:
1800   \int_case:nnF \l__unravel_head_cmd_int
1801   {
1802     { \__unravel_tex_use:n { def_font } }
1803     { \__unravel_prev_input:V \l__unravel_head_tl }
1804     { \__unravel_tex_use:n { letterspace_font } }
1805     { \__unravel_prev_input:V \l__unravel_head_tl }
1806     { \__unravel_tex_use:n { pdf_copy_font } }
1807     { \__unravel_prev_input:V \l__unravel_head_tl }
1808     { \__unravel_tex_use:n { set_font } }
1809     { \__unravel_prev_input:V \l__unravel_head_tl }
1810     { \__unravel_tex_use:n { def_family } }
1811     {
1812       \__unravel_prev_input:V \l__unravel_head_tl
1813       \__unravel_scan_int:
1814     }
1815   }
1816   {
1817     \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
1818     \__unravel_back_input:
1819     \__unravel_prev_input:n { \__unravel_nullfont: }
1820   }
1821 }

```

(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).

```

1822 \cs_new_protected_nopar:Npn \__unravel_scan_font_int:
1823 {
1824   \int_case:nnF \l__unravel_head_char_int
1825   {
1826     { 0 } { \__unravel_scan_font_ident: }
1827     { 1 } { \__unravel_scan_font_ident: }
1828     { 6 } { \__unravel_scan_font_ident: }
1829   }
1830   { \__unravel_scan_font_ident: \__unravel_scan_int: }
1831 }

```

(End definition for \\_\_unravel\_scan\_font\_int:.)

\\_\_unravel\_scan\_font\_dimen: Find operands for \fontdimen.

```

1832 \cs_new_protected_nopar:Npn \__unravel_scan_font_dimen:
1833 {

```

```

1834     \__unravel_scan_int:
1835     \__unravel_scan_font_ident:
1836 }

```

(End definition for \\_\_unravel\_scan\_font\_dimen:.)

```

\__unravel_scan_something_internal:n
\__unravel_scan_something_aux:nwn

```

Receives an (explicit) “level” argument:

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

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

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

If the argument (`#3` in the auxiliary) is  $< 4$  but the level that will be produced (`#1` in the auxiliary) is  $\geq 4$  (that is, 4, 5, or 8) complain about a missing number and insert a zero dimension, to get exactly T<sub>E</sub>X’s error recovery. If the level produced is 8, complain that `\the` cannot do this.

Otherwise, scan the arguments (in a new input level). If both the argument and the level produced are  $< 4$ , then get the value with `\__unravel_thing_use_get:nnNN` which downgrades from glue to dimension to integer and produces the `incompatible-units` error if needed. The only remaining case is that the argument is 5 (since 4 is never used) and the level produced is that or less: then the value found is used with `\__unravel_the:w`.

Finally, tell the user the tokens that have been found (if there was a single token, its meaning as well) and their value. Use `=>` rather than `=` because the value displayed is the value used, not the actual value (this matters in constructions such as `\parindent=\parskip` where a skip or a dimen is downgraded to a dimen or an int, or when there was an error).

```

1837 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1838 {
1839     \__unravel_set_cmd:
1840     \__unravel_set_action_text:
1841     \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1842     \exp_after:wN \__unravel_scan_something_aux:nwn
1843     \l__unravel_tmpa_tl \q_stop {#1}
1844 }
1845 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1846 {
1847     \int_compare:nT { #3 < 4 <= #1 }
1848     {

```

```

1849     \__unravel_back_input:
1850     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
1851     \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
1852     \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
1853     \__unravel_break:w
1854 }
1855 \int_compare:nNnT {#1} = { 8 }
1856 {
1857     \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
1858     \__unravel_scan_something_internal_auxii:Vn \c_zero { 0 }
1859     \__unravel_break:w
1860 }
1861 \tl_if_empty:nF {#2}
1862 {
1863     \__unravel_prev_input_gpush:N \l__unravel_head_tl
1864     \__unravel_print_action:
1865     #2
1866     \__unravel_prev_input_gpop:N \l__unravel_head_tl
1867 }
1868 \int_compare:nNnTF {#3} < { 4 }
1869 { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
1870 { \tl_set:Nx \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
1871 \__unravel_scan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
1872 \__unravel_break_point:
1873 \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
1874 }
1875 \cs_new_protected:Npn \__unravel_scan_something_internal_auxii:nn #1#2
1876 {
1877     \__unravel_prev_input_silent:n {#1}
1878     \__unravel_set_action_text:
1879     \__unravel_set_action_text:x
1880     { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
1881     \int_gset:Nn \g__unravel_val_level_int {#2}
1882 }
1883 \cs_generate_variant:Nn \__unravel_scan_something_internal_auxii:nn { V }

```

(End definition for \\_\_unravel\_scan\_something\_internal:n and \\_\_unravel\_scan\_something\_aux:nwn.)

\\_\_unravel\_thing\_case: This expands to a digit (the level generated by whatever token is the current head),  
 \\_\_unravel\_thing\_last\_item: followed by some code to fetch necessary operands. In most cases, this can be done by  
 \\_\_unravel\_thing\_register: simply looking at the cmd integer, but for last\_item, set\_aux and register, the level  
 of the token depends on the char integer. When the token is not allowed after \the (or  
 at any other position where \\_\_unravel\_scan\_something\_internal:n is called), the  
 resulting level is 8, large enough so that the main function knows it is forbidden.

```

1884 \cs_new_nopar:Npn \__unravel_thing_case:
1885 {
1886     \int_case:nnF \l__unravel_head_cmd_int
1887     {
1888         { 68 } { 0 } % char_given
1889         { 69 } { 0 } % math_given
1890         { 70 } { \__unravel_thing_last_item: } % last_item
1891         { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
1892         { 72 } { 5 } % assign_toks
1893         { 73 } { 0 } % assign_int

```

```

1894 { 74 } { 1 } % assign_dimen
1895 { 75 } { 2 } % assign_glue
1896 { 76 } { 3 } % assign_mu_glue
1897 { 77 } { 1 \_unravel_scan_font_dimen: } % assign_font_dimen
1898 { 78 } { 0 \_unravel_scan_font_int: } % assign_font_int
1899 { 79 } { \_unravel_thing_set_aux: } % set_aux
1900 { 80 } { 0 } % set_prev_graf
1901 { 81 } { 1 } % set_page_dimen
1902 { 82 } { 0 } % set_page_int
1903 { 83 } { 1 \_unravel_scan_int: } % set_box_dimen
1904 { 84 } { 0 \_unravel_scan_int: } % set_shape
1905 { 85 } { 0 \_unravel_scan_int: } % def_code
1906 { 86 } { 4 \_unravel_scan_int: } % def_family
1907 { 87 } { 4 } % set_font
1908 { 88 } { 4 } % def_font
1909 { 89 } { \_unravel_thing_register: } % register
1910 {101 } { 4 } % letterspace_font
1911 {102 } { 4 } % pdf_copy_font
1912 }
1913 { 8 }
1914 }
1915 \cs_new_nopar:Npn \_unravel_thing_set_aux:
1916 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
1917 \cs_new_nopar:Npn \_unravel_thing_last_item:
1918 {
1919 \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
1920 {
1921 \int_case:nnF \l__unravel_head_char_int
1922 {
1923 { 1 } { 1 } % lastkern
1924 { 2 } { 2 } % lastskip
1925 }
1926 { 0 } % other integer parameters
1927 }
1928 {
1929 \int_case:nnF \l__unravel_head_char_int
1930 {
1931 { 26 } { 0 \_unravel_scan_normal_glue: } % gluestretchorder
1932 { 27 } { 0 \_unravel_scan_normal_glue: } % glueshrinkorder
1933 { 28 } % fontcharwd
1934 { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1935 { 29 } % fontcharht
1936 { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1937 { 30 } % fontchardp
1938 { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1939 { 31 } % fontcharic
1940 { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1941 { 32 } { 1 \_unravel_scan_int: } % parshape length
1942 { 33 } { 1 \_unravel_scan_int: } % parshape indent
1943 { 34 } { 1 \_unravel_scan_int: } % parshape dimen
1944 { 35 } { 1 \_unravel_scan_normal_glue: } % gluestretch
1945 { 36 } { 1 \_unravel_scan_normal_glue: } % glueshrink
1946 { 37 } { 2 \_unravel_scan_mu_glue: } % mutoglu
1947 { 38 } { 3 \_unravel_scan_normal_glue: } % gluetomu

```

```

1948         { 39 } % numexpr
1949         { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
1950     { 40 } % dimexpr
1951     { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
1952     { 41 } % glueexpr
1953     { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
1954     { 42 } % muexpr
1955     { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
1956     }
1957     { }
1958 }
1959 }
1960 \cs_new_nopar:Npn \__unravel_thing_register:
1961 {
1962     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
1963     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = \c_zero
1964     { \__unravel_scan_int: }
1965 }

```

(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.

```

1966 \cs_new_protected:Npn \__unravel_scan_toks_register:
1967 {
1968     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
1969     { \__unravel_scan_int: }
1970 }

```

(End definition for `\__unravel_scan_toks_register:`.)

`\__unravel_thing_use_get:nnNN` Given a level found #1 and a target level #2 (both in  $[0, 3]$ ), turn the token list #3 into the desired level or less, and store the result in #4.

```

1971 \cs_new_protected:Npn \__unravel_thing_use_get:nnNN #1#2#3#4
1972 {
1973     \int_compare:nNnTF {#2} < { 3 }
1974     {
1975         \int_compare:nNnT {#1} = { 3 }
1976         { \__unravel_tex_error:nV { incompatible-units } #3 }
1977         \tl_set:Nx #4
1978         {
1979             \int_case:nn { \int_min:nn {#1} {#2} }
1980             {
1981                 { 0 } \int_eval:n
1982                 { 1 } \dim_eval:n
1983                 { 2 } \skip_eval:n
1984             }
1985             { \int_compare:nNnT {#1} = { 3 } \etex_mutogluue:D #3 }
1986         }
1987     }
1988     {
1989         \int_case:nnF {#1}
1990         {
1991             { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
1992             { 3 } { \tl_set:Nx #4 { \muskip_eval:n {#3} } }

```



```

1993     }
1994     {
1995         \_unravel_tex_error:nV { incompatible-units } #3
1996         \tl_set:Nx #4 { \muskip_eval:n { \etex_gluetomu:D #3 } }
1997     }
1998 }
1999 }

```

(End definition for \\_unravel\_thing\_use\_get:nnNN.)

```

\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
2000 \cs_new_protected:Npn \_unravel_scan_expr:N #1
2001 { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
2002 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
2003 {
2004     \_unravel_get_x_non_blank:
2005     \_unravel_scan_factor:N #1
2006     \_unravel_scan_expr_op:NN #1#2
2007 }
2008 \cs_new_protected:Npn \_unravel_scan_expr_op:NN #1#2
2009 {
2010     \_unravel_get_x_non_blank:
2011     \tl_case:NnF \l__unravel_head_tl
2012     {
2013         \c__unravel_plus_tl
2014         {
2015             \_unravel_prev_input:V \l__unravel_head_tl
2016             \_unravel_scan_expr_aux:NN #1#2
2017         }
2018         \c__unravel_minus_tl
2019         {
2020             \_unravel_prev_input:V \l__unravel_head_tl
2021             \_unravel_scan_expr_aux:NN #1#2
2022         }
2023         \c__unravel_times_tl
2024         {
2025             \_unravel_prev_input:V \l__unravel_head_tl
2026             \_unravel_get_x_non_blank:
2027             \_unravel_scan_factor:N \_unravel_scan_int:
2028             \_unravel_scan_expr_op:NN #1#2
2029         }
2030         \c__unravel_over_tl
2031         {
2032             \_unravel_prev_input:V \l__unravel_head_tl
2033             \_unravel_get_x_non_blank:
2034             \_unravel_scan_factor:N \_unravel_scan_int:
2035             \_unravel_scan_expr_op:NN #1#2
2036         }
2037         \c__unravel_rp_tl
2038         {
2039             \bool_if:NTF #2
2040             { \_unravel_prev_input:V \l__unravel_head_tl }
2041             { \_unravel_back_input: }
2042         }

```

```

2043     }
2044     {
2045         \bool_if:NTF #2
2046         {
2047             \__unravel_error:nnnnn { missing-rparen } { } { } { } { } { }
2048             \__unravel_back_input:
2049             \__unravel_prev_input:V \c__unravel_rp_tl
2050         }
2051         {
2052             \token_if_eq_meaning:NMF \l__unravel_head_token \scan_stop:
2053             { \__unravel_back_input: }
2054         }
2055     }
2056 }
2057 \cs_new_protected:Npn \__unravel_scan_factor:N #1
2058 {
2059     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2060     {
2061         \__unravel_prev_input:V \l__unravel_head_tl
2062         \__unravel_scan_expr_aux:NN #1 \c_true_bool
2063     }
2064     {
2065         \__unravel_back_input:
2066         #1
2067     }
2068 }

```

(End definition for \\_\_unravel\_scan\_expr:N, \\_\_unravel\_scan\_expr\_aux:NN, and \\_\_unravel\_scan\_factor:N.)

\\_\_unravel\_scan\_signs: Skips blanks, scans signs, and places them to the right of the last item of \\_\_unravel\_prev\_input:n.

```

2069 \cs_new_protected_nopar:Npn \__unravel_scan_signs:
2070 {
2071     \__unravel_get_x_non_blank:
2072     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
2073     {
2074         \__unravel_prev_input:V \l__unravel_head_tl
2075         \__unravel_scan_signs:
2076     }
2077     {
2078         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
2079         {
2080             \__unravel_prev_input:V \l__unravel_head_tl
2081             \__unravel_scan_signs:
2082         }
2083     }
2084 }

```

(End definition for \\_\_unravel\_scan\_signs:.)

```

\__unravel_scan_int:
\__unravel_scan_int_char: 2085 \cs_new_protected_nopar:Npn \__unravel_scan_int:
\__unravel_scan_int_lq: 2086 {
\__unravel_scan_int_explicit:n 2087     \__unravel_scan_signs:

```

```

2088     \__unravel_set_cmd:
2089     \__unravel_cmd_if_internal:TF
2090     { \__unravel_scan_something_internal:n { 0 } }
2091     { \__unravel_scan_int_char: }
2092 }
2093 \cs_new_protected_nopar:Npn \__unravel_scan_int_char:
2094 {
2095     \tl_case:NnF \l__unravel_head_tl
2096     {
2097         \c__unravel_lq_tl { \__unravel_scan_int_lq: }
2098         \c__unravel_rq_tl
2099         {
2100             \__unravel_prev_input:V \l__unravel_head_tl
2101             \__unravel_get_x_next:
2102             \__unravel_scan_int_explicit:Nn \c_false_bool { ' }
2103         }
2104         \c__unravel_dq_tl
2105         {
2106             \__unravel_prev_input:V \l__unravel_head_tl
2107             \__unravel_get_x_next:
2108             \__unravel_scan_int_explicit:Nn \c_false_bool { " }
2109         }
2110     }
2111     { \__unravel_scan_int_explicit:Nn \c_false_bool { } }
2112 }
2113 \cs_new_protected_nopar:Npn \__unravel_scan_int_lq:
2114 {
2115     \__unravel_get_next:
2116     \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2117     {
2118         \tl_set:Nx \l__unravel_head_tl
2119         { \__unravel_token_to_char:N \l__unravel_head_token }
2120     }
2121     \tl_set:Nx \l__unravel_tmpa_tl
2122     { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
2123     \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2124     \__unravel_print_action:x
2125     { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2126     \__unravel_skip_optional_space:
2127 }
2128 \cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
2129 {
2130     \if_int_compare:w \c_one
2131     < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2132     \exp_after:wN \use_i:nn
2133     \else:
2134     \exp_after:wN \use_ii:nn
2135     \fi:
2136     {
2137         \__unravel_prev_input:V \l__unravel_head_tl
2138         \__unravel_get_x_next:
2139         \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
2140     }
2141 }

```

```

2142 \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2143 { \__unravel_back_input: }
2144 \bool_if:NF #1
2145 {
2146   \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2147   \__unravel_prev_input:n { 0 }
2148 }
2149 }
2150 }

```

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

`\__unravel_scan_normal_dimen:`

```

2151 \cs_new_protected_nopar:Npn \__unravel_scan_normal_dimen:
2152 { \__unravel_scan_dimen:nN { 2 } \c_false_bool }

```

(End definition for `\__unravel_scan_normal_dimen:.`)

`\__unravel_scan_dimen:nN`

The first argument is 2 if the unit may not be mu and 3 if the unit must be mu (or fil). The second argument is `\c_true_bool` if fil, fill, filll are permitted, and is otherwise **false**. These arguments are similar to those of TeX's own `scan_dimen` procedure, in which mu is `bool(#1=3)` and inf is `#2`. The third argument of this procedure is omitted here, as the corresponding shortcut is provided as a separate function, `\__unravel_scan_dim_unit:nN`.

```

2153 \cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
2154 {
2155   \__unravel_scan_signs:
2156   \__unravel_prev_input_gpush:
2157   \__unravel_set_cmd:
2158   \__unravel_cmd_if_internal:TF
2159   {
2160     \int_compare:nNnTF {#1} = { 3 }
2161     { \__unravel_scan_something_internal:n { 3 } }
2162     { \__unravel_scan_something_internal:n { 1 } }
2163     \int_compare:nNnT \g__unravel_val_level_int = { 0 }
2164     { \__unravel_scan_dim_unit:nN {#1} #2 }
2165   }
2166   { \__unravel_scan_dimen_char:nN {#1} #2 }
2167   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2168   \__unravel_prev_input_silent:V \l__unravel_head_tl
2169 }
2170 \cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
2171 {
2172   \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2173   { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2174   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
2175   {
2176     \__unravel_prev_input:n { . }
2177     \__unravel_scan_decimal_loop:
2178   }
2179   {
2180     \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl
2181     {
2182       \__unravel_back_input:

```

```

2183         \__unravel_scan_int:
2184         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
2185         { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
2186         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
2187         {
2188             \__unravel_input_gpop:N \l__unravel_tmpb_gtl
2189             \__unravel_prev_input:n { . }
2190             \__unravel_scan_decimal_loop:
2191         }
2192     }
2193     {
2194         \__unravel_back_input:
2195         \__unravel_scan_int:
2196     }
2197 }
2198 \__unravel_scan_dim_unit:nN {#1} #2
2199 }
2200 \cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
2201 {
2202     \bool_if:NT #2
2203     {
2204         \__unravel_scan_keyword:nT { fFiIlL }
2205         {
2206             \__unravel_scan_inf_unit_loop:
2207             \__unravel_break:w
2208         }
2209     }
2210     \__unravel_get_x_non_blank:
2211     \__unravel_set_cmd:
2212     \__unravel_cmd_if_internal:TF
2213     {
2214         \__unravel_prev_input_gpush:
2215         \__unravel_scan_something_internal:n {#1}
2216         \__unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2217         \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2218         \__unravel_break:w
2219     }
2220     { \__unravel_back_input: }
2221     \int_compare:nNnT {#1} = { 3 }
2222     {
2223         \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
2224         \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
2225         \__unravel_prev_input:n { mu }
2226         \__unravel_break:w
2227     }
2228     \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
2229     \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
2230     \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
2231     \__unravel_scan_keyword:nT { tTrRuUeE }
2232     { \__unravel_prepare_mag: }
2233     \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
2234     \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
2235     \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
2236     \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }

```

```

2237 \_unravel\_scan\_keyword:nT { mMM } { \_unravel\_break:w }
2238 \_unravel\_scan\_keyword:nT { bBpP } { \_unravel\_break:w }
2239 \_unravel\_scan\_keyword:nT { dDdD } { \_unravel\_break:w }
2240 \_unravel\_scan\_keyword:nT { cCcC } { \_unravel\_break:w }
2241 \_unravel\_scan\_keyword:nT { nNdD } { \_unravel\_break:w }
2242 \_unravel\_scan\_keyword:nT { nNcC } { \_unravel\_break:w }
2243 \_unravel\_scan\_keyword:nT { sSpP } { \_unravel\_break:w }
2244 \_unravel\_tex\_error:nV { missing-pt } \l\_unravel\_head\_tl
2245 \_unravel\_prev\_input:n { pt }
2246 \_unravel\_break\_point:
2247 }
2248 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_inf\_unit\_loop:
2249 { \_unravel\_scan\_keyword:nT { lL } { \_unravel\_scan\_inf\_unit\_loop: } }
2250 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_decimal\_loop:
2251 {
2252 \_unravel\_get\_x\_next:
2253 \tl\_if\_empty:NTF \l\_unravel\_head\_tl
2254 { \use\_ii:nn }
2255 { \tl\_if\_in:nVTF { 0123456789 } \l\_unravel\_head\_tl }
2256 {
2257 \_unravel\_prev\_input:V \l\_unravel\_head\_tl
2258 \_unravel\_scan\_decimal\_loop:
2259 }
2260 {
2261 \token\_if\_eq\_catcode:NMF \l\_unravel\_head\_token \c\_space\_token
2262 { \_unravel\_back\_input: }
2263 \_unravel\_prev\_input\_silent:n { ~ }
2264 }
2265 }

```

(End definition for \\_unravel\\_scan\\_dimen:nN.)

```

\_unravel\_scan\_normal\_glue:
\_unravel\_scan\_mu\_glue:
2266 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_normal\_glue:
2267 { \_unravel\_scan\_glue:n { 2 } }
2268 \cs\_new\_protected\_nopar:Npn \_unravel\_scan\_mu\_glue:
2269 { \_unravel\_scan\_glue:n { 3 } }

```

(End definition for \\_unravel\\_scan\\_normal\\_glue: and \\_unravel\\_scan\\_mu\\_glue:.)

\\_unravel\\_scan\\_glue:n

```

2270 \cs\_new\_protected:Npn \_unravel\_scan\_glue:n #1
2271 {
2272 \_unravel\_prev\_input\_gpush:
2273 \_unravel\_scan\_signs:
2274 \_unravel\_prev\_input\_gpush:
2275 \_unravel\_set\_cmd:
2276 \_unravel\_cmd\_if\_internal:TF
2277 {
2278 \_unravel\_scan\_something\_internal:n {#1}
2279 \int\_case:nnF \g\_unravel\_val\_level\_int
2280 {
2281 { 0 } { \_unravel\_scan\_dim\_unit:nN {#1} \c\_false\_bool }
2282 { 1 } { }
2283 }

```

```

2284         { \_unravel_break:w }
2285     }
2286     { \_unravel_back_input: \_unravel_scan_dimen:nN {#1} \c_false_bool }
2287     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2288     \_unravel_prev_input_gpush:
2289     \_unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2290     \_unravel_scan_keyword:nT { pPlLuUsS }
2291     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2292     \_unravel_scan_keyword:nT { mMiInNuUsS }
2293     { \_unravel_scan_dimen:nN {#1} \c_true_bool }
2294     \_unravel_break_point:
2295     \_unravel_prev_input_join_get:nN {#1} \l__unravel_tmpa_tl
2296     \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2297 }

```

(End definition for \\_unravel\_scan\_glue:n.)

\\_unravel\_scan\_file\_name:

```

2298 \cs_new_protected_nopar:Npn \_unravel_scan_file_name:
2299 {
2300     \bool_gset_true:N \g__unravel_name_in_progress_bool
2301     \_unravel_get_x_non_blank:
2302     \_unravel_scan_file_name_loop:
2303     \bool_gset_false:N \g__unravel_name_in_progress_bool
2304     \_unravel_prev_input_silent:n { ~ }
2305 }
2306 \cs_new_protected_nopar:Npn \_unravel_scan_file_name_loop:
2307 {
2308     \_unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2309     { \_unravel_back_input: }
2310     {
2311         \tl_set:Nx \l__unravel_tmpa_tl
2312         { \_unravel_token_to_char:N \l__unravel_head_token }
2313         \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2314         {
2315             \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2316             \_unravel_get_x_next:
2317             \_unravel_scan_file_name_loop:
2318         }
2319     }
2320 }

```

(End definition for \\_unravel\_scan\_file\_name:.)

\\_unravel\_scan\_r\_token: This is analogous to  $\text{\TeX}$ '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.

```

2321 \cs_new_protected_nopar:Npn \_unravel_scan_r_token:
2322 {
2323     \bool_do_while:nn
2324     { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2325     { \_unravel_get_next: }
2326     \_unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2327     {
2328         \_unravel_error:nnnnn { missing-cs } { } { } { } { } { }

```

```

2329     \__unravel_back_input:
2330     \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2331   }
2332   \__unravel_prev_input_silent:V \l__unravel_head_tl
2333   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2334 }

```

(End definition for \\_\_unravel\_scan\_r\_token:.)

\\_\_unravel\_scan\_toks\_to\_str:

```

2335 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2336 {
2337   \__unravel_prev_input_gpush:
2338   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2339   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2340   \__unravel_prev_input_silent:x
2341   { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2342 }

```

(End definition for \\_\_unravel\_scan\_toks\_to\_str:.)

\\_\_unravel\_scan\_toks:NN

```

2343 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2344 {
2345   \bool_if:NT #1 { \__unravel_scan_param: }
2346   \__unravel_scan_left_brace:
2347   \bool_if:NTF #2
2348     { \__unravel_scan_group_x:N #1 }
2349     { \__unravel_scan_group_n:N #1 }
2350 }

```

(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.

```

2351 \cs_new_protected_nopar:Npn \__unravel_scan_param:
2352 {
2353   \tl_clear:N \l__unravel_tmpa_tl
2354   \__unravel_scan_param_aux:
2355   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2356   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2357 }
2358 \cs_new_protected_nopar:Npn \__unravel_scan_param_aux:
2359 {
2360   \__unravel_get_next:
2361   \tl_concat:NNN \l__unravel_tmpa_tl
2362     \l__unravel_tmpa_tl \l__unravel_head_tl
2363   \tl_if_empty:NTF \l__unravel_head_tl
2364     { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2365 }

```

(End definition for \\_\_unravel\_scan\_param: and \\_\_unravel\_scan\_param\_aux:.)



\\_unravel\_scan\_group\_n:N

```

2366 \cs_new_protected:Npn \_unravel_scan_group_n:N #1
2367 {
2368   \_unravel_back_input:
2369   \_unravel_input_gpop_item:NF \l__unravel_head_tl
2370   {
2371     \_unravel_error:nnnnn { runaway-text } { } { } { } { }
2372     \_unravel_exit:w
2373   }
2374   \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2375   \bool_if:NT #1
2376   { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2377   \_unravel_prev_input_silent:V \l__unravel_head_tl
2378 }

```

(End definition for \\_unravel\_scan\_group\_n:N.)

\\_unravel\_scan\_group\_x:N

```

2379 \cs_new_protected:Npn \_unravel_scan_group_x:N #1
2380 {
2381   \_unravel_input_gpop_tl:N \l__unravel_head_tl
2382   \_unravel_back_input:V \l__unravel_head_tl
2383   \bool_if:NTF #1
2384   {
2385     \_unravel_prev_input_silent:V \c_left_brace_str
2386     \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2387     \_unravel_scan_group_xdef:n { 1 }
2388   }
2389   {
2390     \_unravel_prev_input_gpush_gtl:
2391     \_unravel_prev_input_gtl:N \l__unravel_head_gtl
2392     \_unravel_scan_group_x:n { 1 }
2393     \_unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2394     \_unravel_prev_input_silent:x
2395     { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2396   }
2397 }

```

(End definition for \\_unravel\_scan\_group\_x:N.)

\\_unravel\_scan\_group\_xdef:n

```

2398 \cs_new_protected:Npn \_unravel_scan_group_xdef:n #1
2399 {
2400   \_unravel_get_token_x:N \c_true_bool
2401   \tl_if_empty:NTF \l__unravel_head_tl
2402   {
2403     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2404     {
2405       \_unravel_prev_input_silent:V \c_left_brace_str
2406       \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2407       \_unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2408     }
2409     {
2410       \_unravel_prev_input_silent:V \c_right_brace_str

```

```

2411         \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2412         \int_compare:nNnF {#1} = \c_one
2413         { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2414     }
2415 }
2416 {
2417     \__unravel_prev_input_silent:V \l__unravel_head_tl
2418     \tl_put_right:Nx \l__unravel_defining_tl
2419     { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2420     \__unravel_scan_group_xdef:n {#1}
2421 }
2422 }
2423 \cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }

```

(End definition for \\_\_unravel\_scan\_group\_xdef:n.)

\\_\_unravel\_scan\_group\_x:n

```

2424 \cs_new_protected:Npn \__unravel_scan_group_x:n #1
2425 {
2426     \__unravel_get_token_x:N \c_false_bool
2427     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2428     \tl_if_empty:NTF \l__unravel_head_tl
2429     {
2430         \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2431         { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2432         {
2433             \int_compare:nNnF {#1} = \c_one
2434             { \__unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2435         }
2436     }
2437     { \__unravel_scan_group_x:n {#1} }
2438 }
2439 \cs_generate_variant:Nn \__unravel_scan_group_x:n { f }

```

(End definition for \\_\_unravel\_scan\_group\_x:n.)

\\_\_unravel\_get\_token\_x:N

```

2440 \cs_new_protected:Npn \__unravel_get_token_x:N #1
2441 {
2442     \__unravel_get_next:
2443     \__unravel_token_if_protected:NF \l__unravel_head_token
2444     {
2445         \__unravel_set_cmd:
2446         \int_compare:nNnTF
2447         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { the } }
2448         {
2449             \__unravel_get_the:
2450             \bool_if:NTF #1
2451             {
2452                 \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
2453                 \__unravel_prev_input:V \l__unravel_head_tl
2454             }
2455             {
2456                 \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head
2457                 \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl

```

```

2458         \__unravel_print_action:
2459     }
2460 }
2461 { \__unravel_expand: }
2462 \__unravel_get_token_x:N #1
2463 }
2464 }

```

(End definition for \\_\_unravel\_get\_token\_x:N.)

\\_\_unravel\_scan\_alt\_rule:

```

2465 \cs_new_protected_nopar:Npn \__unravel_scan_alt_rule:
2466 {
2467     \__unravel_scan_keyword:nTF { wWiIdDtThH }
2468     {
2469         \__unravel_scan_normal_dimen:
2470         \__unravel_scan_alt_rule:
2471     }
2472     {
2473         \__unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2474         {
2475             \__unravel_scan_normal_dimen:
2476             \__unravel_scan_alt_rule:
2477         }
2478         {
2479             \__unravel_scan_keyword:nT { dDeEpPtThH }
2480             {
2481                 \__unravel_scan_normal_dimen:
2482                 \__unravel_scan_alt_rule:
2483             }
2484         }
2485     }
2486 }

```

(End definition for \\_\_unravel\_scan\_alt\_rule:.)

\\_\_unravel\_scan\_spec: Some TeX primitives accept the keywords `to` and `spread`, followed by a dimension.

```

2487 \cs_new_protected_nopar:Npn \__unravel_scan_spec:
2488 {
2489     \__unravel_scan_keyword:nTF { tToO } { \__unravel_scan_normal_dimen: }
2490     {
2491         \__unravel_scan_keyword:nT { sSpPrReEaAdD }
2492         { \__unravel_scan_normal_dimen: }
2493     }
2494     \__unravel_scan_left_brace:
2495 }

```

(End definition for \\_\_unravel\_scan\_spec:.)

## 2.8 Working with boxes

\\_\_unravel\_do\_box:N When this procedure is called, the last item in the previous-input sequence is

- empty if the box is meant to be put in the input stream,

- `\setbox⟨int⟩` if it is meant to be stored somewhere,
- `\moveright⟨dim⟩`, `\moveleft⟨dim⟩`, `\lower⟨dim⟩`, `\raise⟨dim⟩` if it is meant to be shifted,
- `\leaders` or `\cleaders` or `\xleaders`, in which case the argument is `\c_true_bool` (otherwise `\c_false_bool`).

If a `make_box` command follows, we fetch the operands. If leaders are followed by a rule, then this is also ok. In all other cases, call `\__unravel_do_box_error:` to clean up.

```

2496 \cs_new_protected:Npn \__unravel_do_box:N #1
2497 {
2498   \__unravel_get_x_non_relax:
2499   \__unravel_set_cmd:
2500   \int_compare:nNnTF
2501     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2502     { \__unravel_do_begin_box:N #1 }
2503     {
2504       \bool_if:NTF #1
2505       {
2506         \int_case:nnTF \l__unravel_head_cmd_int
2507         {
2508           { \__unravel_tex_use:n { hrule } } { } { }
2509           { \__unravel_tex_use:n { vrule } } { } { }
2510         }
2511         { \__unravel_do_leaders_rule: }
2512         { \__unravel_do_box_error: }
2513       }
2514       { \__unravel_do_box_error: }
2515     }
2516 }

```

(End definition for `\__unravel_do_box:N`.)

`\__unravel_do_box_error:` Put the (non-`make_box`) command back into the input and complain. Then recover by throwing away the action (last item of the previous-input sequence). For some reason (this appears to be what  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  does), there is no need to remove the after assignment token here.

```

2517 \cs_new_protected_nopar:Npn \__unravel_do_box_error:
2518 {
2519   \__unravel_back_input:
2520   \__unravel_error:nnnnn { missing-box } { } { } { } { }
2521   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2522   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2523 }

```

(End definition for `\__unravel_do_box_error:.`)

`\__unravel_do_begin_box:N` We have just found a `make_box` command and placed it into the last item of the previous-input sequence. If it is “simple” (`\box⟨int⟩`, `\copy⟨int⟩`, `\lastbox`, `\vsplit⟨int⟩` to `⟨dim⟩`) then we grab its operands, then call `\__unravel_do_simple_box:N` to finish up. If it is `\vtop` or `\vbox` or `\hbox`, we need to work harder.

```

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

```

```

2526 \__unravel_prev_input:V \l__unravel_head_tl
2527 \int_case:nnTF \l__unravel_head_char_int
2528 {
2529   { 0 } { \__unravel_scan_int: } % box
2530   { 1 } { \__unravel_scan_int: } % copy
2531   { 2 } { } % lastbox
2532   { 3 } % vsplit
2533   {
2534     \__unravel_scan_int:
2535     \__unravel_scan_to:
2536     \__unravel_scan_normal_dimen:
2537   }
2538 }
2539 { \__unravel_do_simple_box:N #1 }
2540 { \__unravel_do_box_explicit:N #1 }
2541 }

```

(End definition for \\_\_unravel\_do\_begin\_box:N.)

\\_\_unravel\_do\_simple\_box:N For leaders, we need to fetch a glue. In all cases, retrieve the box construction (such as \raise3pt\vsplit7to5em). Finally, let T<sub>E</sub>X run the code and print what we have done. In the case of \shipout, check that \mag has a value between 1 and 32768.

```

2542 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2543 {
2544   \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2545   {
2546     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2547     \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2548     { \__unravel_prepare_mag: }
2549     \tl_use:N \l__unravel_head_tl \scan_stop:
2550     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2551     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2552   }
2553 }

```

(End definition for \\_\_unravel\_do\_simple\_box:N.)

\\_\_unravel\_do\_leaders\_fetch\_skip:

```

2554 \cs_new_protected_nopar:Npn \__unravel_do_leaders_fetch_skip:
2555 {
2556   \__unravel_get_x_non_relax:
2557   \__unravel_set_cmd:
2558   \int_compare:nNnTF \l__unravel_head_cmd_int
2559   = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2560   {
2561     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2562     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2563     \__unravel_do_append_glue:
2564   }
2565   {
2566     \__unravel_back_input:
2567     \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2568     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2569     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }

```

```

2570     }
2571 }

```

(End definition for `\_unravel_do_leaders_fetch_skip:`)

`\_unravel_do_box_explicit:N` At this point, the last item in the previous-input sequence is typically `\setbox0\hbox` or `\raise 3pt\hbox`. Scan for keywords `to` and `spread` and a left brace. Install a hook in `\everyhbox` or `\everyvbox` (whichever T<sub>E</sub>X 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<sub>E</sub>X perform the box operation (here we need to provide the begin-group token, as it was scanned but not placed in the previous-input sequence). T<sub>E</sub>X 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.

```

2572 \cs_new_protected:Npn \_unravel_do_box_explicit:N #1
2573 {
2574   \token_if_eq_meaning:NNTF \l__unravel_head_token \_unravel_hbox:w
2575   { \_unravel_box_hook:N \tex_everyhbox:D }
2576   { \_unravel_box_hook:N \tex_everyvbox:D }
2577   % ^^A todo: TeX calls |normal_paragraph| here.
2578   \_unravel_scan_spec:
2579   \_unravel_prev_input_gpop:N \l__unravel_head_tl
2580   \_unravel_set_action_text:x
2581   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2582   \seq_push:Nf \l__unravel_leaders_box_seq
2583   { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2584   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2585   \gtl_gconcat:NNN \g__unravel_output_gtl
2586   \g__unravel_output_gtl \c_group_begin_gtl
2587   \tl_use:N \l__unravel_head_tl
2588   \c_group_begin_token \_unravel_box_hook_end:
2589 }

```

(End definition for `\_unravel_do_box_explicit:N`)

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

`\_unravel_box_hook:w`

`\_unravel_box_hook_end:`

```

2590 \cs_new_protected:Npn \_unravel_box_hook:N #1
2591 {
2592   \tl_set:NV \l__unravel_tmpa_tl #1
2593   \str_if_eq_x:nnF
2594   { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \_unravel_box_hook:w }
2595   {
2596     \_unravel_exp_args:Nx #1
2597     {
2598       \exp_not:n { \_unravel_box_hook:w \prg_do_nothing: }
2599       \exp_not:V #1
2600     }
2601   }
2602   \cs_gset_protected:Npn \_unravel_box_hook:w ##1 \_unravel_box_hook_end:
2603   {
2604     \exp_args:No #1 {##1}

```

```

2605     \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2606     \gtl_clear:N \l__unravel_after_group_gtl
2607     \__unravel_print_action:
2608     \__unravel_back_input:o {##1}
2609     \__unravel_set_action_text:x
2610     { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2611     \tl_if_empty:oF {##1} { \__unravel_print_action: }
2612   }
2613 }
2614 \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2615 \cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:

```

(End definition for \\_\_unravel\_box\_hook:N, \\_\_unravel\_box\_hook:w, and \\_\_unravel\_box\_hook\_end:.)

\\_\_unravel\_do\_leaders\_rule: After finding a vrule or hrule command and looking for depth, heigh and width keywords, we are in the same situation as after finding a box. Fetch the required skip accordingly.

```

2616 \cs_new_protected_nopar:Npn \__unravel_do_leaders_rule:
2617 {
2618   \__unravel_prev_input:V \l__unravel_head_tl
2619   \__unravel_scan_alt_rule:
2620   \__unravel_do_leaders_fetch_skip:
2621 }

```

(End definition for \\_\_unravel\_do\_leaders\_rule:.)

## 2.9 Paragraphs

\\_\_unravel\_charcode\_if\_safe:nTF

```

2622 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2623 {
2624   \bool_if:nTF
2625   {
2626     \int_compare_p:n { #1 = '!' }
2627     || \int_compare_p:n { ' ' <= #1 <= '[' }
2628     || \int_compare_p:n { #1 = ']' }
2629     || \int_compare_p:n { ' ' <= #1 <= 'z' }
2630   }
2631   { \prg_return_true: }
2632   { \prg_return_false: }
2633 }

```

(End definition for \\_\_unravel\_charcode\_if\_safe:nTF.)

\\_\_unravel\_char:n  
\\_\_unravel\_char:V  
\\_\_unravel\_char:x

```

2634 \cs_new_protected:Npn \__unravel_char:n #1
2635 {
2636   \tex_char:D #1 \scan_stop:
2637   \__unravel_charcode_if_safe:nTF {#1}
2638   { \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } } }
2639   {
2640     \tl_set:Nx \l__unravel_tmpa_tl
2641     { \exp_not:N \char \int_eval:n {#1} ~ }
2642   }

```

```

2643 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2644 \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2645 }
2646 \cs_generate_variant:Nn \__unravel_char:n { V }
2647 \cs_new_protected:Npn \__unravel_char:x
2648 { \__unravel_exp_args:Nx \__unravel_char:n }

```

(End definition for \\_\_unravel\_char:n.)

```

\__unravel_char_in_mmode:n
\__unravel_char_in_mmode:V
\__unravel_char_in_mmode:x
2649 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2650 {
2651   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2652   { % math active
2653     \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_head_gtl
2654     { \char_generate:nn {#1} { 12 } }
2655     \__unravel_back_input:
2656   }
2657   { \__unravel_char:n {#1} }
2658 }
2659 \cs_generate_variant:Nn \__unravel_char_in_mmode:n { V }
2660 \cs_new_protected:Npn \__unravel_char_in_mmode:x
2661 { \__unravel_exp_args:Nx \__unravel_char_in_mmode:n }

```

(End definition for \\_\_unravel\_char\_in\_mmode:n.)

```

\__unravel_mathchar:n
\__unravel_mathchar:x
2662 \cs_new_protected:Npn \__unravel_mathchar:n #1
2663 {
2664   \tex_mathchar:D #1 \scan_stop:
2665   \tl_set:Nx \l__unravel_tmpa_tl
2666   { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2667   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2668   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2669 }
2670 \cs_new_protected:Npn \__unravel_mathchar:x
2671 { \__unravel_exp_args:Nx \__unravel_mathchar:n }

```

(End definition for \\_\_unravel\_mathchar:n.)

\\_\_unravel\_new\_graf:N The argument is a boolean, indicating whether the paragraph should be indented. We have much less work to do here than TeX itself. Our only task is to correctly position the \everypar tokens in the input that we will read, rather than letting TeX run the code right away.

```

2672 \cs_new_protected:Npn \__unravel_new_graf:N #1
2673 {
2674   \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
2675   \__unravel_everypar:w { }
2676   \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2677   \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl
2678   \__unravel_back_input:V \l__unravel_tmpa_tl
2679   \__unravel_print_action:x
2680   {
2681     \g__unravel_action_text_str \c_space_tl : ~

```



```

2682         \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2683     }
2684 }

```

(End definition for `\__unravel_new_graf:N`.)

`\__unravel_end_graf:`

```

2685 \cs_new_protected_nopar:Npn \__unravel_end_graf:
2686 { \mode_if_horizontal:T { \__unravel_normal_paragraph: } }

```

(End definition for `\__unravel_end_graf:.`)

`\__unravel_normal_paragraph:`

```

2687 \cs_new_protected_nopar:Npn \__unravel_normal_paragraph:
2688 {
2689     \tex_par:D
2690     \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2691     \__unravel_print_action:x { Paragraph~end. }
2692 }

```

(End definition for `\__unravel_normal_paragraph:.`)

`\__unravel_build_page:`

```

2693 \cs_new_protected_nopar:Npn \__unravel_build_page:
2694 {
2695 }

```

(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.

```

2696 \cs_new_protected_nopar:Npn \__unravel_handle_right_brace:
2697 {
2698     \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
2699     {
2700         \gtl_gconcat:NNN \g__unravel_output_gtl
2701         \g__unravel_output_gtl \c_group_end_gtl
2702         \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
2703         \int_case:nn \__unravel_currentgrouptype:
2704         {
2705             { 1 } { \__unravel_end_simple_group: } % simple
2706             { 2 } { \__unravel_end_box_group: } % hbox
2707             { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2708             { 4 } { \__unravel_end_graf: \__unravel_end_box_group: } % vbox
2709             { 5 } { \__unravel_end_graf: \__unravel_end_box_group: } % vtop
2710             { 6 } { \__unravel_end_align_group: } % align
2711             { 7 } { \__unravel_end_no_align_group: } % no_align
2712             { 8 } { \__unravel_end_output_group: } % output
2713             { 9 } { \__unravel_end_simple_group: } % math
2714             { 10 } { \__unravel_end_disc_group: } % disc
2715             { 11 } { \__unravel_end_graf: \__unravel_end_simple_group: } % insert
2716             { 12 } { \__unravel_end_graf: \__unravel_end_simple_group: } % vcenter
2717             { 13 } { \__unravel_end_math_choice_group: } % math_choice
2718         }

```

```

2719     }
2720     { % bottom_level, semi_simple, math_shift, math_left
2721       \l__unravel_head_token
2722       \__unravel_print_action:
2723     }
2724   }

```

(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.

```

2725 \cs_new_protected_nopar:Npn \__unravel_end_simple_group:
2726 {
2727   \l__unravel_head_token
2728   \__unravel_print_action:
2729 }

```

(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.

```

2730 \cs_new_protected_nopar:Npn \__unravel_end_box_group:
2731 {
2732   \seq_pop:NN \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2733   \exp_args:No \__unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
2734 }
2735 \cs_new_protected:Npn \__unravel_end_box_group_aux:n #1
2736 {
2737   \str_if_eq:x:nnTF {#1} { Z }
2738   { \__unravel_end_simple_group: }
2739   {
2740     \__unravel_get_x_non_relax:
2741     \__unravel_set_cmd:
2742     \int_compare:nNnTF \l__unravel_head_cmd_int
2743       = { \__unravel_tex_use:n { #1 skip } }
2744     {
2745       \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2746       \__unravel_do_append_glue:
2747     }
2748     {
2749       \__unravel_back_input:
2750       \c_group_end_token \group_begin: \group_end:
2751       \__unravel_print_action:
2752     }
2753   }
2754 }

```

(End definition for \\_\_unravel\_end\_box\_group:.)

\\_\_unravel\_off\_save:

```

2755 \cs_new_protected_nopar:Npn \__unravel_off_save:
2756 {
2757   \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
2758   { % bottom-level
2759     \__unravel_error:nxxxx { extra-close }

```

```

2760         { \token_to_meaning:N \l__unravel_head_token } { } { } { }
2761     }
2762     {
2763         \__unravel_back_input:
2764         \int_case:nnF \__unravel_currentgrouptype:
2765         {
2766             { 14 } % semi_simple_group
2767             { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
2768             { 15 } % math_shift_group
2769             { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2770             { 16 } % math_left_group
2771             { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2772         }
2773         { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2774         \__unravel_back_input:
2775         \__unravel_error:nxxxx { off-save }
2776         { \gtl_to_str:N \l__unravel_head_gtl } { } { } { } { }
2777     }
2778 }

```

(End definition for \\_\_unravel\_off\_save:.)

## 2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
2779 \cs_new_protected:Npn \__unravel_mode_math:n #1
2780 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
2781 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
2782 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
2783 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
2784 {
2785     \mode_if_math:TF
2786     { \__unravel_insert_dollar_error: }
2787     { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
2788 }
2789 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
2790 {
2791     \mode_if_vertical:TF
2792     { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
2793     {#1}
2794 }

```

(End definition for \\_\_unravel\_mode\_math:n, \\_\_unravel\_mode\_non\_math:n, and \\_\_unravel\_mode\_vertical:n.)

\\_\_unravel\_head\_for\_vmode: See T<sub>E</sub>X's head\_for\_vmode.

```

2795 \cs_new_protected_nopar:Npn \__unravel_head_for_vmode:
2796 {
2797     \mode_if_inner:TF
2798     {
2799         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hruler:D
2800         {
2801             \__unravel_error:nnnnn { hruler-bad-mode } { } { } { } { }
2802             \__unravel_print_action:

```

```

2803     }
2804     { \_unravel_off_save: }
2805   }
2806   {
2807     \_unravel_back_input:
2808     \gtl_set:Nn \l__unravel_head_gtl { \par }
2809     \_unravel_back_input:
2810   }
2811 }

```

(End definition for \\_unravel\_head\_for\_vmode:.)

\\_unravel\_goto\_inner\_math:

```

2812 \cs_new_protected_nopar:Npn \_unravel_goto_inner_math:
2813 {
2814   \_unravel_box_hook:N \tex_everymath:D
2815   $ % $
2816   \_unravel_box_hook_end:
2817 }

```

(End definition for \\_unravel\_goto\_inner\_math:.)

\\_unravel\_goto\_display\_math:

```

2818 \cs_new_protected_nopar:Npn \_unravel_goto_display_math:
2819 {
2820   \_unravel_box_hook:N \tex_everydisplay:D
2821   $ $
2822   \_unravel_box_hook_end:
2823 }

```

(End definition for \\_unravel\_goto\_display\_math:.)

\\_unravel\_after\_math:

```

2824 \cs_new_protected_nopar:Npn \_unravel_after_math:
2825 {
2826   \mode_if_inner:TF
2827   {
2828     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2829     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2830     $ % $
2831   }
2832   {
2833     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2834     \_unravel_get_x_next:
2835     \token_if_eq_catcode:NNF
2836       \l__unravel_head_token \c_math_toggle_token
2837     {
2838       \_unravel_back_input:
2839       \tl_set:Nn \l__unravel_head_tl { $ } % $
2840       \_unravel_error:nnnnn { missing-dollar } { } { } { } { }
2841     }
2842     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2843     \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
2844     $ $
2845   }

```

```

2846     \__unravel_print_action:
2847 }

```

(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.

```

2848 \cs_new_protected_nopar:Npn \__unravel_do_step:
2849 {
2850     \__unravel_set_action_text:
2851     \bool_if:NT \g__unravel_internal_debug_bool
2852     { \__unravel_exp_args:Nx \iow_term:n { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int
2853     \cs_if_exist_use:cF
2854     { \__unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2855     { \__unravel_error:nxxxx { internal } { unknown-command } { } { } { } }
2856 }

```

(End definition for \\_\_unravel\_do\_step:.)

## 2.13 Commands

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

### 2.13.1 Characters: from 0 to 15

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

- relax=0 for \relax.
- begin-group\_char=1 for begin-group characters (catcode 1).
- end-group\_char=2 for end-group characters (catcode 2).
- math\_char=3 for math shift (math toggle in expl3) characters (catcode 3).
- tab\_mark=4 for \span
- alignment\_char=4 for alignment tab characters (catcode 4).
- car\_ret=5 for \cr and \crr.
- macro\_char=6 for macro parameter characters (catcode 6).
- superscript\_char=7 for superscript characters (catcode 7).
- subscript\_char=8 for subscript characters (catcode 8).
- endv=9 for ?.
- blank\_char=10 for blank spaces (catcode 10).
- the\_char=11 for letters (catcode 11).
- other\_char=12 for other characters (catcode 12).

- par\_end=13 for \par.
- stop=14 for \end and \dump.
- delim\_num=15 for \delimiter.

Not implemented at all: endv.

\relax does nothing.

```
2857 \__unravel_new_tex_cmd:nn { relax } % 0
2858 { \__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.

```
2859 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
2860 {
2861   \gtl_gconcat:NNN \g__unravel_output_gtl
2862   \g__unravel_output_gtl \c_group_begin_gtl
2863   \__unravel_print_action:
2864   \l__unravel_head_token
2865   \gtl_clear:N \l__unravel_after_group_gtl
2866 }
2867 \__unravel_new_tex_cmd:nn { end-group_char } % 2
2868 { \__unravel_handle_right_brace: }
```

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

```
2869 \__unravel_new_tex_cmd:nn { math_char } % 3
2870 {
2871   \__unravel_mode_non_vertical:n
2872   {
2873     \mode_if_math:TF
2874     {
2875       \int_compare:nNnTF
2876       \__unravel_currentgrouptype: = { 15 } % math_shift_group
2877       { \__unravel_after_math: }
2878       { \__unravel_off_save: }
2879     }
2880     {
2881       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2882       \__unravel_get_next:
2883       \token_if_eq_catcode:NNTF
2884       \l__unravel_head_token \c_math_toggle_token
2885       {
2886         \mode_if_inner:TF
2887         { \__unravel_back_input: \__unravel_goto_inner_math: }
2888         {
2889           \gtl_gput_right:NV
2890           \g__unravel_output_gtl \l__unravel_head_tl
2891           \__unravel_goto_display_math:
2892         }
2893       }
2894       { \__unravel_back_input: \__unravel_goto_inner_math: }
2895     }
2896   }
2897 }
```

Some commands are errors when they reach T<sub>E</sub>X's stomach. Among others, `tab_mark=alignment_char`, `car_ret` and `macro_char`. We let T<sub>E</sub>X insert the proper error.

```

2898 \__unravel_new_tex_cmd:nn { alignment_char } % 4
2899 { \l__unravel_head_token \__unravel_print_action: }
2900 \__unravel_new_tex_cmd:nn { car_ret } % 5
2901 { \l__unravel_head_token \__unravel_print_action: }
2902 \__unravel_new_tex_cmd:nn { macro_char } % 6
2903 { \l__unravel_head_token \__unravel_print_action: }

2904 \__unravel_new_tex_cmd:nn { superscript_char } % 7
2905 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2906 \__unravel_new_tex_cmd:nn { subscript_char } % 8
2907 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2908 \cs_new_protected_nopar:Npn \__unravel_sub_sup:
2909 {
2910   \__unravel_prev_input_gpush:N \l__unravel_head_tl
2911   \__unravel_print_action:
2912   \__unravel_get_x_non_relax:
2913   \__unravel_set_cmd:
2914   \int_case:nnTF \l__unravel_head_cmd_int
2915   {
2916     { \__unravel_tex_use:n { the_char } }
2917     { \__unravel_prev_input:V \l__unravel_head_tl }
2918     { \__unravel_tex_use:n { other_char } }
2919     { \__unravel_prev_input:V \l__unravel_head_tl }
2920     { \__unravel_tex_use:n { char_given } }
2921     { \__unravel_prev_input:V \l__unravel_head_tl }
2922     { \__unravel_tex_use:n { char_num } }
2923     {
2924       \__unravel_prev_input:V \l__unravel_head_tl
2925       \__unravel_scan_int:
2926     }
2927     { \__unravel_tex_use:n { math_char_num } }
2928     {
2929       \__unravel_prev_input:V \l__unravel_head_tl
2930       \__unravel_scan_int:
2931     }
2932     { \__unravel_tex_use:n { math_given } }
2933     { \__unravel_prev_input:V \l__unravel_head_tl }
2934     { \__unravel_tex_use:n { delim_num } }
2935     { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
2936   }
2937   {
2938     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2939     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2940     \tl_use:N \l__unravel_head_tl \scan_stop:
2941   }
2942   {
2943     \__unravel_back_input:
2944     \__unravel_scan_left_brace:
2945     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2946     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2947     \gtl_gconcat:NNN \g__unravel_output_gtl

```

```

2948         \g__unravel_output_gtl \c_group_begin_gtl
2949         \tl_use:N \l__unravel_head_tl \c_group_begin_token
2950     }
2951     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2952 }

2953 \__unravel_new_tex_cmd:nn { endv } % 9
2954 { \__unravel_not_implemented:n { alignments } }

Blank spaces are ignored in vertical and math modes in the same way as \relax is
in all modes. In horizontal mode, add them to the output.

2955 \__unravel_new_tex_cmd:nn { blank_char } % 10
2956 {
2957     \mode_if_horizontal:T
2958     {
2959         \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
2960         \l__unravel_head_token
2961     }
2962     \__unravel_print_action:
2963 }

Letters and other characters leave vertical mode.

2964 \__unravel_new_tex_cmd:nn { the_char } % 11
2965 {
2966     \__unravel_mode_non_vertical:n
2967     {
2968         \tl_set:Nx \l__unravel_tmpa_tl
2969         { ' \__unravel_token_to_char:N \l__unravel_head_token }
2970         \mode_if_math:TF
2971         { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
2972         { \__unravel_char:V \l__unravel_tmpa_tl }
2973     }
2974 }

2975 \__unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12

2976 \__unravel_new_tex_cmd:nn { par_end } % 13
2977 {
2978     \__unravel_mode_non_math:n
2979     {
2980         \mode_if_vertical:TF
2981         { \__unravel_normal_paragraph: }
2982         {
2983             % if align_state<0 then off_save;
2984             \__unravel_end_graf:
2985             \mode_if_vertical:T
2986             { \mode_if_inner:F { \__unravel_build_page: } }
2987         }
2988     }
2989 }

2990 \__unravel_new_tex_cmd:nn { stop } % 14
2991 {
2992     \__unravel_mode_vertical:n
2993     {
2994         \mode_if_inner:TF
2995         { \__unravel_forbidden_case: }

```



```

2996     {
2997         % ^^A todo: unless its_all_over
2998         \int_gdecr:N \g__unravel_ends_int
2999         \int_compare:nNnTF \g__unravel_ends_int > \c_zero
3000         {
3001             \__unravel_back_input:
3002             \__unravel_back_input:n
3003             {
3004                 \__unravel_hbox:w to \tex_hsize:D { }
3005                 \tex_vfill:D
3006                 \tex_penalty:D - '10000000000 ~
3007             }
3008             \__unravel_build_page:
3009             \__unravel_print_action:x { End-everything! }
3010         }
3011         {
3012             \__unravel_print_outcome:
3013             \l__unravel_head_token
3014         }
3015     }
3016 }
3017 }
3018 \__unravel_new_tex_cmd:nn { delim_num } % 15
3019 {
3020     \__unravel_mode_math:n
3021     {
3022         \__unravel_prev_input_gpush:N \l__unravel_head_tl
3023         \__unravel_print_action:
3024         \__unravel_scan_int:
3025         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3026         \tl_use:N \l__unravel_head_tl \scan_stop:
3027         \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3028     }
3029 }

```

### 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`.

```

3030 \__unravel_new_tex_cmd:nn { char_num } % 16
3031 {
3032   \__unravel_mode_non_vertical:n
3033   {
3034     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3035     \__unravel_print_action:
3036     \__unravel_scan_int:
3037     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3038     \mode_if_math:TF
3039     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
3040     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
3041   }
3042 }

```

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.

```

3043 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3044 {
3045   \__unravel_mode_math:n
3046   {
3047     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3048     \__unravel_print_action:
3049     \__unravel_scan_int:
3050     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3051     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
3052   }
3053 }
3054 \__unravel_new_tex_cmd:nn { mark } % 18
3055 {
3056   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3057   \__unravel_print_action:
3058   \int_compare:nNnF \l__unravel_head_char_int = \c_zero
3059   { \__unravel_scan_int: }
3060   \__unravel_prev_input_gpush:
3061   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3062   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3063   \__unravel_prev_input_gpop:N \l__unravel_head_tl

```

```

3064 \__unravel_print_action:x
3065 { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3066 \tl_put_right:Nx \l__unravel_head_tl
3067 { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3068 \tl_use:N \l__unravel_head_tl
3069 }

```

We now implement the primitives `\show`, `\showbox`, `\showthe`, `\showlists`, `\showgroups`, `\showtokens` and `\showifs`. Those with no operand are sent to `TeX` after printing the action. Those with operands print first, then scan their operands, then are sent to `TeX`. The case of `\show` is a bit special, as its operand is a single token, which cannot easily be put into the the previous-input sequence in general. Since no expansion can occur, simply grab the token and show it.

```

3070 \__unravel_new_tex_cmd:nn { xray } % 19
3071 {
3072   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3073   \__unravel_print_action:
3074   \int_case:nnF \l__unravel_head_char_int
3075   {
3076     { 0 }
3077     { % show
3078       \__unravel_get_next:
3079       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3080       \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl
3081     }
3082     { 2 }
3083     { % showthe
3084       \__unravel_get_x_next:
3085       \__unravel_scan_something_internal:n { 5 }
3086       \__unravel_prev_input_gpop:N \l__unravel_head_tl
3087       \__unravel_exp_args:Nx \use:n
3088       { \etex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3089     }
3090   }
3091   { % no operand for showlists, showgroups, showifs
3092     \int_compare:nNnT \l__unravel_head_char_int = \c_one % showbox
3093     { \__unravel_scan_int: }
3094     \int_compare:nNnT \l__unravel_head_char_int = \c_five % showtokens
3095     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3096     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3097     \tl_use:N \l__unravel_head_tl \scan_stop:
3098   }
3099 }

```

make\_box=20 for `\box`, `\copy`, `\lastbox`, `\vsplit`, `\vtop`, `\vbox`, and `\hbox` (106).

```

3100 \__unravel_new_tex_cmd:nn { make_box } % 20
3101 {
3102   \__unravel_prev_input_gpush:
3103   \__unravel_back_input:
3104   \__unravel_do_box:N \c_false_bool
3105 }

```

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

```

3106 \cs_new_protected_nopar:Npn \__unravel_do_move:

```

```

3107 {
3108   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3109   \__unravel_print_action:
3110   \__unravel_scan_normal_dimen:
3111   \__unravel_do_box:N \c_false_bool
3112 }

(End definition for \__unravel_do_move:.)
hmove=21 for \moveright and \moveleft.

3113 \__unravel_new_tex_cmd:nn { hmove } % 21
3114 {
3115   \mode_if_vertical:TF
3116   { \__unravel_do_move: } { \__unravel_forbidden_case: }
3117 }

vmove=22 for \lower and \raise.

3118 \__unravel_new_tex_cmd:nn { vmove } % 22
3119 {
3120   \mode_if_vertical:TF
3121   { \__unravel_forbidden_case: } { \__unravel_do_move: }
3122 }

```

\\_\_unravel\_do\_unpackage:

```

3123 \cs_new_protected_nopar:Npn \__unravel_do_unpackage:
3124 {
3125   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3126   \__unravel_print_action:
3127   \__unravel_scan_int:
3128   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3129   \tl_use:N \l__unravel_head_tl \scan_stop:
3130   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3131 }

(End definition for \__unravel_do_unpackage:.)
un_hbox=23 for \unhbox and \unhcopy.

3132 \__unravel_new_tex_cmd:nn { un_hbox } % 23
3133 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

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

3134 \__unravel_new_tex_cmd:nn { un_vbox } % 24
3135 {
3136   \__unravel_mode_vertical:n
3137   {
3138     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3139     { \l__unravel_head_token \__unravel_print_action: }
3140     { \__unravel_do_unpackage: }
3141   }
3142 }

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.

3143 \__unravel_new_tex_cmd:nn { remove_item } % 25
3144 { \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.

```

3145 \cs_new_protected_nopar:Npn \__unravel_do_append_glue:
3146 {
3147   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3148   { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3149   {
3150     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3151     \__unravel_print_action:
3152     \exp_args:Nf \__unravel_scan_glue:n
3153     { \int_eval:n { \l__unravel_head_char_int - 2 } }
3154     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3155     \tl_use:N \l__unravel_head_tl \scan_stop:
3156     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3157   }
3158 }

```

(End definition for `\__unravel_do_append_glue:.`)

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

```

3159 \__unravel_new_tex_cmd:nn { hskip } % 26
3160 { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }
3161 \__unravel_new_tex_cmd:nn { vskip } % 27
3162 { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }
3163 \__unravel_new_tex_cmd:nn { mskip } % 28
3164 { \__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.

```

3165 \cs_new_protected_nopar:Npn \__unravel_do_append_kern:
3166 {
3167   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3168   \__unravel_print_action:
3169   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
3170   { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
3171   { \__unravel_scan_dimen:nN { 3 } \c_false_bool }
3172   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3173   \tl_use:N \l__unravel_head_tl \scan_stop:
3174   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3175 }

```

(End definition for `\__unravel_do_append_kern:.`)

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

```

3176 \__unravel_new_tex_cmd:nn { kern } % 29
3177 { \__unravel_do_append_kern: }
3178 \__unravel_new_tex_cmd:nn { mkern } % 30
3179 { \__unravel_mode_math:n { \__unravel_do_append_kern: } }

```

```

leader_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).
3180 \_unravel_new_tex_cmd:nn { leader_ship } % 31
3181 {
3182   \_unravel_prev_input_gpush:N \l\_unravel_head_tl
3183   \_unravel_print_action:
3184   \_unravel_do_box:N \c_true_bool
3185 }

```

### 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

```

3186 \_unravel_new_tex_cmd:nn { halign } % 32
3187 { \_unravel_not_implemented:n { halign } }
3188 \_unravel_new_tex_cmd:nn { valign } % 33
3189 { \_unravel_not_implemented:n { valign } }
3190 \_unravel_new_tex_cmd:nn { no_align } % 34
3191 { \_unravel_not_implemented:n { noalign } }
3192 \_unravel_new_tex_cmd:nn { vrule } % 35
3193 { \_unravel_mode_non_vertical:n { \_unravel_do_rule: } }
3194 \_unravel_new_tex_cmd:nn { hrule } % 36
3195 { \_unravel_mode_vertical:n { \_unravel_do_rule: } }
3196 \cs_new_protected_nopar:Npn \_unravel_do_rule:
3197 {
3198   \_unravel_prev_input_gpush:N \l\_unravel_head_tl
3199   \_unravel_print_action:
3200   \_unravel_scan_alt_rule:

```

```

3201 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3202 \tl_use:N \l__unravel_head_tl \scan_stop:
3203 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3204 }

3205 \__unravel_new_tex_cmd:nn { insert } % 37
3206 { \__unravel_begin_insert_or_adjust: }
3207 \__unravel_new_tex_cmd:nn { vadjust } % 38
3208 {
3209 \mode_if_vertical:TF
3210 { \__unravel_forbidden_case: } { \__unravel_begin_insert_or_adjust: }
3211 }

3212 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
3213 {
3214 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3215 {
3216 \__unravel_print_action:
3217 \__unravel_get_x_non_blank:
3218 \__unravel_set_cmd:
3219 \__unravel_do_step:
3220 }
3221 { \__unravel_not_implemented:n { pdfprimitive } }
3222 }

3223 \__unravel_new_tex_cmd:nn { after_assignment } % 40
3224 {
3225 \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3226 \__unravel_get_next:
3227 \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3228 \__unravel_print_action:x
3229 {
3230 Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3231 \gtl_to_str:N \l__unravel_head_gtl
3232 }
3233 }

```

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.

```

3234 \__unravel_new_tex_cmd:nn { after_group } % 41
3235 {
3236 \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3237 \__unravel_get_next:
3238 \int_compare:nNnTF \__unravel_currentgrouptype: = \c_zero
3239 {
3240 \__unravel_print_action:x
3241 {
3242 Aftergroup~(level~0~=>~dropped):~
3243 \tl_to_str:N \l__unravel_tmpa_tl
3244 \gtl_to_str:N \l__unravel_head_gtl
3245 }
3246 }
3247 {
3248 \gtl_concat:NNN \l__unravel_after_group_gtl
3249 \l__unravel_after_group_gtl \l__unravel_head_gtl
3250 \__unravel_print_action:x

```

```

3251         {
3252             Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3253             \gtl_to_str:N \l__unravel_head_gtl
3254         }
3255     }
3256 }
See \__unravel_do_append_glue:.
3257 \__unravel_new_tex_cmd:nn { break_penalty } % 42
3258 {
3259     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3260     \__unravel_print_action:
3261     \__unravel_scan_int:
3262     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3263     \tl_use:N \l__unravel_head_tl \scan_stop:
3264     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3265 }
3266 \__unravel_new_tex_cmd:nn { start_par } % 43
3267 {
3268     \mode_if_vertical:TF
3269     {
3270         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3271         { \__unravel_new_graf:N \c_false_bool }
3272         { \__unravel_new_graf:N \c_true_bool }
3273     }
3274     {
3275         \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3276         {
3277             \__unravel_hbox:w width \tex_parindent:D { }
3278             \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3279         }
3280         \__unravel_print_action:
3281     }
3282 }
3283 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3284 {
3285     \mode_if_vertical:TF { \__unravel_forbidden_case: }
3286     { \l__unravel_head_token \__unravel_print_action: }
3287 }

```

\\_\_unravel\_do\_accent:

```

3288 \cs_new_protected_nopar:Npn \__unravel_do_accent:
3289 {
3290     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3291     \__unravel_print_action:
3292     \__unravel_scan_int:
3293     \__unravel_do_assignments:
3294     \bool_if:nTF
3295     {
3296         \token_if_eq_catcode_p:NN
3297         \l__unravel_head_token \c_catcode_letter_token
3298         ||
3299         \token_if_eq_catcode_p:NN

```



```

3300         \l__unravel_head_token \c_catcode_other_token
3301         ||
3302         \int_compare_p:nNn
3303         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3304     }
3305     { \__unravel_prev_input:V \l__unravel_head_tl }
3306     {
3307         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3308         {
3309             \__unravel_prev_input:V \l__unravel_head_tl
3310             \__unravel_scan_int:
3311         }
3312         { \__unravel_break:w }
3313     }
3314     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3315     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3316     \tl_use:N \l__unravel_head_tl \scan_stop:
3317     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3318     \__unravel_break_point:
3319 }

```

(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).

```

3320 \cs_new_protected_nopar:Npn \__unravel_do_math_accent:
3321 {
3322     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3323     \__unravel_print_action:
3324     \__unravel_scan_int:
3325     \__unravel_scan_math:
3326     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3327     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3328     \tl_use:N \l__unravel_head_tl \scan_stop:
3329     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3330 }

```

(End definition for \\_\_unravel\_do\_math\_accent:.)

```

3331 \__unravel_new_tex_cmd:nn { accent } % 45
3332 {
3333     \__unravel_mode_non_vertical:n
3334     {
3335         \mode_if_math:TF
3336         { \__unravel_do_math_accent: } { \__unravel_do_accent: }
3337     }
3338 }
3339 \__unravel_new_tex_cmd:nn { math_accent } % 46
3340 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
3341 \__unravel_new_tex_cmd:nn { discretionary } % 47
3342 { \__unravel_not_implemented:n { discretionary } }

```

#### 2.13.4 Maths: from 48 to 56

- eq\_no=48
- left\_right=49
- math\_comp=50
- limit\_switch=51
- above=52
- math\_style=53
- math\_choice=54
- non\_script=55
- vcenter=56

```
3343 \_unravel_new_tex_cmd:nn { eq_no } % 48
3344 { \_unravel_not_implemented:n { eqno } }
3345 \_unravel_new_tex_cmd:nn { left_right } % 49
3346 { \_unravel_not_implemented:n { left/right } }
3347 \_unravel_new_tex_cmd:nn { math_comp } % 50
3348 { \_unravel_not_implemented:n { math~comp } }
3349 \_unravel_new_tex_cmd:nn { limit_switch } % 51
3350 { \_unravel_not_implemented:n { limits } }
3351 \_unravel_new_tex_cmd:nn { above } % 52
3352 { \_unravel_not_implemented:n { above } }
3353 \_unravel_new_tex_cmd:nn { math_style } % 53
3354 { \_unravel_not_implemented:n { math~style } }
3355 \_unravel_new_tex_cmd:nn { math_choice } % 54
3356 { \_unravel_not_implemented:n { math~choice } }
3357 \_unravel_new_tex_cmd:nn { non_script } % 55
3358 { \_unravel_not_implemented:n { non~script } }
3359 \_unravel_new_tex_cmd:nn { vcenter } % 56
3360 { \_unravel_not_implemented:n { vcenter } }
```

#### 2.13.5 From 57 to 70

- case\_shift=57
- message=58
- extension=59
- in\_stream=60
- begin\_group=61
- end\_group=62
- omit=63

- ex\_space=64
- no\_boundary=65
- radical=66
- end\_cs\_name=67
- char\_given=68
- math\_given=69
- last\_item=70

```

3361 \__unravel_new_tex_cmd:nn { case_shift } % 57
3362 {
3363   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3364   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3365   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3366   \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
3367 }
3368 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
3369 {
3370   #1 { \__unravel_back_input:n {#2} }
3371   \__unravel_print_action:x
3372   { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3373 }
3374 \__unravel_new_tex_cmd:nn { message } % 58
3375 {
3376   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3377   \__unravel_print_action:
3378   \__unravel_scan_toks_to_str:
3379   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3380   \tl_use:N \l__unravel_head_tl
3381   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3382 }

```

Extensions are implemented in a later section.

```

3383 \__unravel_new_tex_cmd:nn { extension } % 59
3384 {
3385   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3386   \__unravel_print_action:
3387   \__unravel_scan_extension_operands:
3388   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3389   \tl_use:N \l__unravel_head_tl \scan_stop:
3390   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3391 }
3392 \__unravel_new_tex_cmd:nn { in_stream } % 60
3393 {
3394   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3395   \__unravel_print_action:
3396   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3397   {
3398     \__unravel_scan_int:
3399     \__unravel_scan_optional_equals:

```

```

3400     \__unravel_scan_file_name:
3401     }
3402     { \__unravel_scan_int: }
3403     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3404     \tl_use:N \l__unravel_head_tl \scan_stop:
3405     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3406     }
3407 \__unravel_new_tex_cmd:nn { begin_group } % 61
3408 {
3409     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3410     \l__unravel_head_token
3411     \gtl_clear:N \l__unravel_after_group_gtl
3412     \__unravel_print_action:
3413 }
3414 \__unravel_new_tex_cmd:nn { end_group } % 62
3415 {
3416     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3417     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3418     \l__unravel_head_token
3419     \__unravel_print_action:
3420 }
3421 \__unravel_new_tex_cmd:nn { omit } % 63
3422 { \__unravel_not_implemented:n { omit } }
3423 \__unravel_new_tex_cmd:nn { ex_space } % 64
3424 {
3425     \__unravel_mode_non_vertical:n
3426     { \l__unravel_head_token \__unravel_print_action: }
3427 }
3428 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3429 {
3430     \__unravel_mode_non_vertical:n
3431     { \l__unravel_head_token \__unravel_print_action: }
3432 }
3433 \__unravel_new_tex_cmd:nn { radical } % 66
3434 {
3435     \__unravel_mode_math:n
3436     {
3437         \__unravel_prev_input_gpush:N \l__unravel_head_tl
3438         \__unravel_print_action:
3439         \__unravel_scan_int:
3440         \__unravel_scan_math:
3441         \__unravel_prev_input_gpop:N \l__unravel_head_tl
3442         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3443         \tl_use:N \l__unravel_head_tl \scan_stop:
3444         \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3445     }
3446 }
3447 \__unravel_new_tex_cmd:nn { end_cs_name } % 67
3448 {
3449     \__unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
3450     \__unravel_print_action:
3451 }

```

See the\_char and other\_char.

```

3452 \__unravel_new_tex_cmd:nn { char_given } % 68
3453 {
3454   \__unravel_mode_non_vertical:n
3455   {
3456     \mode_if_math:TF
3457     { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
3458     { \__unravel_char:V \l__unravel_head_char_int }
3459   }
3460 }

```

See math\_char\_num.

```

3461 \__unravel_new_tex_cmd:nn { math_given } % 69
3462 {
3463   \__unravel_mode_math:n
3464   { \__unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3465 }
3466 \__unravel_new_tex_cmd:nn { last_item } % 70
3467 { \__unravel_forbidden_case: }

```

## 2.13.6 Extensions

\\_\_unravel\_scan\_extension\_operands:

```

3468 \cs_new_protected_nopar:Npn \__unravel_scan_extension_operands:
3469 {
3470   \int_case:nnF \l__unravel_head_char_int
3471   {
3472     { 0 } % openout
3473     {
3474       \__unravel_scan_int:
3475       \__unravel_scan_optional_equals:
3476       \__unravel_scan_file_name:
3477     }
3478     { 1 } % write
3479     {
3480       \__unravel_scan_int:
3481       \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3482     }
3483     { 2 } % closeout
3484     { \__unravel_scan_int: }
3485     { 3 } % special
3486     { \__unravel_scan_toks_to_str: }
3487     { 4 } % immediate
3488     { \__unravel_scan_immediate_operands: }
3489     { 5 } % setlanguage
3490     {
3491       \mode_if_horizontal:TF
3492       { \__unravel_scan_int: }
3493       { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3494     }
3495     { 6 } % pdfliteral
3496     {
3497       \__unravel_scan_keyword:nF { dDiIrReEcCtT }

```

```

3498         { \__unravel_scan_keyword:n { pPaAgGeE } }
3499         \__unravel_scan_pdf_ext_toks:
3500     }
3501 { 7 } % pdfobj
3502 {
3503     \__unravel_scan_keyword:nTF
3504     { rReEsSeErRvVeEoObBjJnNuUmM }
3505     { \__unravel_skip_optional_space: }
3506     {
3507         \__unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3508         { \__unravel_scan_int: }
3509         \__unravel_scan_keyword:nT { sStTrReEaAmM }
3510         {
3511             \__unravel_scan_keyword:nT { aAtTtTrR }
3512             { \__unravel_scan_pdf_ext_toks: }
3513         }
3514         \__unravel_scan_keyword:n { fFiIlLeE }
3515         \__unravel_scan_pdf_ext_toks:
3516     }
3517 }
3518 { 8 } % pdfrefobj
3519 { \__unravel_scan_int: }
3520 { 9 } % pdfxform
3521 {
3522     \__unravel_scan_keyword:nT { aAtTtTrR }
3523     { \__unravel_scan_pdf_ext_toks: }
3524     \__unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3525     { \__unravel_scan_pdf_ext_toks: }
3526     \__unravel_scan_int:
3527 }
3528 { 10 } % pdfrefxform
3529 { \__unravel_scan_int: }
3530 { 11 } % pdfximage
3531 { \__unravel_scan_image: }
3532 { 12 } % pdfrefximage
3533 { \__unravel_scan_int: }
3534 { 13 } % pdfannot
3535 {
3536     \__unravel_scan_keyword:nTF
3537     { rReEsSeErRvVeEoObBjJnNuUmM }
3538     { \__unravel_scan_optional_space: }
3539     {
3540         \__unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3541         { \__unravel_scan_int: }
3542         \__unravel_scan_alt_rule:
3543         \__unravel_scan_pdf_ext_toks:
3544     }
3545 }
3546 { 14 } % pdfstartlink
3547 {
3548     \mode_if_vertical:TF
3549     { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } { } }
3550     {
3551         \__unravel_scan_rule_attr:

```

```

3552         \_unravel_scan_action:
3553     }
3554 }
3555 { 15 } % pdfendlink
3556 {
3557     \mode_if_vertical:T
3558     { \_unravel_error:nnnn { invalid-mode } { } { } { } { } }
3559 }
3560 { 16 } % pdfoutline
3561 {
3562     \_unravel_scan_keyword:nT { aAtTtTrR }
3563     { \_unravel_scan_pdf_ext_toks: }
3564     \_unravel_scan_action:
3565     \_unravel_scan_keyword:nT { cCoOuUnNtT }
3566     { \_unravel_scan_int: }
3567     \_unravel_scan_pdf_ext_toks:
3568 }
3569 { 17 } % pdfdest
3570 { \_unravel_scan_pdfdest_operands: }
3571 { 18 } % pdfthread
3572 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3573 { 19 } % pdfstartthread
3574 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3575 { 20 } % pdfendthread
3576 { }
3577 { 21 } % pdfsavepos
3578 { }
3579 { 22 } % pdfinfo
3580 { \_unravel_scan_pdf_ext_toks: }
3581 { 23 } % pdfcatalog
3582 {
3583     \_unravel_scan_pdf_ext_toks:
3584     \_unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
3585     { \_unravel_scan_action: }
3586 }
3587 { 24 } % pdfnames
3588 { \_unravel_scan_pdf_ext_toks: }
3589 { 25 } % pdffontattr
3590 {
3591     \_unravel_scan_font_ident:
3592     \_unravel_scan_pdf_ext_toks:
3593 }
3594 { 26 } % pdfincludechars
3595 {
3596     \_unravel_scan_font_ident:
3597     \_unravel_scan_pdf_ext_toks:
3598 }
3599 { 27 } % pdfmapfile
3600 { \_unravel_scan_pdf_ext_toks: }
3601 { 28 } % pdfmapline
3602 { \_unravel_scan_pdf_ext_toks: }
3603 { 29 } % pdftrailer
3604 { \_unravel_scan_pdf_ext_toks: }
3605 { 30 } % pdfresettimer

```

```

3606     { }
3607 { 31 } % pdffontexpand
3608     {
3609         \__unravel_scan_font_ident:
3610         \__unravel_scan_optional_equals:
3611         \__unravel_scan_int:
3612         \__unravel_scan_int:
3613         \__unravel_scan_int:
3614         \__unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3615         { \__unravel_skip_optional_space: }
3616     }
3617 { 32 } % pdfsetrandomseed
3618     { \__unravel_scan_int: }
3619 { 33 } % pdfsnaprefpoint
3620     { }
3621 { 34 } % pdfsnapy
3622     { \__unravel_scan_normal_glue: }
3623 { 35 } % pdfsnapycomp
3624     { \__unravel_scan_int: }
3625 { 36 } % pdfglyphtounicode
3626     {
3627         \__unravel_scan_pdf_ext_toks:
3628         \__unravel_scan_pdf_ext_toks:
3629     }
3630 { 37 } % pdfcolorstack
3631     { \__unravel_scan_pdfcolorstack_operands: }
3632 { 38 } % pdfsetmatrix
3633     { \__unravel_scan_pdf_ext_toks: }
3634 { 39 } % pdfsave
3635     { }
3636 { 40 } % pdfrestore
3637     { }
3638 { 41 } % pdfnobluiltintounicode
3639     { \__unravel_scan_font_ident: }
3640 }
3641 { } % no other cases.
3642 }

```

(End definition for \\_\_unravel\_scan\_extension\_operands:.)

\\_\_unravel\_scan\_pdfcolorstack\_operands:

```

3643 \cs_new_protected_nopar:Npn \__unravel_scan_pdfcolorstack_operands:
3644 {
3645     \__unravel_scan_int:
3646     \__unravel_scan_keyword:nF { sSeEtT }
3647     {
3648         \__unravel_scan_keyword:nF { pPuUsShH }
3649         {
3650             \__unravel_scan_keyword:nF { pPoOpP }
3651             {
3652                 \__unravel_scan_keyword:nF { cCuUrRrReEnNtT }
3653                 {
3654                     \__unravel_error:nnnnn { color-stack-action-missing }
3655                     { } { } { } { } { }

```



```

3656         }
3657     }
3658 }
3659 }
3660 }

```

(End definition for \\_unravel\_scan\_pdfcolorstack\_operands:.)

\\_unravel\_scan\_rule\_attr:

```

3661 \cs_new_protected_nopar:Npn \_unravel_scan_rule_attr:
3662 {
3663     \_unravel_scan_alt_rule:
3664     \_unravel_scan_keyword:nT { aAtTtTrR }
3665     { \_unravel_scan_pdf_ext_toks: }
3666 }

```

(End definition for \\_unravel\_scan\_rule\_attr:.)

\\_unravel\_scan\_action:

```

3667 \cs_new_protected_nopar:Npn \_unravel_scan_action:
3668 {
3669     \_unravel_scan_keyword:nTF { uUsSeErR }
3670     { \_unravel_scan_pdf_ext_toks: }
3671     {
3672         \_unravel_scan_keyword:nF { gGoOtToO }
3673         {
3674             \_unravel_scan_keyword:nF { tThHrReEaAdD }
3675             { \_unravel_error:nnnnn { action-type-missing } { } { } { } { } }
3676         }
3677     }
3678     \_unravel_scan_keyword:nT { fFiIlLeE }
3679     { \_unravel_scan_pdf_ext_toks: }
3680     \_unravel_scan_keyword:nTF { pPaAgGeE }
3681     {
3682         \_unravel_scan_int:
3683         \_unravel_scan_pdf_ext_toks:
3684     }
3685     {
3686         \_unravel_scan_keyword:nTF { nNaAmMeE }
3687         { \_unravel_scan_pdf_ext_toks: }
3688         {
3689             \_unravel_scan_keyword:nTF { nNuUmM }
3690             { \_unravel_scan_int: }
3691             { \_unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
3692         }
3693     }
3694     \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3695     { \_unravel_skip_optional_space: }
3696     {
3697         \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3698         { \_unravel_skip_optional_space: }
3699     }
3700 }

```

(End definition for \\_unravel\_scan\_action:.)

`\_unravel_scan_image:` Used by `\pdfximage`.

```
3701 \cs_new_protected_nopar:Npn \_unravel_scan_image:
3702 {
3703   \_unravel_scan_rule_attr:
3704   \_unravel_scan_keyword:nTF { nNaAmMeEdD }
3705   { \_unravel_scan_pdf_ext_toks: }
3706   {
3707     \_unravel_scan_keyword:nT { pPaAgGeE }
3708     { \_unravel_scan_int: }
3709   }
3710   \_unravel_scan_keyword:nT { cCoOlLoOrRsSpPaAcCeE }
3711   { \_unravel_scan_int: }
3712   \_unravel_scan_pdf_ext_toks:
3713 }
```

*(End definition for \\_unravel\_scan\_image:.)*

`\_unravel_scan_immediate_operands:`

```
3714 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands:
3715 {
3716   \_unravel_get_x_next:
3717   \_unravel_set_cmd:
3718   \int_compare:nNnTF
3719   \l__unravel_head_cmd_int = { \_unravel_tex_use:n { extension } }
3720   {
3721     \int_compare:nNnTF
3722     \l__unravel_head_char_int < { 3 } % openout, write, closeout
3723     { \_unravel_scan_immediate_operands_aux: }
3724     {
3725       \int_case:nnF \l__unravel_head_char_int
3726       {
3727         { 7 } { \_unravel_scan_extension_operands_aux: } % pdfobj
3728         { 9 } { \_unravel_scan_extension_operands_aux: } % pdfxform
3729         { 11 } { \_unravel_scan_extension_operands_aux: } %pdfximage
3730       }
3731       { \_unravel_scan_immediate_operands_bad: }
3732     }
3733   }
3734   { \_unravel_scan_immediate_operands_bad: }
3735 }
3736 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_aux:
3737 {
3738   \_unravel_prev_input:V \l__unravel_head_tl
3739   \_unravel_scan_extension_operands:
3740 }
3741 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_bad:
3742 {
3743   \_unravel_back_input:
3744   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3745   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3746   \_unravel_prev_input_gpush:
3747 }
3748
```

*(End definition for \\_unravel\_scan\_immediate\_operands:.)*

\\_unravel\_scan\_pdfdest\_operands:

```
3749 \cs_new_protected_nopar:Npn \_unravel_scan_pdfdest_operands:
3750 {
3751   \_unravel_scan_keyword:nTF { nNuUmM }
3752   { \_unravel_scan_int: }
3753   {
3754     \_unravel_scan_keyword:nTF { nNaAmMeE }
3755     { \_unravel_scan_pdf_ext_toks: }
3756     { \_unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
3757   }
3758   \_unravel_scan_keyword:nTF { xXyYzZ }
3759   {
3760     \_unravel_scan_keyword:nT { zZoOoOmM }
3761     { \_unravel_scan_int: }
3762   }
3763   {
3764     \_unravel_scan_keyword:nF { fFiItTbBhH }
3765     {
3766       \_unravel_scan_keyword:nF { fFiItTbBvV }
3767       {
3768         \_unravel_scan_keyword:nF { fFiItTbB }
3769         {
3770           \_unravel_scan_keyword:nF { fFiItThHhH }
3771           {
3772             \_unravel_scan_keyword:nF { fFiItTvV }
3773             {
3774               \_unravel_scan_keyword:nTF
3775               { fFiItTrR }
3776               {
3777                 \_unravel_skip_optional_space:
3778                 \_unravel_scan_alt_rule:
3779                 \use_none:n
3780               }
3781               {
3782                 \_unravel_scan_keyword:nF
3783                 { fFiItT }
3784                 {
3785                   \_unravel_error:nnnnn { destination-type-missing }
3786                   { } { } { } { } { }
3787                 }
3788               }
3789             }
3790           }
3791         }
3792       }
3793     }
3794   }
3795   \_unravel_skip_optional_space:
3796 }
```

(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`.

```

3797 \cs_set_protected_nopar:Npn \__unravel_tmp:w
3798 {
3799   \__unravel_prev_input_gpush:
3800   \__unravel_prefixed_command:
3801 }
3802 \int_step_inline:nnnn
3803 { \__unravel_tex_use:n { max_non_prefixed_command } + 1 }
3804 { 1 }
3805 { \__unravel_tex_use:n { max_command } }
3806 { \cs_new_eq:cN { __unravel_cmd_#1: } \__unravel_tmp:w }

```

`\__unravel_prefixed_command:` Accumulated prefix codes so far are stored as the last item of the previous-input sequence.

```

3807 \cs_new_protected_nopar:Npn \__unravel_prefixed_command:
3808 {
3809   \int_while_do:nNnn
3810   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
3811   {
3812     \__unravel_prev_input:V \l__unravel_head_tl
3813     \__unravel_get_x_non_relax:
3814     \__unravel_set_cmd:
3815     \int_compare:nNnF \l__unravel_head_cmd_int
3816     > { \__unravel_tex_use:n { max_non_prefixed_command } }
3817     {
3818       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3819       \__unravel_error:nxxxx { erroneous-prefixes }
3820       { \tl_to_str:N \l__unravel_tmpa_tl }
3821       { \tl_to_str:N \l__unravel_head_tl }
3822       { } { }
3823       \__unravel_back_input:
3824       \__unravel_omit_after_assignment:w
3825     }
3826   }
3827   % ^^A todo: Discard non-\global prefixes if they are irrelevant
3828   % ^^A todo: Adjust for the setting of \globaldefs
3829   \cs_if_exist_use:cF
3830   { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3831   {
3832     \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
3833     \__unravel_omit_after_assignment:w
3834   }
3835   \__unravel_after_assignment:
3836 }

```

(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
3837 \cs_new_protected_nopar:Npn \__unravel_after_assignment:
3838 {
3839   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3840   \gtl_gclear:N \g__unravel_after_assignment_gtl
3841 }
3842 \cs_new_protected_nopar:Npn \__unravel_omit_after_assignment:w
3843   #1 \__unravel_after_assignment: { }

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

\__unravel_prefixed_new:nn
3844 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3845 {
3846   \cs_new_protected_nopar:cpn
3847     { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3848 }

(End definition for \__unravel_prefixed_new:nn.)

\__unravel_assign_token:n
3849 \cs_new_protected:Npn \__unravel_assign_token:n #1
3850 {
3851   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3852   #1
3853   \tl_use:N \l__unravel_head_tl \scan_stop:
3854   \__unravel_print_assigned_token:
3855 }

(End definition for \__unravel_assign_token:n.)

\__unravel_assign_register:
3856 \cs_new_protected_nopar:Npn \__unravel_assign_register:
3857 {
3858   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3859   \tl_use:N \l__unravel_head_tl \scan_stop:
3860   \__unravel_print_assigned_register:
3861 }

(End definition for \__unravel_assign_register:.)

\__unravel_assign_value:nn
3862 \cs_new_protected:Npn \__unravel_assign_value:nn #1#2
3863 {
3864   \tl_if_empty:nF {#1}
3865   {
3866     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3867     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3868     #1
3869     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3870   }
3871   \__unravel_prev_input:V \l__unravel_head_tl
3872   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3873   \__unravel_scan_optional_equals:
3874   #2

```

```

3875     \_unravel_assign_register:
3876 }

```

(End definition for \\_unravel\_assign\_value:nn.)

\\_unravel\_assign\_toks:

```

3877 \_unravel_prefixed_new:nn { toks_register } % 71
3878 {
3879   \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3880   { % \toks
3881     \_unravel_prev_input_gpush:N \l__unravel_head_tl
3882     \_unravel_print_action:
3883     \_unravel_scan_int:
3884     \_unravel_prev_input_gpop:N \l__unravel_head_tl
3885   }
3886   \_unravel_assign_toks:
3887 }
3888 \_unravel_prefixed_new:nn { assign_toks } % 72
3889 { \_unravel_assign_toks: }
3890 \cs_new_protected_nopar:Npn \_unravel_assign_toks:
3891 {
3892   \_unravel_prev_input_silent:V \l__unravel_head_tl
3893   \_unravel_print_action:
3894   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3895   \_unravel_scan_optional_equals:
3896   \_unravel_get_x_non_relax:
3897   \_unravel_set_cmd:
3898   \int_compare:nNnTF
3899     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { toks_register } }
3900   {
3901     \_unravel_prev_input:V \l__unravel_head_tl
3902     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3903     { \_unravel_scan_int: }
3904   }
3905   {
3906     \int_compare:nNnTF
3907       \l__unravel_head_cmd_int = { \_unravel_tex_use:n { assign_toks } }
3908     { \_unravel_prev_input:V \l__unravel_head_tl }
3909     {
3910       \_unravel_back_input:
3911       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3912     }
3913   }
3914   \_unravel_assign_register:
3915 }

```

(End definition for \\_unravel\_assign\_toks:.)

```

3916 \_unravel_prefixed_new:nn { assign_int } % 73
3917 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
3918 \_unravel_prefixed_new:nn { assign_dimen } % 74
3919 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
3920 \_unravel_prefixed_new:nn { assign_glue } % 75
3921 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_glue: } }
3922 \_unravel_prefixed_new:nn { assign_mu_glue } % 76

```

```

3923 { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_mu\_glue: } }
3924 \_unravel\_prefixed\_new:nn { assign\_font\_dimen } % 77
3925 {
3926   \_unravel\_assign\_value:nn
3927   { \_unravel\_scan\_int: \_unravel\_scan\_font\_ident: }
3928   { \_unravel\_scan\_normal\_dimen: }
3929 }
3930 \_unravel\_prefixed\_new:nn { assign\_font\_int } % 78
3931 {
3932   \_unravel\_assign\_value:nn
3933   { \_unravel\_scan\_font\_int: } { \_unravel\_scan\_int: }
3934 }
3935 \_unravel\_prefixed\_new:nn { set\_aux } % 79
3936 { % prevdepth = 1, spacefactor = 102
3937   \int\_compare:nNnTF \l\_unravel\_head\_char\_int = \c\_one
3938   { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_normal\_dimen: } }
3939   { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_int: } }
3940 }
3941 \_unravel\_prefixed\_new:nn { set\_prev\_graf } % 80
3942 { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_int: } }
3943 \_unravel\_prefixed\_new:nn { set\_page\_dimen } % 81
3944 { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_normal\_dimen: } }
3945 \_unravel\_prefixed\_new:nn { set\_page\_int } % 82
3946 { \_unravel\_assign\_value:nn { } { \_unravel\_scan\_int: } }
3947 \_unravel\_prefixed\_new:nn { set\_box\_dimen } % 83
3948 {
3949   \_unravel\_assign\_value:nn
3950   { \_unravel\_scan\_int: } { \_unravel\_scan\_normal\_dimen: }
3951 }
3952 \_unravel\_prefixed\_new:nn { set\_shape } % 84
3953 {
3954   \_unravel\_assign\_value:nn { \_unravel\_scan\_int: }
3955   {
3956     \prg\_replicate:nn
3957     {
3958       \tl\_if\_head\_eq\_meaning:VNT
3959       \l\_unravel\_defined\_tl \tex\_parshape:D { \c\_two * }
3960       \tl\_tail:N \l\_unravel\_defined\_tl
3961     }
3962     { \_unravel\_scan\_int: }
3963   }
3964 }
3965 \_unravel\_prefixed\_new:nn { def\_code } % 85
3966 {
3967   \_unravel\_assign\_value:nn
3968   { \_unravel\_scan\_int: } { \_unravel\_scan\_int: }
3969 }
3970 \_unravel\_prefixed\_new:nn { def\_family } % 86
3971 {
3972   \_unravel\_assign\_value:nn
3973   { \_unravel\_scan\_int: } { \_unravel\_scan\_font\_ident: }
3974 }
3975 \_unravel\_prefixed\_new:nn { set\_font } % 87

```

```

3976 {
3977   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3978   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3979   \tl_use:N \l__unravel_head_tl \scan_stop:
3980   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3981   \__unravel_print_action:
3982 }
3983 \__unravel_prefixed_new:nn { def_font } % 88
3984 {
3985   \__unravel_prev_input_silent:V \l__unravel_head_tl
3986   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3987   \__unravel_scan_r_token:
3988   \__unravel_print_action:x
3989   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
3990   \__unravel_scan_optional_equals:
3991   \__unravel_scan_file_name:
3992   \bool_gset_true:N \g__unravel_name_in_progress_bool
3993   \__unravel_scan_keyword:nTF { aAtT }
3994   { \__unravel_scan_normal_dimen: }
3995   {
3996     \__unravel_scan_keyword:nT { sScCaAlLeEdD }
3997     { \__unravel_scan_int: }
3998   }
3999   \bool_gset_false:N \g__unravel_name_in_progress_bool
4000   \__unravel_assign_token:n { }
4001 }

```

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

let, futurelet

```

4002 \__unravel_prefixed_new:nn { let } % 94
4003 {
4004   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4005   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4006   { % |let|
4007     \__unravel_scan_r_token:
4008     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4009     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4010     \__unravel_get_next:
4011     \bool_while_do:nn
4012     { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4013     { \__unravel_get_next: }
4014     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4015     { \__unravel_get_next: }
4016     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4017     { \__unravel_get_next: }
4018   }
4019   { % |futurelet|
4020     \__unravel_scan_r_token:
4021     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4022     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4023     \__unravel_get_next:
4024     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4025     \__unravel_get_next:

```



```

4026     \__unravel_back_input:
4027     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4028     \__unravel_back_input:
4029   }
4030   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4031   \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4032   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4033   \__unravel_exp_args:Nx \use:n
4034   {
4035     \exp_not:V \l__unravel_head_tl
4036     \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4037   }
4038   \__unravel_print_assigned_token:
4039 }
4040 \__unravel_prefixed_new:nn { shorthand_def } % 95
4041 {
4042   \__unravel_prev_input_silent:V \l__unravel_head_tl
4043   \tl_set:Nx \l__unravel_prev_action_tl
4044   { \tl_to_str:N \l__unravel_head_tl }
4045   \__unravel_scan_r_token:
4046   \__unravel_print_action:x
4047   { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4048   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4049   \__unravel_scan_optional_equals:
4050   \__unravel_scan_int:
4051   \__unravel_assign_token:n { }
4052 }

```

`\__unravel_read_to_cs_safe:nTF`  
`\__unravel_read_to_cs_safe:fTF`

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

```

4053 \__unravel_prefixed_new:nn { read_to_cs } % 96
4054 {
4055   \__unravel_prev_input_silent:V \l__unravel_head_tl
4056   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4057   \__unravel_scan_int:
4058   \__unravel_scan_to:
4059   \__unravel_scan_r_token:
4060   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4061   \__unravel_read_to_cs_safe:fTF
4062   { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
4063   { \__unravel_assign_token:n { } }
4064   {
4065     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4066     \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4067   }
4068 }
4069 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
4070 {
4071   \int_compare:nNnTF { \etex_interactionmode:D } > { 1 }

```

```

4072     { \prg_return_true: }
4073     {
4074         \int_compare:nNnTF {#1} < { 0 }
4075         { \prg_return_false: }
4076         {
4077             \int_compare:nNnTF {#1} > { 15 }
4078             { \prg_return_false: }
4079             {
4080                 \tex_ifeof:D #1 \exp_stop_f:
4081                 \prg_return_false:
4082                 \else:
4083                 \prg_return_true:
4084                 \fi:
4085             }
4086         }
4087     }
4088 }
4089 \cs_generate_variant:Nn \_unravel_read_to_cs_safe:nTF { f }

(End definition for \_unravel_read_to_cs_safe:nTF.)

4090 \_unravel_prefixed_new:nn { def } % 97
4091 {
4092     \_unravel_prev_input_get:N \l__unravel_tmpa_tl
4093     \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4094     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4095     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4096     \int_compare:nNnTF \l__unravel_head_char_int < \c_two
4097     { % def/gdef
4098         \_unravel_scan_r_token:
4099         \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4100         \_unravel_scan_toks:NN \c_true_bool \c_false_bool
4101     }
4102     { % edef/xdef
4103         \_unravel_scan_r_token:
4104         \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4105         \_unravel_scan_toks:NN \c_true_bool \c_true_bool
4106     }
4107     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4108     \_unravel_prev_input:V \l__unravel_head_tl
4109     \_unravel_assign_token:n
4110     { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4111 }

```

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

```

4112 \_unravel_prefixed_new:nn { set_box } % 98
4113 {
4114     \_unravel_prev_input:V \l__unravel_head_tl
4115     \_unravel_scan_int:
4116     \_unravel_scan_optional_equals:
4117     \bool_if:NTF \g__unravel_set_box_allowed_bool
4118     { \_unravel_do_box:N \c_false_bool }
4119     {

```

```

4120         \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
4121         \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4122         \__unravel_omit_after_assignment:w
4123     }
4124 }

\hyphenation and \patterns
4125 \__unravel_prefixed_new:nn { hyph_data } % 99
4126 {
4127     \__unravel_prev_input:V \l__unravel_head_tl
4128     \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4129     \__unravel_assign_token:n { }
4130 }

4131 \__unravel_prefixed_new:nn { set_interaction } % 100
4132 {
4133     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4134     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4135     \tl_use:N \l__unravel_head_tl \scan_stop:
4136     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4137 }

4138 \__unravel_prefixed_new:nn { letterspace_font } % 101
4139 {
4140     \__unravel_prev_input_silent:V \l__unravel_head_tl
4141     \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4142     \__unravel_scan_r_token:
4143     \__unravel_print_action:x
4144     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4145     \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4146     \__unravel_scan_optional_equals:
4147     \__unravel_scan_font_ident:
4148     \__unravel_scan_int:
4149     \__unravel_assign_token:n { }
4150 }

4151 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4152 {
4153     \__unravel_prev_input_silent:V \l__unravel_head_tl
4154     \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4155     \__unravel_scan_r_token:
4156     \__unravel_print_action:x
4157     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4158     \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4159     \__unravel_scan_optional_equals:
4160     \__unravel_scan_font_ident:
4161     \__unravel_assign_token:n { }
4162 }

```

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

```

4163 \__unravel_prefixed_new:nn { register } % 89
4164 { \__unravel_do_register:N \c_zero }
4165 \__unravel_prefixed_new:nn { advance } % 90
4166 { \__unravel_do_operation:N \c_one }
4167 \__unravel_prefixed_new:nn { multiply } % 91

```

```

4168 { \_unravel\_do\_operation:N \c\_two }
4169 \_unravel\_prefixed\_new:nn { divide } % 92
4170 { \_unravel\_do\_operation:N \c\_three }

```

\\_unravel\\_do\\_operation:N  
 \\_unravel\\_do\\_operation\\_fail:w

```

4171 \cs\_new\_protected:Npn \_unravel\_do\_operation:N #1
4172 {
4173   \_unravel\_prev\_input\_silent:V \l\_unravel\_head\_tl
4174   \_unravel\_print\_action:
4175   \_unravel\_get\_x\_next:
4176   \_unravel\_set\_cmd:
4177   \int\_compare:nNnTF
4178     \l\_unravel\_head\_cmd\_int > { \_unravel\_tex\_use:n { assign\_mu\_glue } }
4179   {
4180     \int\_compare:nNnTF
4181       \l\_unravel\_head\_cmd\_int = { \_unravel\_tex\_use:n { register } }
4182       { \_unravel\_do\_register:N #1 }
4183       { \_unravel\_do\_operation\_fail:w }
4184     }
4185   {
4186     \int\_compare:nNnTF
4187       \l\_unravel\_head\_cmd\_int < { \_unravel\_tex\_use:n { assign\_int } }
4188       { \_unravel\_do\_operation\_fail:w }
4189     {
4190       \_unravel\_prev\_input:V \l\_unravel\_head\_tl
4191       \exp\_args:NNf \_unravel\_do\_register\_set:Nn #1
4192       {
4193         \int\_eval:n
4194         {
4195           \l\_unravel\_head\_cmd\_int
4196           - \_unravel\_tex\_use:n { assign\_toks }
4197         }
4198       }
4199     }
4200   }
4201 }
4202 \cs\_new\_protected\_nopar:Npn \_unravel\_do\_operation\_fail:w
4203 {
4204   \_unravel\_error:nnnnn { after-advance } { } { } { } { }
4205   \_unravel\_prev\_input\_gpop:N \l\_unravel\_tmpa\_tl
4206   \_unravel\_omit\_after\_assignment:w
4207 }

```

(End definition for \\_unravel\\_do\\_operation:N and \\_unravel\\_do\\_operation\\_fail:w.)

\\_unravel\\_do\\_register:N  
 \\_unravel\\_do\\_register\\_aux:Nn

```

4208 \cs\_new\_protected:Npn \_unravel\_do\_register:N #1
4209 {
4210   \exp\_args:NNV \_unravel\_do\_register\_aux:Nn #1
4211   \l\_unravel\_head\_char\_int
4212 }
4213 \cs\_new\_protected:Npn \_unravel\_do\_register\_aux:Nn #1#2
4214 {
4215   \int\_compare:nNnTF { \tl\_tail:n {#2} } = \c\_zero

```

```

4216 {
4217   \_unravel_prev_input_gpush:N \l__unravel_head_tl
4218   \_unravel_print_action:
4219   \_unravel_scan_int:
4220   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4221   \_unravel_prev_input_silent:V \l__unravel_head_tl
4222 }
4223 {
4224   \_unravel_prev_input_silent:V \l__unravel_head_tl
4225   \_unravel_print_action:
4226 }
4227 \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4228 \exp_args:NNf \_unravel_do_register_set:Nn #1
4229 { \int_eval:n { #2 / 1 000 000 } }
4230 }

```

(End definition for \\_unravel\_do\_register:N and \\_unravel\_do\_register\_aux:Nn.)

\\_unravel\_do\_register\_set:Nn

```

4231 \cs_new_protected:Npn \_unravel_do_register_set:Nn #1#2
4232 {
4233   \int_compare:nNnTF {#1} = \c_zero
4234   { % truly register command
4235     \_unravel_scan_optional_equals:
4236   }
4237   { % \advance, \multiply, \divide
4238     \_unravel_scan_keyword:nF { bByY }
4239     { \_unravel_prev_input_silent:n { by } }
4240   }
4241   \int_compare:nNnTF {#1} < \c_two
4242   {
4243     \int_case:nnF {#2}
4244     {
4245       { 1 } { \_unravel_scan_int: } % count
4246       { 2 } { \_unravel_scan_normal_dimen: } % dim
4247       { 3 } { \_unravel_scan_normal_glue: } % glue
4248       { 4 } { \_unravel_scan_mu_glue: } % muglue
4249     }
4250     { \_unravel_error:nxxxx { internal } { do-reg=#2 } { } { } { } }
4251   }
4252   { \_unravel_scan_int: }
4253   \_unravel_assign_register:
4254 }

```

(End definition for \\_unravel\_do\_register\_set:Nn.)

The following is used for instance when making accents.

```

4255 \cs_new_protected_nopar:Npn \_unravel_do_assignments:
4256 {
4257   \_unravel_get_x_non_relax:
4258   \_unravel_set_cmd:
4259   \int_compare:nNnT
4260   \l__unravel_head_cmd_int
4261   > { \_unravel_tex_use:n { max_non_prefixed_command } }
4262   {

```

```

4263         \bool_gset_false:N \g__unravel_set_box_allowed_bool
4264         \__unravel_prev_input_gpush:
4265         \__unravel_prefixed_command:
4266         \bool_gset_true:N \g__unravel_set_box_allowed_bool
4267         \__unravel_do_assignments:
4268     }
4269 }

```

## 2.14 Expandable primitives

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

- `undefined_cs=103` for undefined control sequences (not quite a primitive).
- `expand_after=104` for `\expandafter` and `\unless`.
- `no_expand=105` for `\noexpand` and `\pdfprimitive`.
- `input=106` for `\input`, `\endinput` and `\scantokens`.
- `if_test=107` for the conditionals, `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifpdfprimitive`, `\ifpdfabsnum`, and `\ifpdfabsdim`.
- `fi_or_else=108` for `\fi`, `\else` and `\or`.
- `cs_name=109` for `\csname`.
- `convert=110` for `\number`, `\romannumeral`, `\string`, `\meaning`, `\fontname`, `\eTeXrevision`, `\pdftexrevision`, `\pdftexbanner`, `\pdffontname`, `\pdffontobjnum`, `\pdffontsize`, `\pdfpageref`, `\pdfxformname`, `\pdfescapestring`, `\pdfescapename`, `\leftmarginkern`, `\rightmarginkern`, `\pdfstrcmp`, `\pdfcolorstackinit`, `\pdfescapehex`, `\pdfunescapehex`, `\pdfcreationdate`, `\pdffilemoddate`, `\pdffilesize`, `\pdfmdfivesum`, `\pdffiledump`, `\pdfmatch`, `\pdflastmatch`, `\pdfuniformdeviate`, `\pdfnormaldeviate`, `\pdfinsertht`, `\pdfximagebbox`, and `\jobname`.
- `the=111` for `\the`, `\unexpanded`, and `\detokenize`.
- `top_bot_mark=112` `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`, `\topmarks`, `\firstmarks`, `\botmarks`, `\splitfirstmarks`, and `\splitbotmarks`.
- `call=113` for macro calls, implemented by `\__unravel_macro_call:`.
- `end_template=117` for T<sub>E</sub>X's end template.

Let T<sub>E</sub>X trigger an error.

```

4270 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
4271 { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }

\__unravel_expandafter:
\__unravel_unless:
4272 \__unravel_new_tex_expandable:nn { expand_after } % 104
\__unravel_unless_bad:
4273 {
4274     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4275     { \__unravel_expandafter: } { \__unravel_unless: }

```

```

4276 }
4277 \cs_new_protected_nopar:Npn \__unravel_expandafter:
4278 {
4279   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4280   \__unravel_get_next:
4281   \gtl_concat:NNN \l__unravel_head_gtl
4282     \l__unravel_tmpb_gtl \l__unravel_head_gtl
4283   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4284   \__unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
4285   \__unravel_get_next:
4286   \__unravel_token_if_expandable:NTF \l__unravel_head_token
4287     { \__unravel_expand: }
4288     { \__unravel_back_input: }
4289   \__unravel_prev_input_gpop:N \l__unravel_head_gtl
4290   \__unravel_set_action_text:x
4291     { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4292   \gtl_pop_left:N \l__unravel_head_gtl
4293   \__unravel_back_input:
4294   \__unravel_print_action:
4295 }
4296 \cs_new_protected_nopar:Npn \__unravel_unless:
4297 {
4298   \__unravel_get_token:
4299   \int_compare:nNnTF
4300     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4301     {
4302       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4303         { \__unravel_unless_bad: }
4304         {
4305           \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4306           % \int_add:Nn \l__unravel_head_char_int { 32 }
4307           \__unravel_expand_nonmacro:
4308         }
4309       }
4310     { \__unravel_unless_bad: }
4311 }
4312 \cs_new_protected_nopar:Npn \__unravel_unless_bad:
4313 {
4314   \__unravel_error:nnnnn { bad-unless } { } { } { } { }
4315   \__unravel_back_input:
4316 }

```

(End definition for \\_\_unravel\_expandafter:, \\_\_unravel\_unless:, and \\_\_unravel\_unless\_bad:.)

```

\__unravel_noexpand:
\__unravel_pdfprimitive: 4317 \__unravel_new_tex_expandable:nn { no_expand } % 105
4318 {
4319   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4320     { \__unravel_noexpand: }
4321     { \__unravel_pdfprimitive: }
4322 }
4323 \cs_new_protected_nopar:Npn \__unravel_noexpand:
4324 {
4325   \__unravel_get_token:

```

```

4326 \__unravel_back_input:
4327 \__unravel_token_if_expandable:NT \l__unravel_head_token
4328 {
4329   \cs_gset_protected_nopar:Npx \__unravel_get_next:
4330   {
4331     \cs_gset_protected_nopar:Npn \__unravel_get_next:
4332     { \exp_not:o { \__unravel_get_next: } }
4333     \exp_not:o { \__unravel_get_next: }
4334     \exp_not:n { \cs_set_eq:NN \l__unravel_head_token \tex_relax:D }
4335   }
4336 }
4337 }
4338 \cs_new_protected_nopar:Npn \__unravel_pdfprimitive:
4339 { \__unravel_not_implemented:n { pdfprimitive } }

```

(End definition for \\_\_unravel\_noexpand: and \\_\_unravel\_pdfprimitive:.)

```

\__unravel_endinput:
\__unravel_scantokens:
\__unravel_input:
4340 \__unravel_new_tex_expandable:nn { input } % 106
4341 {
4342   \int_case:nnF \l__unravel_head_char_int
4343   {
4344     { 1 } { \__unravel_endinput: } % \endinput
4345     { 2 } { \__unravel_scantokens: } % \scantokens
4346   }
4347   { % 0=\input
4348     \bool_if:NTF \g__unravel_name_in_progress_bool
4349     { \__unravel_insert_relax: } { \__unravel_input: }
4350   }
4351 }
4352 \cs_new_protected_nopar:Npn \__unravel_endinput:
4353 {
4354   \group_begin:
4355   \msg_warning:nn { unravel } { endinput-ignored }
4356   \group_end:
4357   \__unravel_print_action:
4358 }
4359 \cs_new_protected_nopar:Npn \__unravel_scantokens:
4360 {
4361   \__unravel_prev_input_gpush:
4362   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4363   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4364   \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4365   \__unravel_back_input:V \l__unravel_head_tl
4366   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4367 }
4368 \cs_new_protected_nopar:Npn \__unravel_input:
4369 {
4370   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4371   \__unravel_scan_file_name:
4372   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4373   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4374   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4375   \__unravel_back_input:V \l__unravel_tmpa_tl

```



```

4376   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4377   }

```

(End definition for \\_unravel\_endinput:, \\_unravel\_scantokens:, and \\_unravel\_input:.)

\\_unravel\_csname\_loop:

```

4378   \_unravel_new_tex_expandable:nn { cs_name } % 109
4379   {
4380     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4381     \_unravel_print_action:
4382     \_unravel_csname_loop:
4383     \_unravel_prev_input_silent:V \l__unravel_head_tl
4384     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4385     \_unravel_back_input_tl_o:
4386   }
4387   \cs_new_protected_nopar:Npn \_unravel_csname_loop:
4388   {
4389     \_unravel_get_x_next:
4390     \_unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4391     {
4392       \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4393       {
4394         \_unravel_back_input:
4395         \_unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4396         \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4397       }
4398     }
4399     {
4400       \_unravel_prev_input_silent:x
4401       { \_unravel_token_to_char:N \l__unravel_head_token }
4402       \_unravel_csname_loop:
4403     }
4404   }

```

(End definition for \\_unravel\_csname\_loop:.)

```

4405   \_unravel_new_tex_expandable:nn { convert } % 110
4406   {
4407     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4408     \_unravel_print_action:
4409     \int_case:nn \l__unravel_head_char_int
4410     {
4411       0      \_unravel_scan_int:
4412       1      \_unravel_scan_int:
4413       2 { \_unravel_get_next: \_unravel_prev_input:V \l__unravel_head_tl }
4414       3 { \_unravel_get_next: \_unravel_prev_input:V \l__unravel_head_tl }
4415       4      \_unravel_scan_font_ident:
4416       8      \_unravel_scan_font_ident:
4417       9      \_unravel_scan_font_ident:
4418       { 10 } \_unravel_scan_font_ident:
4419       { 11 } \_unravel_scan_int:
4420       { 12 } \_unravel_scan_int:
4421       { 13 } \_unravel_scan_pdf_ext_toks:
4422       { 14 } \_unravel_scan_pdf_ext_toks:
4423       { 15 } \_unravel_scan_int:

```

```

4424     { 16 } \__unravel_scan_int:
4425     { 17 } \__unravel_scan_pdfstrcmp:
4426     { 18 } \__unravel_scan_pdfcolorstackinit:
4427     { 19 } \__unravel_scan_pdf_ext_toks:
4428     { 20 } \__unravel_scan_pdf_ext_toks:
4429     { 22 } \__unravel_scan_pdf_ext_toks:
4430     { 23 } \__unravel_scan_pdf_ext_toks:
4431     { 24 }
4432     {
4433         \__unravel_scan_keyword:n { fFillLeE }
4434         \__unravel_scan_pdf_ext_toks:
4435     }
4436     { 25 } \__unravel_scan_pdffiledump:
4437     { 26 } \__unravel_scan_pdfmatch:
4438     { 27 } \__unravel_scan_int:
4439     { 28 } \__unravel_scan_int:
4440     { 30 } \__unravel_scan_int:
4441     { 31 } \__unravel_scan_pdfximagebbox:
4442     }
4443     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4444     \__unravel_back_input_tl_o:
4445     }
4446     \cs_new_protected_nopar:Npn \__unravel_scan_pdfstrcmp:
4447     {
4448         \__unravel_scan_toks_to_str:
4449         \__unravel_scan_toks_to_str:
4450     }
4451     \cs_new_protected_nopar:Npn \__unravel_scan_pdfximagebbox:
4452     { \__unravel_scan_int: \__unravel_scan_int: }
4453     \cs_new_protected_nopar:Npn \__unravel_scan_pdfcolorstackinit:
4454     {
4455         \__unravel_scan_keyword:nTF { pPaAgGeE }
4456         { \bool_set_true:N \l__unravel_tmpa_bool }
4457         { \bool_set_false:N \l__unravel_tmpb_bool }
4458         \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4459         { \__unravel_scan_keyword:n { pPaAgGeE } }
4460         \__unravel_scan_toks_to_str:
4461     }
4462     \cs_new_protected_nopar:Npn \__unravel_scan_pdffiledump:
4463     {
4464         \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
4465         \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
4466         \__unravel_scan_pdf_ext_toks:
4467     }
4468     \cs_new_protected_nopar:Npn \__unravel_scan_pdfmatch:
4469     {
4470         \__unravel_scan_keyword:n { iIcCaAsSeE }
4471         \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
4472         { \__unravel_scan_int: }
4473         \__unravel_scan_pdf_ext_toks:
4474         \__unravel_scan_pdf_ext_toks:
4475     }

```

\\_\_unravel\_get\_the:

```

4476 \_unravel_new_tex_expandable:nn { the } % 111
4477 {
4478   \_unravel_get_the:
4479   \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
4480   \_unravel_back_input:V \l__unravel_tmpa_tl
4481   \_unravel_print_action:
4482 }
4483 \cs_new_protected_nopar:Npn \_unravel_get_the:
4484 {
4485   \_unravel_prev_input_gpush:N \l__unravel_head_tl
4486   \_unravel_print_action:
4487   \int_if_odd:nTF \l__unravel_head_char_int
4488   { % \unexpanded, \detokenize
4489     \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4490     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4491     \_unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4492   }
4493   { % \the
4494     \_unravel_get_x_next:
4495     \_unravel_scan_something_internal:n { 5 }
4496     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4497     \_unravel_set_action_text:x
4498     {
4499       \tl_head:N \l__unravel_head_tl
4500       => \tl_tail:N \l__unravel_head_tl
4501     }
4502     \tl_set:Nx \l__unravel_head_tl
4503     { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4504   }
4505 }

```

(End definition for \\_unravel\_get\_the:.)

```

4506 \_unravel_new_tex_expandable:nn { top_bot_mark } % 112
4507 { \_unravel_back_input_tl_o: }

4508 \_unravel_new_tex_expandable:nn { end_template } % 117
4509 {
4510   \_unravel_not_implemented:n { end-template } { } { } { }
4511   \_unravel_back_input_tl_o:
4512 }

```

### 2.14.1 Conditionals

```

\_unravel_pass_text:
\_unravel_pass_text_done:w
4513 \cs_new_protected_nopar:Npn \_unravel_pass_text:
4514 {
4515   \_unravel_input_if_empty:TF
4516   { \_unravel_pass_text_empty: }
4517   {
4518     \_unravel_input_get:N \l__unravel_tmpb_gtl
4519     \if_true:
4520       \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one
4521       \exp_after:wN \_unravel_pass_text_done:w
4522     \fi:

```

```

4523         \_unravel_input_gpop:N \l\_unravel_tmpb_gtl
4524         \exp_after:wN \_unravel_pass_text:
4525     \else:
4526         \use:c { fi: }
4527         \int_set_eq:NN \l\_unravel_if_nesting_int \c_one
4528         \_unravel_input_gpop:N \l\_unravel_tmpb_gtl
4529         \exp_after:wN \_unravel_pass_text_nested:
4530     \fi:
4531 }
4532 }
4533 \cs_new_protected_nopar:Npn \_unravel_pass_text_done:w
4534 {
4535     \_unravel_get_next:
4536     \token_if_eq_meaning:NNT \l\_unravel_head_token \fi: { \if_true: }
4537     \else:
4538 }

```

(End definition for \\_unravel\_pass\_text: and \\_unravel\_pass\_text\_done:w.)

\\_unravel\_pass\_text\_nested: Again, if there is no more input we are in trouble. The construction otherwise essentially results in

```

\if_true: \if_true: \else: <head>
\int_decr:N \l\_unravel_if_nesting_int \use_none:nnnnn \fi:
\use_none:nnn \fi:
\int_incr:N \l\_unravel_if_nesting_int \fi:

```

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

```

4539 \cs_new_protected_nopar:Npn \_unravel_pass_text_nested:
4540 {
4541     \_unravel_input_if_empty:TF
4542     { \_unravel_pass_text_empty: }
4543     {
4544         \_unravel_input_get:N \l\_unravel_tmpb_gtl
4545         \if_true:
4546             \if_true:
4547                 \gtl_head_do:NN \l\_unravel_tmpb_gtl \else:
4548                 \int_decr:N \l\_unravel_if_nesting_int
4549                 \use_none:nnnnn
4550             \fi:
4551             \use_none:nnn
4552         \fi:
4553         \int_incr:N \l\_unravel_if_nesting_int
4554         \fi:
4555         \_unravel_input_gpop:N \l\_unravel_unused_gtl
4556         \int_compare:nNnTF \l\_unravel_if_nesting_int = \c_zero
4557         { \_unravel_pass_text: }
4558         { \_unravel_pass_text_nested: }
4559     }
4560 }

```

(End definition for \\_unravel\_pass\_text\_nested:.)

\\_unravel\_pass\_text\_empty:

```

4561 \cs_new_protected_nopar:Npn \_unravel_pass_text_empty:
4562 {
4563   \_unravel_error:nnnnn { runaway-if } { } { } { } { }
4564   \_unravel_exit:w
4565 }

```

(End definition for \\_unravel\_pass\_text\_empty:.)

\\_unravel\_cond\_push:

\\_unravel\_cond\_pop:

```

4566 \cs_new_protected:Npn \_unravel_cond_push:
4567 {
4568   \tl_gput_left:Nx \g__unravel_if_limit_tl
4569   { { \int_use:N \g__unravel_if_limit_int } }
4570   \int_gincr:N \g__unravel_if_depth_int
4571   \int_gzero:N \g__unravel_if_limit_int
4572 }
4573 \cs_new_protected_nopar:Npn \_unravel_cond_pop:
4574 {
4575   \int_gset:Nn \g__unravel_if_limit_int
4576   { \tl_head:N \g__unravel_if_limit_tl }
4577   \tl_gset:Nx \g__unravel_if_limit_tl
4578   { \tl_tail:N \g__unravel_if_limit_tl }
4579   \int_gdecr:N \g__unravel_if_depth_int
4580 }

```

(End definition for \\_unravel\_cond\_push: and \\_unravel\_cond\_pop:.)

\\_unravel\_change\_if\_limit:nn

```

4581 \cs_new_protected:Npn \_unravel_change_if_limit:nn #1#2
4582 {
4583   \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4584   { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4585   {
4586     \tl_clear:N \l__unravel_tmpa_tl
4587     \prg_replicate:nn { \g__unravel_if_depth_int - #2 - \c_one }
4588     {
4589       \tl_put_right:Nx \l__unravel_tmpa_tl
4590       { { \tl_head:N \g__unravel_if_limit_tl } }
4591       \tl_gset:Nx \g__unravel_if_limit_tl
4592       { \tl_tail:N \g__unravel_if_limit_tl }
4593     }
4594     \tl_gset:Nx \g__unravel_if_limit_tl
4595     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4596   }
4597 }

```

(End definition for \\_unravel\_change\_if\_limit:nn.)

```

4598 \_unravel_new_tex_expandable:nn { if_test } % 107
4599 {
4600   \_unravel_cond_push:
4601   \exp_args:NV \_unravel_cond_aux:n \g__unravel_if_depth_int
4602 }

```

\\_unravel\_cond\_aux:nn

```

4603 \cs_new_protected:Npn \_unravel_cond_aux:n #1
4604 {
4605   \int_case:nnF \l__unravel_head_char_int
4606   {
4607     { 12 } { \_unravel_test_ifx:n {#1} }
4608     { 16 } { \_unravel_test_case:n {#1} }
4609     { 21 } { \_unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4610   }
4611   {
4612     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4613     \_unravel_print_action:
4614     \int_case:nn \l__unravel_head_char_int
4615     {
4616       { 0 } { \_unravel_test_two_chars: } % if
4617       { 1 } { \_unravel_test_two_chars: } % ifcat
4618       { 2 } % ifnum
4619       { \_unravel_test_two_vals:N \_unravel_scan_int: }
4620       { 3 } % ifdim
4621       { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4622       { 4 } { \_unravel_scan_int: } % ifodd
4623       % { 5 } { } % ifvmode
4624       % { 6 } { } % ifhmode
4625       % { 7 } { } % ifmmode
4626       % { 8 } { } % ifinner
4627       { 9 } { \_unravel_scan_int: } % ifvoid
4628       { 10 } { \_unravel_scan_int: } % ifhbox
4629       { 11 } { \_unravel_scan_int: } % ifvbox
4630       { 13 } { \_unravel_scan_int: } % ifeof
4631       % { 14 } { } % iftrue
4632       % { 15 } { } % iffalse
4633       { 17 } { \_unravel_test_ifdefined: } % ifdefined
4634       { 18 } { \_unravel_test_ifcsname: } % ifcsname
4635       { 19 } % iffontchar
4636       { \_unravel_scan_font_ident: \_unravel_scan_int: }
4637       % { 20 } { } % ifincsname % ^^A todo: something?
4638       { 22 } % ifpdfabsnum
4639       { \_unravel_test_two_vals:N \_unravel_scan_int: }
4640       { 23 } % ifpdfabsdim
4641       { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4642     }
4643     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4644     \_unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4645     \l__unravel_head_tl \scan_stop:
4646     \exp_after:wN \_unravel_cond_true:n
4647   \else:
4648     \exp_after:wN \_unravel_cond_false:n
4649   \fi:
4650   {#1}
4651 }
4652 }

```

(End definition for \\_unravel\_cond\_aux:nn.)

\\\_unravel\_cond\_true:n

```

4653 \cs_new_protected:Npn \\_unravel_cond_true:n #1
4654 {
4655   \\_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4656   \\_unravel_print_action:x { \g_unravel_action_text_str = true }
4657 }

```

(End definition for \\\_unravel\_cond\_true:n.)

\\\_unravel\_cond\_false:n

\\\_unravel\_cond\_false\_loop:n

\\\_unravel\_cond\_false\_common:

```

4658 \cs_new_protected:Npn \\_unravel_cond_false:n #1
4659 {
4660   \\_unravel_cond_false_loop:n {#1}
4661   \\_unravel_cond_false_common:
4662   \\_unravel_print_action:x { \g_unravel_action_text_str = false }
4663 }
4664 \cs_new_protected:Npn \\_unravel_cond_false_loop:n #1
4665 {
4666   \\_unravel_pass_text:
4667   \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
4668   {
4669     \token_if_eq_meaning:NNT \l_unravel_head_token \or:
4670     {
4671       \\_unravel_error:nnnnn { extra-or } { } { } { } { }
4672       \\_unravel_cond_false_loop:n {#1}
4673     }
4674   }
4675   {
4676     \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
4677     { \\_unravel_cond_pop: }
4678     \\_unravel_cond_false_loop:n {#1}
4679   }
4680 }
4681 \cs_new_protected_nopar:Npn \\_unravel_cond_false_common:
4682 {
4683   \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
4684   { \\_unravel_cond_pop: }
4685   { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
4686 }

```

(End definition for \\\_unravel\_cond\_false:n, \\\_unravel\_cond\_false\_loop:n, and \\\_unravel\_cond\_false\_common:.)

\\\_unravel\_test\_two\_vals:N

```

4687 \cs_new_protected:Npn \\_unravel_test_two_vals:N #1
4688 {
4689   #1
4690   \\_unravel_get_x_non_blank:
4691   \tl_if_in:nVF { < = > } \l_unravel_head_tl
4692   {
4693     \\_unravel_error:nnnnn { missing-equals } { } { } { } { }
4694     \\_unravel_back_input:
4695     \tl_set:Nn \l_unravel_head_tl { = }
4696   }

```

```

4697     \__unravel_prev_input:V \l__unravel_head_tl
4698     #1
4699   }

```

(End definition for \\_\_unravel\_test\_two\_vals:N.)

\\_\_unravel\_test\_two\_chars:

```

\__unravel_test_two_chars_aux:
4700 \cs_new_protected_nopar:Npn \__unravel_test_two_chars:
4701 {
4702   \__unravel_test_two_chars_aux:
4703   \__unravel_prev_input:V \l__unravel_head_tl
4704   \__unravel_test_two_chars_aux:
4705   \__unravel_prev_input:V \l__unravel_head_tl
4706 }
4707 \cs_new_protected_nopar:Npn \__unravel_test_two_chars_aux:
4708 {
4709   \__unravel_get_x_next:
4710   \gtl_if_tl:NF \l__unravel_head_gtl
4711   {
4712     \tl_set:Nx \l__unravel_head_tl
4713     {
4714       \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
4715       { \c_group_begin_token } { \c_group_end_token }
4716     }
4717   }
4718   \tl_put_left:Nn \l__unravel_head_tl { \exp_not:N } % ^^A todo: prettify.
4719 }

```

(End definition for \\_\_unravel\_test\_two\_chars: and \\_\_unravel\_test\_two\_chars\_aux:.)

\\_\_unravel\_test\_ifx:n

\\_\_unravel\_test\_ifx\_aux:w

```

4720 \cs_new_protected:Npn \__unravel_test_ifx:n #1
4721 {
4722   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4723   \__unravel_print_action:
4724   \__unravel_get_next:
4725   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4726   \__unravel_get_next:
4727   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4728   \__unravel_set_action_text:x
4729   {
4730     Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4731     \gtl_to_str:N \l__unravel_tmpb_gtl
4732     \gtl_to_str:N \l__unravel_head_gtl
4733   }
4734   \gtl_head_do:NN \l__unravel_tmpb_gtl \__unravel_test_ifx_aux:w
4735   \exp_after:wN \__unravel_cond_true:n
4736   \else:
4737   \exp_after:wN \__unravel_cond_false:n
4738   \fi:
4739   {#1}
4740 }
4741 \cs_new_nopar:Npn \__unravel_test_ifx_aux:w
4742 { \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
4743 \cs_new_protected:Npn \_unravel_test_case:n #1
4744 {
4745   \_unravel_prev_input_gpush:N \l__unravel_head_tl
4746   \_unravel_print_action:
4747   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
4748   \_unravel_scan_int:
4749   \_unravel_prev_input_get:N \l__unravel_head_tl
4750   \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
4751   % ^^A does text_case_aux use prev_input_seq?
4752   \exp_args:No \_unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
4753   \_unravel_prev_input_gpop:N \l__unravel_head_tl
4754   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4755 }
4756 \cs_new_protected:Npn \_unravel_test_case_aux:nn #1#2
4757 {
4758   \int_compare:nNnTF {#1} = \c_zero
4759   { \_unravel_change_if_limit:nn { 4 } {#2} }
4760   {
4761     \_unravel_pass_text:
4762     \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
4763     {
4764       \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
4765       {
4766         \exp_args:Nf \_unravel_test_case_aux:nn
4767         { \int_eval:n { #1 - 1 } } {#2}
4768       }
4769       { \_unravel_cond_false_common: }
4770     }
4771     {
4772       \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4773       { \_unravel_cond_pop: }
4774       \_unravel_test_case_aux:nn {#1} {#2}
4775     }
4776   }
4777 }

```

(End definition for \\_unravel\_test\_case:n and \\_unravel\_test\_case\_aux:nn.)

```

\_unravel_test_ifdefined:
4778 \cs_new_protected_nopar:Npn \_unravel_test_ifdefined:
4779 {
4780   \_unravel_input_if_empty:TF
4781   { \_unravel_pass_text_empty: }
4782   {
4783     \_unravel_input_gpop:N \l__unravel_tmpb_gtl
4784     \_unravel_set_action_text:x
4785     {
4786       Conditional:~ \tl_to_str:N \l__unravel_head_tl
4787       \gtl_to_str:N \l__unravel_tmpb_gtl
4788     }
4789     \_unravel_prev_input:x

```

```

4790     {
4791         \gtl_if_tl:NTF \l__unravel_tmpb_gtl
4792         { \gtl_head:N \l__unravel_tmpb_gtl }
4793         { \gtl_to_str:N \l__unravel_tmpb_gtl }
4794     }
4795 }
4796 }

```

(End definition for \\_\_unravel\_test\_ifdefined:.)

\\_\_unravel\_test\_ifcsname:

```

4797 \cs_new_protected_nopar:Npn \__unravel_test_ifcsname:
4798 {
4799     \__unravel_csname_loop:
4800     \__unravel_prev_input:V \l__unravel_head_tl
4801 }

```

(End definition for \\_\_unravel\_test\_ifcsname:.)

```

4802 \__unravel_new_tex_expandable:nm { fi_or_else } % 108
4803 {
4804     \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
4805     {
4806         \int_compare:nNnTF \g__unravel_if_limit_int = \c_zero
4807         {
4808             \int_compare:nNnTF \g__unravel_if_depth_int = \c_zero
4809             { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
4810             { \__unravel_insert_relax: }
4811         }
4812         { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
4813     }
4814     {
4815         \__unravel_set_action_text:
4816         \int_compare:nNnF \l__unravel_head_char_int = \c_two
4817         {
4818             \__unravel_if_or_else_loop:
4819             \__unravel_set_action_text:x
4820             {
4821                 \g__unravel_action_text_str \c_space_tl
4822                 => ~ skipped ~ to ~ \tl_to_str:N \l__unravel_head_tl
4823             }
4824         }
4825         % ^^A todo: in this print_action the token itself is missing.
4826         \__unravel_print_action:
4827         \__unravel_cond_pop:
4828     }
4829 }
4830 \cs_new_protected_nopar:Npn \__unravel_if_or_else_loop:
4831 {
4832     \int_compare:nNnF \l__unravel_head_char_int = \c_two
4833     {
4834         \__unravel_pass_text:
4835         \__unravel_set_cmd:
4836         \__unravel_if_or_else_loop:
4837     }
4838 }

```

## 2.15 User interaction

### 2.15.1 Print

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

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

`\l__unravel_print_tl`

```
4839 \tl_new:N \l__unravel_print_tl
4840 \sys_if_engine luatex:TF
4841 {
4842   \cs_new_protected:Npx \__unravel_print_normalize_null:
4843   {
4844     \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
4845     { \char_generate:nn { 0 } { 12 } }
4846     { \tl_to_str:n { ^^@ } }
4847   }
4848 }
4849 { \cs_new_protected:Npn \__unravel_print_normalize_null: { } }
```

*(End definition for \\_\_unravel\_print\_normalize\_null: and \l\_\_unravel\_print\_tl.)*

`\__unravel_print:n`

`\__unravel_print:x`

```
4850 \cs_new_protected:Npn \__unravel_print:n #1
4851 {
4852   \tl_set:Nn \l__unravel_print_tl {#1}
4853   \__unravel_print_normalize_null:
4854   \__unravel_exp_args:Nx \iow_term:n { \l__unravel_print_tl }
4855 }
4856 \cs_new_protected:Npn \__unravel_print:x
4857 { \__unravel_exp_args:Nx \__unravel_print:n }
```

*(End definition for \\_\_unravel\_print:n.)*

`\__unravel_print_message:nn`

The message to be printed should come already detokenized, as #2. It will be wrapped to 80 characters per line, with #1 before each line.

```
4858 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
4859 { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_print:n }
```

*(End definition for \\_\_unravel\_print\_message:nn.)*

`\__unravel_set_action_text:x`

```
4860 \cs_new_protected:Npn \__unravel_set_action_text:x #1
4861 {
4862   \group_begin:
4863     \__unravel_set_escapechar:n { 92 }
4864     \str_gset:Nx \g__unravel_action_text_str {#1}
4865   \group_end:
4866 }
```

*(End definition for \\_\_unravel\_set\_action\_text:x.)*

`\__unravel_set_action_text:`

```
4867 \cs_new_protected_nopar:Npn \__unravel_set_action_text:
4868 {
4869   \__unravel_set_action_text:x
4870   {
4871     \tl_to_str:N \l__unravel_head_tl
4872     \tl_if_single_token:VT \l__unravel_head_tl
4873     { = ~ \exp_after:wN \token_to_meaning:N \l__unravel_head_tl }
4874   }
4875 }
```

*(End definition for \\_\_unravel\_set\_action\_text:.)*

`\__unravel_print_state:`

```
4876 \cs_new_protected:Npn \__unravel_print_state:
4877 {
4878   \group_begin:
4879   \__unravel_set_escapechar:n { 92 }
4880   \tl_use:N \g__unravel_before_print_state_tl
4881   \int_compare:nNnT \g__unravel_noise_int > \c_zero
4882   {
4883     \__unravel_print_state_output:
4884     \__unravel_print_state_prev:
4885     \__unravel_print_state_input:
4886   }
4887   \group_end:
4888   \__unravel_prompt:
4889 }
```

*(End definition for \\_\_unravel\_print\_state:.)*

`\__unravel_print_state_output:`

`\__unravel_print_state_output:n`

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

```
4890 \cs_new_protected_nopar:Npn \__unravel_print_state_output:
4891 {
4892   \__unravel_exp_args:Nx \__unravel_print_state_output:n
4893   { \gtl_to_str:N \g__unravel_output_gtl }
4894 }
4895 \cs_new_protected:Npn \__unravel_print_state_output:n #1
4896 {
4897   \tl_if_empty:nF {#1}
4898   {
4899     \__unravel_print_message:nn { <| ~ }
4900     { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
4901   }
4902 }
```

*(End definition for \\_\_unravel\_print\_state\_output: and \\_\_unravel\_print\_state\_output:n.)*

`\__unravel_print_state_prev:`

Never trim ##1.

```
4903 \cs_new_protected_nopar:Npn \__unravel_print_state_prev:
4904 {
4905   \seq_set_map:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
```

```

4906     { \__unravel_to_str:n {##1} }
4907     \seq_remove_all:Nn \l__unravel_tmpa_seq { }
4908     \seq_if_empty:NTF \l__unravel_tmpa_seq
4909     { \__unravel_print_message:nn { || ~ } { } }
4910     {
4911       \seq_map_inline:Nn \l__unravel_tmpa_seq
4912       {
4913         \__unravel_print_message:nn { || ~ } {##1}
4914       }
4915     }
4916   }

```

(End definition for \\_\_unravel\_print\_state\_prev:.)

\\_\_unravel\_print\_state\_input: Print #1 with each line starting with |>~. The \\_\_unravel\_str\_truncate\_right:nn function trims #1 if needed, to fit in a maximum of \g\_\_unravel\_max\_input\_int characters.

```

4917 \cs_new_protected_nopar:Npn \__unravel_print_state_input:
4918 {
4919   \__unravel_exp_args:Nx \__unravel_print_state_input:n
4920   { \__unravel_input_to_str: }
4921 }
4922 \cs_new_protected:Npn \__unravel_print_state_input:n #1
4923 {
4924   \__unravel_print_message:nn { |> ~ }
4925   { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
4926 }

```

(End definition for \\_\_unravel\_print\_state\_input: and \\_\_unravel\_print\_state\_input:n.)

\\_\_unravel\_print\_meaning:

```

4927 \cs_new_protected:Npn \__unravel_print_meaning:
4928 {
4929   \__unravel_input_if_empty:TF
4930   { \__unravel_print_message:nn { } { Empty-input! } }
4931   {
4932     \__unravel_input_get:N \l__unravel_tmpb_gtl
4933     \__unravel_print_message:nn { }
4934     {
4935       \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
4936       = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
4937     }
4938   }
4939 }

```

(End definition for \\_\_unravel\_print\_meaning:.)

\\_\_unravel\_print\_action:

```

\__unravel_print_action:x
4940 \cs_new_protected:Npn \__unravel_print_action:
4941 {
4942   \int_gincr:N \g__unravel_step_int
4943   \__unravel_print:x
4944   {
4945     [====
4946     \bool_if:NT \g__unravel_number_steps_bool

```

```

4947         { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
4948     =====]~
4949     \int_compare:nNnTF
4950     { \str_count:N \g__unravel_action_text_str }
4951     > { \g__unravel_max_action_int }
4952     {
4953         \str_range:Nnn \g__unravel_action_text_str
4954         { 1 } { \g__unravel_max_action_int - 3 } ...
4955     }
4956     { \g__unravel_action_text_str }
4957 }
4958 \__unravel_print_state:
4959 }
4960 \cs_new_protected:Npn \__unravel_print_action:x #1
4961 {
4962     \__unravel_set_action_text:x {#1}
4963     \__unravel_print_action:
4964 }

```

(End definition for \\_\_unravel\_print\_action: and \\_\_unravel\_print\_action:x.)

\\_\_unravel\_print\_gtl\_action:N

```

4965 \cs_new_protected:Npn \__unravel_print_gtl_action:N #1
4966 {
4967     \__unravel_print_action:x { \gtl_to_str:N #1 }
4968 }

```

(End definition for \\_\_unravel\_print\_gtl\_action:N.)

\\_\_unravel\_print\_done:x

```

4969 \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:

```

4970 \cs_new_protected_nopar:Npn \__unravel_print_assigned_token:
4971 {
4972     \__unravel_after_assignment: % ^^A todo: simplify
4973     \__unravel_print_action:x
4974     {
4975         Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4976         = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
4977     }
4978     \__unravel_omit_after_assignment:w
4979 }
4980 \cs_new_protected_nopar:Npn \__unravel_print_assigned_register:
4981 {
4982     \__unravel_after_assignment: % ^^A todo: simplify
4983     \__unravel_exp_args:Nx \__unravel_print_action:x
4984     {
4985         \exp_not:n
4986         {
4987             Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4988             \tl_if_single:NT \l__unravel_defined_tl
4989             { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }

```

```

4990     }
4991     = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
4992   }
4993   \__unravel_omit_after_assignment:w
4994 }

```

(End definition for \\_\_unravel\_print\_assigned\_token: and \\_\_unravel\_print\_assigned\_register:.)

\\_\_unravel\_print\_welcome: Welcome message.

```

4995 \cs_new_protected_nopar:Npn \__unravel_print_welcome:
4996 {
4997   \__unravel_print_message:nn { }
4998   {
4999     \bool_if:NTF \g__unravel_welcome_message_bool
5000     {
5001       \\
5002       =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5003       \iow_indent:n
5004       {
5005         "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\
5006         "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\
5007         "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
5008         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
5009       }
5010     }
5011     { [=====Start~=====] }
5012   }
5013   \__unravel_print_state:
5014 }

```

(End definition for \\_\_unravel\_print\_welcome:.)

\\_\_unravel\_print\_outcome: Final message.

```

5015 \cs_new_protected_nopar:Npn \__unravel_print_outcome:
5016 { \__unravel_print:n { [=====End~=====] } }

```

(End definition for \\_\_unravel\_print\_outcome:.)

## 2.15.2 Prompt

\\_\_unravel\_prompt:

```

5017 \cs_new_protected_nopar:Npn \__unravel_prompt:
5018 {
5019   \int_gdecr:N \g__unravel_nonstop_int
5020   \int_compare:nNnF \g__unravel_nonstop_int > \c_zero
5021   {
5022     \group_begin:
5023     \__unravel_set_escapechar:n { -1 }
5024     \int_set_eq:NN \tex_endlinechar:D \c_minus_one
5025     \tl_use:N \g__unravel_before_prompt_tl
5026     \__unravel_prompt_aux:
5027     \group_end:
5028   }
5029 }
5030 \cs_new_protected_nopar:Npn \__unravel_prompt_aux:

```

```

5031 {
5032   \int_compare:nNnT { \etex_interactionmode:D } = { 3 }
5033   {
5034     \bool_if:NTF \g__unravel_explicit_prompt_bool
5035     { \ior_str_get:Nc \c__unravel_prompt_ior }
5036     { \ior_str_get:Nc \c__unravel_noprompt_ior }
5037     { Your~input }
5038     \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5039   }
5040 }
5041 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5042 {
5043   \tl_if_empty:nF {#1}
5044   {
5045     \__unravel_exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
5046     {
5047       { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
5048       { q }
5049       {
5050         \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
5051         \int_gzero:N \g__unravel_nonstop_int
5052       }
5053       { x }
5054       {
5055         \group_end:
5056         \exp_after:wN \__unravel_exit:w \__unravel_exit:w
5057       }
5058       { X } { \tex_batchmode:D \tex_end:D }
5059       { s } { \__unravel_prompt_scan_int:nn {#1}
5060         \__unravel_prompt_silent_steps:n }
5061       { o } { \__unravel_prompt_scan_int:nn {#1}
5062         { \int_gset:Nn \g__unravel_noise_int } }
5063       { C }
5064       {
5065         \__unravel_exp_args:Nx \use:n
5066         {
5067           \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5068           { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5069         }
5070         \tl_gput_left:Nn \g__unravel_tmpc_tl
5071         { \tl_gclear:N \g__unravel_tmpc_tl }
5072         \group_insert_after:N \g__unravel_tmpc_tl
5073       }
5074       { | } { \__unravel_prompt_scan_int:nn {#1}
5075         \__unravel_prompt_vert:n }
5076       { a } { \__unravel_prompt_all: }
5077     }
5078     { \__unravel_prompt_help: }
5079   }
5080 }
5081 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
5082 {
5083   \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
5084   \l__unravel_prompt_tmpa_int =

```



```

5085     \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5086     \use_ii:nn #1 \scan_stop:
5087   }
5088   \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5089   {
5090     #2 \l__unravel_prompt_tmpa_int
5091     \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
5092   }
5093   \cs_new_protected:Npn \__unravel_prompt_help:
5094   {
5095     \__unravel_print:n { "m":~meaning-of~first-token }
5096     \__unravel_print:n { "q":~semi-quiet~(same-as~"oi") }
5097     \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
5098     \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
5099     \__unravel_print:n
5100     { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
5101     \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
5102     \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
5103     \__unravel_print:n { "a":~print~state~again,~without~truncating }
5104     \__unravel_prompt_aux:
5105   }
5106   \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
5107   {
5108     \int_compare:nNnF {#1} < \c_zero
5109     {
5110       \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
5111       \tl_gset:Nn \g__unravel_before_prompt_tl
5112       {
5113         \int_gset_eq:NN \g__unravel_noise_int \c_one
5114         \tl_gclear:N \g__unravel_before_prompt_tl
5115       }
5116       \int_gset:Nn \g__unravel_nonstop_int {#1}
5117     }
5118   }
5119   \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5120   {
5121     \int_compare:nNnTF {#1} < { 0 }
5122     { \__unravel_prompt_vert:Nn > {#1} }
5123     { \__unravel_prompt_vert:Nn < {#1} }
5124   }
5125   \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5126   {
5127     \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
5128     \tl_gset:Nf \g__unravel_before_print_state_tl
5129     {
5130       \exp_args:NNf \exp_stop_f: \int_compare:nNnTF
5131       { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5132       #1 { \__unravel_prev_input_count: }
5133       {
5134         \int_gset:Nn \g__unravel_nonstop_int
5135         { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5136       }
5137     }
5138     \int_gset_eq:NN \g__unravel_noise_int \c_one

```

```

5139         \tl_gclear:N \g__unravel_before_print_state_tl
5140     }
5141 }
5142 }
5143 \cs_new_protected_nopar:Npn \__unravel_prompt_all:
5144 {
5145     \tl_gset:Nx \g__unravel_tmpc_tl
5146     {
5147         \exp_not:n
5148         {
5149             \tl_gclear:N \g__unravel_tmpc_tl
5150             \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5151             \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5152             \__unravel_print_state:
5153         }
5154         \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5155         \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5156     }
5157     \group_insert_after:N \g__unravel_tmpc_tl
5158 }
5159 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5160 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for \\_\_unravel\_prompt:.)

### 2.15.3 Errors

\\_\_unravel\_not\_implemented:n

```

5161 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5162 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End definition for \\_\_unravel\_not\_implemented:n.)

\\_\_unravel\_error:nnnnn  
\\_\_unravel\_error:nxxxx

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

```

5163 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5164 {
5165     \group_begin:
5166     \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5167     \group_end:
5168 }
5169 \cs_new_protected:Npn \__unravel_error:nxxxx #1#2#3#4#5
5170 {
5171     \group_begin:
5172     \msg_error:nnxxxx { unravel } {#1} {#2} {#3} {#4} {#5}
5173     \group_end:
5174 }

```

(End definition for \\_\_unravel\_error:nnnnn.)

\\_\_unravel\_tex\_msg\_new:nnn

This stores a T<sub>E</sub>X error message.

```

5175 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5176 {
5177     \cs_new_nopar:cpn { __unravel_tex_msg_error_#1: } {#2}
5178     \cs_new_nopar:cpn { __unravel_tex_msg_help_#1: } {#3}
5179 }

```

(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.

```

5180 \cs_new_protected:Npn \_unravel\_tex\_error:nn #1#2
5181 {
5182   \group_begin:
5183   \msg_error:nnxxx { unravel } { tex-error }
5184   { \tl_to_str:n {#2} }
5185   { \use:c { \_unravel\_tex\_msg\_error\_#1: } }
5186   { \use:c { \_unravel\_tex\_msg\_help\_#1: } }
5187   \group_end:
5188 }
5189 \cs_generate_variant:Nn \_unravel\_tex\_error:nn { nV }

```

(End definition for `\_unravel\_tex\_error:nn`.)

`\_unravel\_tex\_fatal\_error:nn` Throw the `tex-fatal` error message, with arguments: `#2` which triggered the fatal error, `TEX`'s error message, and `TEX`'s help text.

```

5190 \cs_new_protected:Npn \_unravel\_tex\_fatal\_error:nn #1#2
5191 {
5192   \_unravel\_error:nnxxx { tex-fatal }
5193   { \tl_to_str:n {#2} }
5194   { \use:c { \_unravel\_tex\_msg\_error\_#1: } }
5195   { \use:c { \_unravel\_tex\_msg\_help\_#1: } }
5196   { }
5197 }
5198 \cs_generate_variant:Nn \_unravel\_tex\_fatal\_error:nn { nV }

```

(End definition for `\_unravel\_tex\_fatal\_error:nn`.)

## 2.16 Keys

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

```

5199 \keys_define:nn { unravel/defaults }
5200 {
5201   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5202   internal-debug .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5203   max-action .int_gset:N = \g__unravel_default_max_action_int ,
5204   max-output .int_gset:N = \g__unravel_default_max_output_int ,
5205   max-input .int_gset:N = \g__unravel_default_max_input_int ,
5206   number-steps .bool_gset:N = \g__unravel_default_number_steps_bool ,
5207   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5208 }
5209 \keys_define:nn { unravel }
5210 {
5211   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5212   internal-debug .bool_gset:N = \g__unravel_internal_debug_bool ,
5213   max-action .int_gset:N = \g__unravel_max_action_int ,
5214   max-output .int_gset:N = \g__unravel_max_output_int ,
5215   max-input .int_gset:N = \g__unravel_max_input_int ,

```

```

5216     number-steps      .bool_gset:N = \g__unravel_number_steps_bool ,
5217     welcome-message   .bool_gset:N = \g__unravel_welcome_message_bool ,
5218 }

```

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.

```

5219 \tl_map_inline:nn { { /defaults } { } }
5220 {
5221   \keys_define:nn { unravel #1 }
5222   {
5223     machine      .meta:nn =
5224     { unravel #1 }
5225     {
5226       explicit-prompt = false ,
5227       internal-debug  = false ,
5228       max-action      = \c_max_int ,
5229       max-output      = \c_max_int ,
5230       max-input       = \c_max_int ,
5231       number-steps    = false ,
5232       welcome-message = false ,
5233     } ,
5234   }
5235 }

```

## 2.17 Main command

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

```

5236 \NewDocumentCommand \unravel { 0 { } m } { \unravel:nn {#1} {#2} }

```

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

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

```

5237 \NewDocumentCommand \unravelsetup { m } { \unravel_setup:n {#1} }

```

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

**\unravel\_setup:n** Set keys, updating both default values and current values.

```

5238 \cs_new_protected:Npn \unravel_setup:n #1
5239 {
5240   \keys_set:nn { unravel/defaults } {#1}
5241   \keys_set:nn { unravel } {#1}
5242 }

```

(End definition for `\unravel_setup:n`. This function is documented on page 2.)

**\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.

```

5243 \cs_new_protected:Npn \unravel:nn #1#2
5244 {

```

```

5245     \__unravel_init_key_vars:
5246     \keys_set:nn { unravel } {#1}
5247     \__unravel_init_vars:
5248     \__unravel_input_gset:n {#2}
5249     \__unravel_print_welcome:
5250     \__unravel_main_loop:
5251     \__unravel_exit_point:
5252     \__unravel_print_outcome:
5253     \__unravel_final_test:
5254     \__unravel_exit_point:
5255 }

```

(End definition for \unravel:nn.)

\\_\_unravel\_init\_key\_vars: Give variables that are affected by keys their default values (also controlled by keys).

```

5256 \cs_new_protected_nopar:Npn \__unravel_init_key_vars:
5257 {
5258     \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bool
5259     \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5260     \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5261     \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bool
5262     \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5263     \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5264     \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
5265 }

```

(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.

```

5266 \cs_new_protected_nopar:Npn \__unravel_init_vars:
5267 {
5268     \seq_gclear:N \g__unravel_prev_input_seq
5269     \gtl_gclear:N \g__unravel_output_gtl
5270     \int_gzero:N \g__unravel_step_int
5271     \tl_gclear:N \g__unravel_if_limit_tl
5272     \int_gzero:N \g__unravel_if_limit_int
5273     \int_gzero:N \g__unravel_if_depth_int
5274     \gtl_gclear:N \g__unravel_after_assignment_gtl
5275     \bool_gset_true:N \g__unravel_set_box_allowed_bool
5276     \bool_gset_false:N \g__unravel_name_in_progress_bool
5277     \gtl_clear:N \l__unravel_after_group_gtl
5278 }

```

(End definition for \\_\_unravel\_init\_vars:.)

\\_\_unravel\_main\_loop: Loop forever, getting the next token (with expansion) and performing the corresponding command.

```

5279 \cs_new_protected_nopar:Npn \__unravel_main_loop:
5280 {
5281     \__unravel_get_x_next:
5282     \__unravel_set_cmd:
5283     \__unravel_do_step:
5284     \__unravel_main_loop:
5285 }

```

(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: 5286 \cs_new_protected_nopar:Npn \_unravel_final_test:
                    5287 {
                    5288   \bool_if:nTF
                    5289   {
                    5290     \tl_if_empty_p:N \g__unravel_if_limit_tl
                    5291     && \int_compare_p:nNn \g__unravel_if_limit_int = \c_zero
                    5292     && \int_compare_p:nNn \g__unravel_if_depth_int = \c_zero
                    5293     && \seq_if_empty_p:N \g__unravel_prev_input_seq
                    5294   }
                    5295   { \_unravel_input_if_empty:TF { } { \_unravel_final_bad: } }
                    5296   { \_unravel_final_bad: }
                    5297 }
                    5298 \cs_new_protected_nopar:Npn \_unravel_final_bad:
                    5299 {
                    5300   \_unravel_error:nnnnn { internal }
                    5301   { the-last-unravel-finished-badly } { } { } { }
                    5302 }
```

(End definition for \\_unravel\_final\_test: and \\_unravel\_final\_bad:.)

## 2.18 Messages

```
5303 \msg_new:nnn { unravel } { unknown-primitive }
5304 { Internal~error:~the-primitive-~'#1'~is-not-known. }
5305 \msg_new:nnn { unravel } { extra-fi-or-else }
5306 { Extra~fi,~or,~or-else. }
5307 \msg_new:nnn { unravel } { missing-dollar }
5308 { Missing~dollar~inserted. }
5309 \msg_new:nnn { unravel } { unknown-expandable }
5310 { Internal~error:~the-expandable-command-~'#1'~is-not-known. }
5311 \msg_new:nnn { unravel } { missing-font-id }
5312 { Missing~font~identifier.~\iow_char:N\~nullfont~inserted. }
5313 \msg_new:nnn { unravel } { missing-rparen }
5314 { Missing~right~parenthesis~inserted~for~expression. }
5315 \msg_new:nnn { unravel } { missing-cs }
5316 { Missing~control~sequence.~\iow_char:N\~inaccessible~inserted. }
5317 \msg_new:nnn { unravel } { missing-box }
5318 { Missing~box~inserted. }
5319 \msg_new:nnn { unravel } { missing-to }
5320 { Missing~keyword~'to'~inserted. }
5321 \msg_new:nnn { unravel } { improper-leaders }
5322 { Leaders~not~followed~by~proper~glue. }
5323 \msg_new:nnn { unravel } { extra-close }
5324 { Extra~right~brace~or~\iow_char:N\~endgroup. }
5325 \msg_new:nnn { unravel } { off-save }
5326 { Something~is~wrong~with~groups. }
5327 \msg_new:nnn { unravel } { hrule-bad-mode }
5328 { \iow_char\~hrule-used-in-wrong-mode. }
5329 \msg_new:nnn { unravel } { invalid-mode }
5330 { Invalid~mode~for~this~command. }
5331 \msg_new:nnn { unravel } { color-stack-action-missing }
```

```

5332 { Missing~color~stack~action. }
5333 \msg_new:nnn { unravel } { action-type-missing }
5334 { Missing~action~type. }
5335 \msg_new:nnn { unravel } { identifier-type-missing }
5336 { Missing~identifier~type. }
5337 \msg_new:nnn { unravel } { destination-type-missing }
5338 { Missing~destination~type. }
5339 \msg_new:nnn { unravel } { erroneous-prefixes }
5340 { Prefixes~applied~to~non~assignment~command. }
5341 \msg_new:nnn { unravel } { improper-setbox }
5342 { \iow_char:N\setbox~while~fetching~base~of~an~accent. }
5343 \msg_new:nnn { unravel } { after-advance }
5344 {
5345   Missing~register~after~\iow_char:N\advance,~
5346   \iow_char:N\multiply,~or~\iow_char:N\divide.
5347 }
5348 \msg_new:nnn { unravel } { bad-unless }
5349 { \iow_char:N\unless~not~followed~by~conditional. }
5350 \msg_new:nnn { unravel } { runaway-if }
5351 { Runaway~\iow_char:N\if... }
5352 \msg_new:nnn { unravel } { runaway-macro-parameter }
5353 {
5354   Runaway~macro~parameter~\# #2~after \\\
5355   \iow_indent:n {#1}
5356 }
5357 \msg_new:nnn { unravel } { extra-or }
5358 { Extra~\iow_char:N\or. }
5359 \msg_new:nnn { unravel } { missing-equals }
5360 { Missing~equals~for~\iow_char:N\ifnum~or~\iow_char:N\ifdim. }
5361 \msg_new:nnn { unravel } { internal }
5362 { Internal~error:~'~#1'~\ Please~report. }
5363 \msg_new:nnn { unravel } { not-implemented }
5364 { The~following~feature~is~not~implemented:~'~#1'. }
5365 \msg_new:nnn { unravel } { endinput-ignored }
5366 { The~primitive~\iow_char:N\endinput~was~ignored. }
5367 \msg_new:nnn { unravel } { missing-something }
5368 { Something~is~missing,~sorry! }
5369 \msg_new:nnnn { unravel } { tex-error }
5370 { TeX~sees~"#1"~and~throws~an~error:\\\\ \iow_indent:n {#2} }
5371 {
5372   \tl_if_empty:nTF {#3}
5373   { TeX~provides~no~further~help~for~this~error. }
5374   { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
5375 }
5376 \msg_new:nnnn { unravel } { tex-fatal }
5377 { TeX~sees~"#1"~and~throws~a~fatal~error:\\\\ \iow_indent:n {#2} }
5378 {
5379   \tl_if_empty:nTF {#3}
5380   { TeX~provides~no~further~help~for~this~error. }
5381   { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
5382 }

```

Some error messages from TeX itself.

```

5383 \__unravel_tex_msg_new:nnn { incompatible-mag }
5384 {

```

```

5385     Incompatible~magnification~
5386     ( \int_to_arabic:n { \__unravel_mag: } );~
5387     the~previous~value~will~be~retained
5388   }
5389   {
5390     I~can~handle~only~one~magnification~ratio~per~job.~So~I've~
5391     reverted~to~the~magnification~you~used~earlier~on~this~run.
5392   }
5393   \__unravel_tex_msg_new:nnn { illegal-mag }
5394   {
5395     Illegal~magnification~has~been~changed~to~1000~
5396     ( \int_to_arabic:n { \__unravel_mag: } )
5397   }
5398   { The~magnification~ratio~must~be~between~1~and~32768. }
5399   \__unravel_tex_msg_new:nnn { missing-number }
5400   { Missing~number,~treated~as~zero }
5401   {
5402     A~number~should~have~been~here;~I~inserted~'0'.~
5403     If~you~can't~figure~out~why~I~needed~to~see~a~number,~
5404     look~up~'weird~error'~in~the~index~to~The~TeXbook.
5405   }
5406   \__unravel_tex_msg_new:nnn { the-cannot }
5407   { You~can't~use~'\tl_to_str:N\l__unravel_head_tl'~after~\iow_char:N\the }
5408   { I'm~forgetting~what~you~said~and~using~zero~instead. }
5409   \__unravel_tex_msg_new:nnn { incompatible-units }
5410   { Incompatible~glue~units }
5411   { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }
5412   \__unravel_tex_msg_new:nnn { missing-mu }
5413   { Illegal~unit~of~measure~(mu~inserted) }
5414   {
5415     The~unit~of~measurement~in~math~glue~must~be~mu.~
5416     To~recover~gracefully~from~this~error,~it's~best~to~
5417     delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5418     two~letters.~(See~Chapter~27~of~The~TeXbook.)
5419   }
5420   \__unravel_tex_msg_new:nnn { missing-pt }
5421   { Illegal~unit~of~measure~(pt~inserted) }
5422   {
5423     Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
5424     cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
5425     I'll~assume~that~you~meant~to~say~pt,~for~printer's~points.~
5426     To~recover~gracefully~from~this~error,~it's~best~to~
5427     delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
5428     two~letters.~(See~Chapter~27~of~The~TeXbook.)
5429   }
5430   \__unravel_tex_msg_new:nnn { missing-lbrace }
5431   { Missing~\iow_char:N\{~inserted }
5432   {
5433     A~left~brace~was~mandatory~here,~so~I've~put~one~in.~
5434     You~might~want~to~delete~and/or~insert~some~corrections~
5435     so~that~I~will~find~a~matching~right~brace~soon.~
5436     (If~you're~confused~by~all~this,~try~typing~'\iow_char:N\}'~now.)
5437   }
5438   \__unravel_tex_msg_new:nnn { extra-endcsname }

```



```

5439 { Extra~\token_to_str:c{endcsname} }
5440 { I'm~ignoring~this,~since~I~wasn't~doing~a~\token_to_str:c{csname}. }
5441 \__unravel_tex_msg_new:nnn { missing-endcsname }
5442 { Missing~\token_to_str:c{endcsname}~inserted }
5443 {
5444   The~control~sequence~marked~<to~be~read~again>~should~
5445   not~appear~between~\token_to_str:c{csname}~and~
5446   \token_to_str:c{endcsname}.
5447 }

Fatal TEX error messages.
5448 \__unravel_tex_msg_new:nnn { cannot-read }
5449 { ***~(cannot~\iow_char:N\\read~from~terminal~in~nonstop~modes) }
5450 { }
5451 \__unravel_tex_msg_new:nnn { file-error }
5452 { ***~(job~aborted,~file~error~in~nonstop~mode) }
5453 { }
5454 \__unravel_tex_msg_new:nnn { interwoven-preambles }
5455 { (interwoven~alignment~preambles~are~not~allowed) }
5456 { }

Restore catcodes to their original values.
5457 \__unravel_setup_restore:
5458 </package>

```