

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

Bruno Le Floch

2013/07/28

## Contents

<b>1</b>	<b>unravel documentation</b>	<b>2</b>
1.1	Differences between <code>unravel</code> and TeX's processing . . . . .	3
1.2	Future perhaps . . . . .	4
<b>2</b>	<b>unravel implementation</b>	<b>4</b>
2.1	Variables . . . . .	4
2.1.1	User interaction . . . . .	4
2.1.2	Working with tokens . . . . .	6
2.1.3	Numbers and conditionals . . . . .	7
2.1.4	Boxes and groups . . . . .	8
2.1.5	Constants . . . . .	8
2.2	Variants and helper functions . . . . .	9
2.3	Numeric codes . . . . .	13
2.4	Get next token . . . . .	28
2.5	Manipulating the input . . . . .	33
2.5.1	Elementary operations . . . . .	33
2.5.2	Insert token for error recovery . . . . .	38
2.5.3	Macro calls . . . . .	38
2.6	Expand next token . . . . .	40
2.7	Basic scanning subroutines . . . . .	41
2.8	Working with boxes . . . . .	61
2.9	Paragraphs . . . . .	65
2.10	Groups . . . . .	67
2.11	Modes . . . . .	69
2.12	One step . . . . .	70
2.13	Commands . . . . .	70
2.13.1	Characters: from 0 to 15 . . . . .	70
2.13.2	Boxes: from 16 to 31 . . . . .	74

---

\*This file has version number 0.0a, last revised 2013/07/28.

2.13.3 From 32 to 47 . . . . .	79
2.13.4 Maths: from 48 to 56 . . . . .	82
2.13.5 From 57 to 70 . . . . .	83
2.13.6 Extensions . . . . .	86
2.13.7 Assignments . . . . .	93
2.14 Expandable primitives . . . . .	103
2.14.1 Conditionals . . . . .	109
2.15 User interaction . . . . .	117
2.15.1 Print . . . . .	117
2.15.2 Prompt . . . . .	122
2.16 Main command . . . . .	124
2.17 Messages . . . . .	125

## 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. The `unravel` package is currently based on the behaviour of pdfT<sub>E</sub>X, but it should work in the X<sub>Y</sub>T<sub>E</sub>X and LuaT<sub>E</sub>X engines 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 1.1, should be reported to Bruno Le Floch <blflatex@gmail.com>.

The only public command is `\unravel`, used as `\unravel {<some tokens>}`. It will show on the terminal the various steps that T<sub>E</sub>X performs, waiting for input from the user at each step. As a result, `unravel` can only be used meaningfully when T<sub>E</sub>X is run in a terminal.

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  $\TeX$ , it is not fast on very large pieces of code. For instance, running it on `\documentclass{article}` takes about ten minutes on my machine, and finishes after slightly more than 20000 steps.

```

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

```

The `\relax` command is needed after `\documentclass{article}` because this command tries to look for an optional argument: `\unravel` would not find any token, and would give up, as  $\TeX$  would if your file ended just after `\documentclass{article}`. After running the above through `pdf $\TeX$` , one can check that the result is identical to that without `unravel`. Note that `\unravel\usepackage{lipsum}\relax`, despite taking as many steps to complete, is four times slower, because `\newcommand` uses delimited arguments, which prevent some optimizations that `unravel` can otherwise obtain. For comparison, `\unravel{\lipsum[1-30]}`, which also takes 20000 step, takes 8 minutes to complete.

## 1.1 Differences between `unravel` and $\TeX$ 's processing

- Alignments (`\halign`, `\valign`, `\noalign`, `\omit`, `\span`, `\cr`, `\crcr`, `&`) are not implemented.

- Math mode is not implemented, but might be.
- Some other primitives are not implemented (but they should claim so).
- For `unravel`, category codes are fixed when a file is read, while `TeX` only fixes category codes when the corresponding characters are converted to tokens. Similarly, the argument of `\scantokens` is converted to the new category code régime in one go, and the result must be balanced.
- Explicit begin-group and end-group characters other than the usual left and right braces will either make `unravel` choke or will be replaced by the usual left and right braces.
- `\endinput` is ignored, as it is not possible to implement it in a way similar to `TeX`'s, and as it is most often used at the very end of files, in a redundant way.
- Negative signs and glue probably mix in wrong ways (this is a fixable bug, please tell me if it affects you).

## 1.2 Future perhaps

- Allow users to change some settings.
- Fix the display for `\if` and `\ifcat` (remove extraneous `\exp_not:N`).

## 2 unravel implementation

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

```

1 <*package>
2 \RequirePackage{expl3}[2013/07/01]
3 \RequirePackage{l3str}[2013/04/24]
4 \RequirePackage{gtl}[2013/07/28]
5 \ProvidesExplPackage
6   {unravel} {2013/07/28} {0.0a} {Watching TeX digest tokens}
7 <@@=unravel>

```

### 2.1 Variables

#### 2.1.1 User interaction

```

\g__unravel_prompt_ior
\g__unravel_prompt_before_tl
\l__unravel_prompt_tmpa_int
8 \ior_new:N \g__unravel_prompt_ior
9 \tl_new:N \g__unravel_prompt_before_tl
10 \int_new:N \l__unravel_prompt_tmpa_int

```

*(End definition for `\g__unravel_prompt_ior`, `\g__unravel_prompt_before_tl`, and `\l__unravel_prompt_tmpa_int`. These variables are documented on page ??.)*

`\g__unravel_nonstop_int` The number of prompts to skip.

`\g__unravel_noise_int`

```

11 \int_new:N \g__unravel_nonstop_int
12 \int_new:N \g__unravel_noise_int
13 \int_gset_eq:NN \g__unravel_noise_int \c_one

```

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

`\l__unravel_debug_bool` If true, debug-mode. Activated by `\UnravelDebug`

```

14 \bool_new:N \l__unravel_debug_bool

```

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

`\g__unravel_step_int` Current expansion step.

```

15 \int_new:N \g__unravel_step_int

```

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

`\g__unravel_action_text_str` Text describing the action, displayed at each step. This should only be altered through `\__unravel_set_action_text:x`, which sets the escape character as appropriate before converting the argument to a string.

```

16 \str_new:N \g__unravel_action_text_str

```

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

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

`\g__unravel_max_output_int`

`\g__unravel_max_prev_int`

`\g__unravel_max_input_int`

```

17 \int_new:N \g__unravel_max_action_int
18 \int_new:N \g__unravel_max_output_int
19 \int_new:N \g__unravel_max_prev_int
20 \int_new:N \g__unravel_max_input_int
21 \int_gset:Nn \g__unravel_max_action_int { 50 }
22 \int_gset:Nn \g__unravel_max_output_int { 300 }
23 \int_gset:Nn \g__unravel_max_prev_int { 300 }
24 \int_gset:Nn \g__unravel_max_input_int { 300 }

```

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

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

```

25 \bool_new:N \g__unravel_speedup_macros_bool
26 \bool_gset_true:N \g__unravel_speedup_macros_bool

```

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

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

```

27 \int_new:N \l__unravel_print_int

```

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

### 2.1.2 Working with tokens

`\g__unravel_input_int` The user input, at each stage of expansion, is stored in multiple `gtl` variables, from `\g_@@_input_⟨n⟩_gtl` to `\g__unravel_input_1_gtl`. The split between variables is akin to T<sub>E</sub>X'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.

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

```
\g__unravel_input_tmpa_int
\l__unravel_input_tmpa_tl
29 \int_new:N \g__unravel_input_tmpa_int
30 \tl_new:N \l__unravel_input_tmpa_tl
(End definition for \g__unravel_input_tmpa_int. This function is documented on page ??.)
```

```
\g__unravel_prev_input_seq
\l__unravel_prev_input_tl
\l__unravel_prev_input_gtl
31 \seq_new:N \g__unravel_prev_input_seq
32 \tl_new:N \l__unravel_prev_input_tl
33 \gtl_new:N \l__unravel_prev_input_gtl
(End definition for \g__unravel_prev_input_seq, \l__unravel_prev_input_tl, and \l__unravel_prev_input_gtl.
These variables are documented on page ??.)
```

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

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

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

```
\l__unravel_head_tl
\l__unravel_head_token
\l__unravel_head_cmd_int
\l__unravel_head_char_int
35 \gtl_new:N \l__unravel_head_gtl
36 \tl_new:N \l__unravel_head_tl
37 \token_new:Nn \l__unravel_head_token { ? }
38 \int_new:N \l__unravel_head_cmd_int
39 \int_new:N \l__unravel_head_char_int
(End definition for \l__unravel_head_gtl and others. These variables are documented on page ??.)
```

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

`\l__unravel_tmpa_tl` Temporary storage.

```
\l__unravel_tmpb_gtl
\g__unravel_tmpc_tl
\l__unravel_tmpa_seq
41 \tl_new:N \l__unravel_tmpa_tl
42 \gtl_new:N \l__unravel_tmpb_gtl
43 \tl_new:N \g__unravel_tmpc_tl
44 \seq_new:N \l__unravel_tmpa_seq
(End definition for \l__unravel_tmpa_tl and others. These variables are documented on page ??.)
```

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

```
45 \tl_new:N \l_unravel_defined_tl
46 \tl_new:N \l_unravel_defining_tl
```

(*End definition for `\l_unravel_defined_tl` and `\l_unravel_defining_tl`. These variables are documented on page ??.*)

`\_unravel_inaccessible:w`

```
47 \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 assignment. Is `\setbox` currently allowed? Should `\input` expand?

```
\g_unravel_set_box_allowed_bool
\g_unravel_name_in_progress_bool
48 \gtl_new:N \g_unravel_after_assignment_gtl
49 \bool_new:N \g_unravel_set_box_allowed_bool
50 \bool_new:N \g_unravel_name_in_progress_bool
```

(*End definition for `\g_unravel_after_assignment_gtl`, `\g_unravel_set_box_allowed_bool`, and `\g_unravel_name_in_progress_bool`. These variables are documented on page ??.*)

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

```
51 \group_begin:
52 \char_set_lccode:nn { '*' } { '# }
53 \tex_lowercase:D
54 { \tl_const:Nn \c_unravel_parameters_tl { ^*1*2*3*4*5*6*7*8*9 } }
55 \group_end:
```

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

### 2.1.3 Numbers and conditionals

`\g_unravel_val_level_int` See T<sub>E</sub>X'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.

```
56 \int_new:N \g_unravel_val_level_int
```

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

```
\c__unravel_plus_tl
\c__unravel_minus_tl 57 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 58 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 59 \tl_const:Nn \c__unravel_times_tl { * }
  \c__unravel_lq_tl 60 \tl_const:Nn \c__unravel_over_tl { / }
  \c__unravel_rq_tl 61 \tl_const:Nn \c__unravel_lq_tl { ‘ }
  \c__unravel_dq_tl 62 \tl_const:Nn \c__unravel_rq_tl { ’ }
  \c__unravel_lp_tl 63 \tl_const:Nn \c__unravel_dq_tl { " }
  \c__unravel_rp_tl 64 \tl_const:Nn \c__unravel_lp_tl { ( }
  \c__unravel_eq_tl 65 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 66 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 67 \tl_const:Nn \c__unravel_comma_tl { , }
68 \tl_const:Nn \c__unravel_point_tl { . }
```

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

`\g__unravel_if_limit_tl` Stack for what T<sub>E</sub>X calls `if_limit`, and its depth.

```
\g__unravel_if_limit_int 69 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 70 \int_new:N \g__unravel_if_limit_int
71 \int_new:N \g__unravel_if_depth_int
```

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

`\l__unravel_if_nesting_int`

```
72 \int_new:N \l__unravel_if_nesting_int
```

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

#### 2.1.4 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.

```
73 \seq_new:N \l__unravel_leaders_box_seq
```

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

`\g__unravel_ends_int`

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

```
74 \int_new:N \g__unravel_ends_int
75 \int_gset:Nn \g__unravel_ends_int { 3 }
```

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

#### 2.1.5 Constants

`\c__unravel_frozen_relax_gtl`

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

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

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



## 2.2 Variants and helper functions

Variants that we need.

```

77 \cs_if_exist:NF \exp_last_unbraced:NNn
78   { \cs_new_eq:NN \exp_last_unbraced:NNn \use:nnn }
79 \cs_generate_variant:Nn \exp_last_unbraced:NNn { NNv }
80 \cs_generate_variant:Nn \str_head:n { f }
81 \cs_generate_variant:Nn \tl_to_str:n { o }
82 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
83 \cs_generate_variant:Nn \tl_if_head_is_space:nTF { o }
84 \cs_generate_variant:Nn \tl_if_head_is_space:nT { V }
85 \cs_generate_variant:Nn \tl_if_head_is_N_type:nTF { o }
86 \cs_generate_variant:Nn \tl_if_in:nnF { nV }
87 \cs_generate_variant:Nn \tl_if_in:nnTF { nV }
88 \cs_generate_variant:Nn \tl_if_eq:nnT { V }
89 \cs_generate_variant:Nn \tl_if_in:NnTF { No , NV }
90 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
91 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nnx }
92 \cs_generate_variant:Nn \gtl_gput_left:Nn { Nx , NV , No }
93 \cs_generate_variant:Nn \gtl_gput_right:Nn { Nx , NV }
94 \cs_generate_variant:Nn \gtl_put_right:Nn { NV }
95 \cs_generate_variant:Nn \ior_get_str:NN { Nc }
96 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
97 \cs_generate_variant:Nn \gtl_to_str:N { c }
98 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
99 \cs_generate_variant:Nn \gtl_get_left:NN { c }
100 \cs_generate_variant:Nn \gtl_gset:Nn { c }
101 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
102 \cs_generate_variant:Nn \gtl_gclear:N { c }

```

```
\__unravel_tl_gset_input:Nnn
```

```
\__unravel_tl_gset_input:Nno
```

```
  \__unravel_tl_gset_input_aux:wN
```

```

103 \cs_new_protected:Npn \__unravel_tl_gset_input:Nnn #1#2#3
104   {
105     \group_begin:
106       \etex_everyeof:D \exp_after:wN { \token_to_str:N @ @ #1 }
107       #2
108       \tl_gclear:N #1
109       \str_if_eq_x:nnF
110         { \token_to_meaning:N \tex_input:D }
111         { \token_to_str:N \input }
112         { \msg_unravel:nx { unravel } { internal } { input-prim } }
113       \exp_after:wN \__unravel_tl_gset_input_aux:wN
114       \exp_after:wN \prg_do_nothing:
115       \tex_input:D \tl_to_str:n {#3} \scan_stop:
116     \group_end:
117     \tl_gput_right:NV #1 \etex_everyeof:D
118   }
119 \cs_generate_variant:Nn \__unravel_tl_gset_input:Nnn { Nno }
120 \use:x
121   {

```

```

122 \cs_new_protected:Npn \exp_not:N \__unravel_tl_gset_input_aux:wN
123   ##1 \token_to_str:N @ @ ##2
124 } { \tl_gset:No #2 {#1} }

```

(End definition for `\__unravel_tl_gset_input:Nnn` and `\__unravel_tl_gset_input:Nno`. These functions are documented on page ??.)

`\__unravel_cs_case:NnF` Currently, `expl3` does not provide a case statement for the meaning of control sequences. However, `\tl_case:NnF` in fact does precisely this.

```

125 \cs_new_eq:NN \__unravel_cs_case:NnF \tl_case:NnF

```

(End definition for `\__unravel_cs_case:NnF`. This function is documented on page ??.)

`\__unravel_strip_escape:w` This is based on the 2013-07-19 (and earlier) version of `\cs_to_str:N`. There are three cases. If the escape character is printable, the charcode test is false, and `\__unravel_strip_escape_aux:N` removes one character. If the escape character is a space, the charcode test is true, and if there is no escape character, the test is unfinished after `\token_to_str:N \ .` In both of those cases, `\__unravel_strip_escape_aux:w` inserts `-\__int_value:w \fi: \c_zero`. If the escape character was a space, the test was true, and `\__int_value:w` converts `\c_zero` to 0, hence the leading roman numeral expansion removes a space from what follows (it is important that what follows cannot start with a digit). Otherwise, the test takes `-` as its second operand, is false, and the roman numeral expansion only sees `\c_zero`, thus does not remove anything from what follows.

```

126 \cs_new_nopar:Npn \__unravel_strip_escape:w
127 {
128   \tex_romannumeral:D
129   \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
130   \__unravel_strip_escape_aux:N
131 }
132 \cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero }
133 \cs_new:Npn \__unravel_strip_escape_aux:w #1#2
134 { - \__int_value:w #1 \c_zero }

```

(End definition for `\__unravel_strip_escape:w`. This function is documented on page ??.)

`\__unravel_token_to_char:N`  
`\__unravel_meaning_to_char:n`  
`\__unravel_meaning_to_char:o` 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_auxi:w
\__unravel_meaning_to_char_auxii:w
135 \cs_new:Npn \__unravel_meaning_to_char:n #1
136 { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
137 \cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
138 { \__unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
139 \cs_new:Npn \__unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
140 { \tl_if_empty:nTF {#2} { ~ } {#2} }
141 \cs_generate_variant:Nn \__unravel_meaning_to_char:n { o }
142 \cs_new:Npn \__unravel_token_to_char:N #1
143 { \__unravel_meaning_to_char:o { \token_to_meaning:N #1 } }

```

(End definition for `\__unravel_token_to_char:N`. This function is documented on page ??.)

```

    \__unravel_to_str:n Use the type-appropriate conversion to string.
    \__unravel_to_str_auxi:w 144 \cs_new:Npn \__unravel_to_str:n #1
    \__unravel_to_str_auxii:w 145 {
    146   \tl_if_head_eq_meaning:nNTF {#1} \scan_stop:
    147     { \__unravel_to_str_auxi:w #1 ? \q_stop }
    148     { \tl_to_str:n }
    149     {#1}
    150   }
    151 \cs_set:Npn \__unravel_tmp:w #1
    152 {
    153   \cs_new:Npn \__unravel_to_str_auxi:w ##1##2 \q_stop
    154     {
    155       \exp_after:wN \__unravel_to_str_auxii:w \token_to_str:N ##1 \q_mark
    156       #1 \tl \q_mark \q_stop
    157     }
    158   \cs_new:Npn \__unravel_to_str_auxii:w ##1 #1 ##2 \q_mark ##3 \q_stop
    159     { \cs_if_exist_use:cF { ##2 _to_str:n } { \tl_to_str:n } }
    160   }
    161 \exp_args:No \__unravel_tmp:w { \tl_to_str:n { s__ } }

```

(End definition for \\_\_unravel\_to\_str:n. This function is documented on page ??.)

```

    \__unravel_prev_input_silent:n
    \__unravel_prev_input_silent:V 162 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
    \__unravel_prev_input_silent:x 163 {
    164   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_tl
    165   \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
    166   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_tl
    167   }
    168 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V , x }
    169 \cs_new_protected:Npn \__unravel_prev_input:n #1
    170 {
    171   \__unravel_prev_input_silent:n {#1}
    172   \__unravel_print_action:x { \tl_to_str:n {#1} }
    173   }
    174 \cs_generate_variant:Nn \__unravel_prev_input:n { V , x }

```

(End definition for \\_\_unravel\_prev\_input\_silent:n, \\_\_unravel\_prev\_input\_silent:V, and \\_\_unravel\_prev\_input\_silent:n. These functions are documented on page ??.)

```

\__unravel_prev_input_gtl:N
175 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
176 {
177   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
178   \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
179   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_prev_input_gtl
180   }

```

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

\\_\_unravel\_token\_if\_expandable\_p:N We need to cook up our own version of \token\_if\_expandable:NTF because the expl3  
 \\_\_unravel\_token\_if\_expandable:NTF one does not think that undefined is expandable.

```

181 \prg_new_conditional:Npnn \__unravel_token_if_expandable:N #1
182 { p , T , F , TF }
183 {
184   \exp_after:wN \if_meaning:w \exp_not:N #1 #1
185   \prg_return_false:
186   \else:
187   \prg_return_true:
188   \fi:
189 }

```

(End definition for \\_\_unravel\_token\_if\_expandable:N.)

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

```

\__unravel_token_if_protected:NTF
190 \prg_new_conditional:Npnn \__unravel_token_if_protected:N #1
191 { p , T , F , TF }
192 {
193   \__unravel_token_if_expandable:NTF #1
194   {
195     \token_if_protected_macro:NTF #1
196     { \prg_return_true: }
197     {
198       \token_if_protected_long_macro:NTF #1
199       { \prg_return_true: }
200       { \prg_return_false: }
201     }
202   }
203   { \prg_return_true: }
204 }

```

(End definition for \\_\_unravel\_token\_if\_protected:N.)

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

```

\__unravel_exit_point:
205 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
206 \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:
207 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
208 \cs_new:Npn \__unravel_break:w #1 \__unravel_break_point: { }

```

(End definition for \\_\_unravel\_break:w and \\_\_unravel\_break\_point:.)

`\__unravel_token_if_definable:NTF` Within a group, set the escape character to a non-zero non-space value (backslash): if the result of converting the token to a string is longer than a single token (false branch), the token was a control sequence and the test is true (note that we did not use `\tl_to_str:n`, as this would fail on explicit macro parameter characters). Otherwise, the token was an explicit character, active or not. Use `\lowercase` to convert the character to a fixed character code Z. Compare with an active Z. In all three outcomes, remember to end the group.

```

209 \group_begin:
210 \char_set_catcode_active:n { 'Z }

```

```

211 \prg_new_protected_conditional:Npnm \__unravel_token_if_definable:N #1
212 { TF }
213 {
214   \group_begin:
215   \int_set:Nn \tex_escapechar:D { 92 }
216   \exp_args:No \tl_if_single:nTF { \token_to_str:N #1 }
217   {
218     \exp_args:Nx \char_set_lccode:nn
219     { ' \str_head:n {#1} } { ' Z }
220     \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
221     { \group_end: \prg_return_true: }
222     { \group_end: \prg_return_false: }
223   }
224   { \group_end: \prg_return_true: }
225 }
226 \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.

```

227 \prg_new_protected_conditional:Npnm \__unravel_gtl_if_head_is_definable:N #1
228 { TF , F }
229 {
230   \gtl_if_single_token:NTF #1
231   {
232     \gtl_if_head_is_N_type:NTF #1
233     {
234       \exp_last_unbraced:Nx \__unravel_token_if_definable:NTF
235       { \gtl_head:N #1 }
236       { \prg_return_true: }
237       { \prg_return_false: }
238     }
239     { \prg_return_false: }
240   }
241   { \prg_return_false: }
242 }

```

*(End definition for \\_\_unravel\_gtl\_if\_head\_is\_definable:NTF.)*

## 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
243 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
244 { \int_const:cn { c__unravel_tex_#1_int } {#2} }

```

```

245 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }
(End definition for \__unravel_tex_const:nn. This function is documented on page ??.)

```

\\_\_unravel\_tex\_primitive:nnn

```

246 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
247 {
248   \tl_const:cx { c__unravel_tex_#1_tl }
249   { { \__unravel_tex_use:n {#2} } {#3} }
250 }
(End definition for \__unravel_tex_primitive:nnn.)

```

\\_\_unravel\_new\_tex\_cmd:nn  
\\_\_unravel\_new\_eq\_tex\_cmd:nn

```

251 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
252 {
253   \cs_new_protected_nopar:cpn
254   { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
255 }
256 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
257 {
258   \cs_new_eq:cc
259   { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
260   { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
261 }
(End definition for \__unravel_new_tex_cmd:nn and \__unravel_new_eq_tex_cmd:nn.)

```

\\_\_unravel\_new\_tex\_expandable:nn

```

262 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
263 {
264   \cs_new_protected_nopar:cpn
265   { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
266 }
(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.

```

267 \__unravel_tex_const:nn { relax } { 0 }
268 \__unravel_tex_const:nn { begin-group_char } { 1 }
269 \__unravel_tex_const:nn { end-group_char } { 2 }
270 \__unravel_tex_const:nn { math_char } { 3 }
271 \__unravel_tex_const:nn { tab_mark } { 4 }
272 \__unravel_tex_const:nn { alignment_char } { 4 }
273 \__unravel_tex_const:nn { car_ret } { 5 }
274 \__unravel_tex_const:nn { macro_char } { 6 }
275 \__unravel_tex_const:nn { superscript_char } { 7 }
276 \__unravel_tex_const:nn { subscript_char } { 8 }
277 \__unravel_tex_const:nn { endv } { 9 }
278 \__unravel_tex_const:nn { blank_char } { 10 }
279 \__unravel_tex_const:nn { the_char } { 11 }
280 \__unravel_tex_const:nn { other_char } { 12 }
281 \__unravel_tex_const:nn { par_end } { 13 }

```

```

282 \__unravel_tex_const:nn { stop } { 14 }
283 \__unravel_tex_const:nn { delim_num } { 15 }
284 \__unravel_tex_const:nn { max_char_code } { 15 }
285 \__unravel_tex_const:nn { char_num } { 16 }
286 \__unravel_tex_const:nn { math_char_num } { 17 }
287 \__unravel_tex_const:nn { mark } { 18 }
288 \__unravel_tex_const:nn { xray } { 19 }
289 \__unravel_tex_const:nn { make_box } { 20 }
290 \__unravel_tex_const:nn { hmove } { 21 }
291 \__unravel_tex_const:nn { vmove } { 22 }
292 \__unravel_tex_const:nn { un_hbox } { 23 }
293 \__unravel_tex_const:nn { un_vbox } { 24 }
294 \__unravel_tex_const:nn { remove_item } { 25 }
295 \__unravel_tex_const:nn { hskip } { 26 }
296 \__unravel_tex_const:nn { vskip } { 27 }
297 \__unravel_tex_const:nn { mskip } { 28 }
298 \__unravel_tex_const:nn { kern } { 29 }
299 \__unravel_tex_const:nn { mkern } { 30 }
300 \__unravel_tex_const:nn { leader_ship } { 31 }
301 \__unravel_tex_const:nn { halign } { 32 }
302 \__unravel_tex_const:nn { valign } { 33 }
303 \__unravel_tex_const:nn { no_align } { 34 }
304 \__unravel_tex_const:nn { vrule } { 35 }
305 \__unravel_tex_const:nn { hrule } { 36 }
306 \__unravel_tex_const:nn { insert } { 37 }
307 \__unravel_tex_const:nn { vadjust } { 38 }
308 \__unravel_tex_const:nn { ignore_spaces } { 39 }
309 \__unravel_tex_const:nn { after_assignment } { 40 }
310 \__unravel_tex_const:nn { after_group } { 41 }
311 \__unravel_tex_const:nn { break_penalty } { 42 }
312 \__unravel_tex_const:nn { start_par } { 43 }
313 \__unravel_tex_const:nn { ital_corr } { 44 }
314 \__unravel_tex_const:nn { accent } { 45 }
315 \__unravel_tex_const:nn { math_accent } { 46 }
316 \__unravel_tex_const:nn { discretionary } { 47 }
317 \__unravel_tex_const:nn { eq_no } { 48 }
318 \__unravel_tex_const:nn { left_right } { 49 }
319 \__unravel_tex_const:nn { math_comp } { 50 }
320 \__unravel_tex_const:nn { limit_switch } { 51 }
321 \__unravel_tex_const:nn { above } { 52 }
322 \__unravel_tex_const:nn { math_style } { 53 }
323 \__unravel_tex_const:nn { math_choice } { 54 }
324 \__unravel_tex_const:nn { non_script } { 55 }
325 \__unravel_tex_const:nn { vcenter } { 56 }
326 \__unravel_tex_const:nn { case_shift } { 57 }
327 \__unravel_tex_const:nn { message } { 58 }
328 \__unravel_tex_const:nn { extension } { 59 }
329 \__unravel_tex_const:nn { in_stream } { 60 }
330 \__unravel_tex_const:nn { begin_group } { 61 }
331 \__unravel_tex_const:nn { end_group } { 62 }

```

```

332 \__unravel_tex_const:nn { omit } { 63 }
333 \__unravel_tex_const:nn { ex_space } { 64 }
334 \__unravel_tex_const:nn { no_boundary } { 65 }
335 \__unravel_tex_const:nn { radical } { 66 }
336 \__unravel_tex_const:nn { end_cs_name } { 67 }
337 \__unravel_tex_const:nn { min_internal } { 68 }
338 \__unravel_tex_const:nn { char_given } { 68 }
339 \__unravel_tex_const:nn { math_given } { 69 }
340 \__unravel_tex_const:nn { last_item } { 70 }
341 \__unravel_tex_const:nn { max_non_prefixed_command } { 70 }
342 \__unravel_tex_const:nn { toks_register } { 71 }
343 \__unravel_tex_const:nn { assign_toks } { 72 }
344 \__unravel_tex_const:nn { assign_int } { 73 }
345 \__unravel_tex_const:nn { assign_dimen } { 74 }
346 \__unravel_tex_const:nn { assign_glue } { 75 }
347 \__unravel_tex_const:nn { assign_mu_glue } { 76 }
348 \__unravel_tex_const:nn { assign_font_dimen } { 77 }
349 \__unravel_tex_const:nn { assign_font_int } { 78 }
350 \__unravel_tex_const:nn { set_aux } { 79 }
351 \__unravel_tex_const:nn { set_prev_graf } { 80 }
352 \__unravel_tex_const:nn { set_page_dimen } { 81 }
353 \__unravel_tex_const:nn { set_page_int } { 82 }
354 \__unravel_tex_const:nn { set_box_dimen } { 83 }
355 \__unravel_tex_const:nn { set_shape } { 84 }
356 \__unravel_tex_const:nn { def_code } { 85 }
357 \__unravel_tex_const:nn { def_family } { 86 }
358 \__unravel_tex_const:nn { set_font } { 87 }
359 \__unravel_tex_const:nn { def_font } { 88 }
360 \__unravel_tex_const:nn { register } { 89 }
361 \__unravel_tex_const:nn { max_internal } { 89 }
362 \__unravel_tex_const:nn { advance } { 90 }
363 \__unravel_tex_const:nn { multiply } { 91 }
364 \__unravel_tex_const:nn { divide } { 92 }
365 \__unravel_tex_const:nn { prefix } { 93 }
366 \__unravel_tex_const:nn { let } { 94 }
367 \__unravel_tex_const:nn { shorthand_def } { 95 }
368 \__unravel_tex_const:nn { read_to_cs } { 96 }
369 \__unravel_tex_const:nn { def } { 97 }
370 \__unravel_tex_const:nn { set_box } { 98 }
371 \__unravel_tex_const:nn { hyph_data } { 99 }
372 \__unravel_tex_const:nn { set_interaction } { 100 }
373 \__unravel_tex_const:nn { letterspace_font } { 101 }
374 \__unravel_tex_const:nn { pdf_copy_font } { 102 }
375 \__unravel_tex_const:nn { max_command } { 102 }
376 \__unravel_tex_const:nn { undefined_cs } { 103 }
377 \__unravel_tex_const:nn { expand_after } { 104 }
378 \__unravel_tex_const:nn { no_expand } { 105 }
379 \__unravel_tex_const:nn { input } { 106 }
380 \__unravel_tex_const:nn { if_test } { 107 }
381 \__unravel_tex_const:nn { fi_or_else } { 108 }

```



```

382 \__unravel_tex_const:nn { cs_name           } { 109 }
383 \__unravel_tex_const:nn { convert          } { 110 }
384 \__unravel_tex_const:nn { the              } { 111 }
385 \__unravel_tex_const:nn { top_bot_mark    } { 112 }
386 \__unravel_tex_const:nn { call            } { 113 }
387 \__unravel_tex_const:nn { end_template    } { 117 }

```

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

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

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

```

388 \__unravel_tex_primitive:nnn { relax           } { relax   } { 256 }
389 \__unravel_tex_primitive:nnn { span           } { tab_mark } { 256 }
390 \__unravel_tex_primitive:nnn { cr             } { car_ret  } { 257 }
391 \__unravel_tex_primitive:nnn { crcr          } { car_ret  } { 258 }
392 \__unravel_tex_primitive:nnn { par           } { par_end  } { 256 }
393 \__unravel_tex_primitive:nnn { end            } { stop     } { 0 }
394 \__unravel_tex_primitive:nnn { dump          } { stop     } { 1 }
395 \__unravel_tex_primitive:nnn { delimiter      } { delim_num } { 0 }
396 \__unravel_tex_primitive:nnn { char          } { char_num } { 0 }
397 \__unravel_tex_primitive:nnn { mathchar      } { math_char_num } { 0 }
398 \__unravel_tex_primitive:nnn { mark          } { mark     } { 0 }
399 \__unravel_tex_primitive:nnn { marks         } { mark     } { 5 }
400 \__unravel_tex_primitive:nnn { show          } { xray     } { 0 }
401 \__unravel_tex_primitive:nnn { showbox       } { xray     } { 1 }
402 \__unravel_tex_primitive:nnn { showthe      } { xray     } { 2 }
403 \__unravel_tex_primitive:nnn { showlists    } { xray     } { 3 }
404 \__unravel_tex_primitive:nnn { showgroups   } { xray     } { 4 }
405 \__unravel_tex_primitive:nnn { showtokens   } { xray     } { 5 }
406 \__unravel_tex_primitive:nnn { showifs      } { xray     } { 6 }
407 \__unravel_tex_primitive:nnn { box          } { make_box } { 0 }
408 \__unravel_tex_primitive:nnn { copy         } { make_box } { 1 }
409 \__unravel_tex_primitive:nnn { lastbox      } { make_box } { 2 }
410 \__unravel_tex_primitive:nnn { vsplit       } { make_box } { 3 }
411 \__unravel_tex_primitive:nnn { vtop        } { make_box } { 4 }
412 \__unravel_tex_primitive:nnn { vbox        } { make_box } { 5 }
413 \__unravel_tex_primitive:nnn { hbox        } { make_box } { 106 }

```

```

414 \__unravel_tex_primitive:nnn { moveright } { hmove } { 0 }
415 \__unravel_tex_primitive:nnn { moveleft } { hmove } { 1 }
416 \__unravel_tex_primitive:nnn { lower } { vmove } { 0 }
417 \__unravel_tex_primitive:nnn { raise } { vmove } { 1 }
418 \__unravel_tex_primitive:nnn { unhbox } { un_hbox } { 0 }
419 \__unravel_tex_primitive:nnn { unhcopy } { un_hbox } { 1 }
420 \__unravel_tex_primitive:nnn { unvbox } { un_vbox } { 0 }
421 \__unravel_tex_primitive:nnn { unvcopy } { un_vbox } { 1 }
422 \__unravel_tex_primitive:nnn { pagediscards } { un_vbox } { 2 }
423 \__unravel_tex_primitive:nnn { splitdiscards } { un_vbox } { 3 }
424 \__unravel_tex_primitive:nnn { unpenalty } { remove_item } { 12 }
425 \__unravel_tex_primitive:nnn { unkern } { remove_item } { 11 }
426 \__unravel_tex_primitive:nnn { unskip } { remove_item } { 10 }
427 \__unravel_tex_primitive:nnn { hfil } { hskip } { 0 }
428 \__unravel_tex_primitive:nnn { hfill } { hskip } { 1 }
429 \__unravel_tex_primitive:nnn { hss } { hskip } { 2 }
430 \__unravel_tex_primitive:nnn { hfilneg } { hskip } { 3 }
431 \__unravel_tex_primitive:nnn { hskip } { hskip } { 4 }
432 \__unravel_tex_primitive:nnn { vfil } { vskip } { 0 }
433 \__unravel_tex_primitive:nnn { vfill } { vskip } { 1 }
434 \__unravel_tex_primitive:nnn { vss } { vskip } { 2 }
435 \__unravel_tex_primitive:nnn { vfilneg } { vskip } { 3 }
436 \__unravel_tex_primitive:nnn { vskip } { vskip } { 4 }
437 \__unravel_tex_primitive:nnn { mskip } { mskip } { 5 }
438 \__unravel_tex_primitive:nnn { kern } { kern } { 1 }
439 \__unravel_tex_primitive:nnn { mkern } { mkern } { 99 }
440 \__unravel_tex_primitive:nnn { shipout } { leader_ship } { 99 }
441 \__unravel_tex_primitive:nnn { leaders } { leader_ship } { 100 }
442 \__unravel_tex_primitive:nnn { cleaders } { leader_ship } { 101 }
443 \__unravel_tex_primitive:nnn { xleaders } { leader_ship } { 102 }
444 \__unravel_tex_primitive:nnn { halign } { halign } { 0 }
445 \__unravel_tex_primitive:nnn { valign } { valign } { 0 }
446 \__unravel_tex_primitive:nnn { beginL } { valign } { 4 }
447 \__unravel_tex_primitive:nnn { endL } { valign } { 5 }
448 \__unravel_tex_primitive:nnn { beginR } { valign } { 8 }
449 \__unravel_tex_primitive:nnn { endR } { valign } { 9 }
450 \__unravel_tex_primitive:nnn { noalign } { no_align } { 0 }
451 \__unravel_tex_primitive:nnn { vrule } { vrule } { 0 }
452 \__unravel_tex_primitive:nnn { hrule } { hrule } { 0 }
453 \__unravel_tex_primitive:nnn { insert } { insert } { 0 }
454 \__unravel_tex_primitive:nnn { vadjust } { vadjust } { 0 }
455 \__unravel_tex_primitive:nnn { ignorespaces } { ignore_spaces } { 0 }
456 \__unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
457 \__unravel_tex_primitive:nnn { aftergroup } { after_group } { 0 }
458 \__unravel_tex_primitive:nnn { penalty } { break_penalty } { 0 }
459 \__unravel_tex_primitive:nnn { indent } { start_par } { 1 }
460 \__unravel_tex_primitive:nnn { noindent } { start_par } { 0 }
461 \__unravel_tex_primitive:nnn { quitvmode } { start_par } { 2 }
462 \__unravel_tex_primitive:nnn { / } { ital_corr } { 0 }
463 \__unravel_tex_primitive:nnn { accent } { accent } { 0 }

```

```

464 \__unravel_tex_primitive:nnn { mathaccent          } { math_accent } { 0 }
465 \__unravel_tex_primitive:nnn { -                  } { discretionary } { 1 }
466 \__unravel_tex_primitive:nnn { discretionary      } { discretionary } { 0 }
467 \__unravel_tex_primitive:nnn { eqno               } { eq_no } { 0 }
468 \__unravel_tex_primitive:nnn { leqno              } { eq_no } { 1 }
469 \__unravel_tex_primitive:nnn { left               } { left_right } { 30 }
470 \__unravel_tex_primitive:nnn { right              } { left_right } { 31 }
471 \__unravel_tex_primitive:nnn { middle             } { left_right } { 17 }
472 \__unravel_tex_primitive:nnn { mathord            } { math_comp } { 16 }
473 \__unravel_tex_primitive:nnn { mathop             } { math_comp } { 17 }
474 \__unravel_tex_primitive:nnn { mathbin            } { math_comp } { 18 }
475 \__unravel_tex_primitive:nnn { mathrel            } { math_comp } { 19 }
476 \__unravel_tex_primitive:nnn { mathopen           } { math_comp } { 20 }
477 \__unravel_tex_primitive:nnn { mathclose          } { math_comp } { 21 }
478 \__unravel_tex_primitive:nnn { mathpunct          } { math_comp } { 22 }
479 \__unravel_tex_primitive:nnn { mathinner          } { math_comp } { 23 }
480 \__unravel_tex_primitive:nnn { underline           } { math_comp } { 26 }
481 \__unravel_tex_primitive:nnn { overline            } { math_comp } { 27 }
482 \__unravel_tex_primitive:nnn { displaylimits      } { limit_switch } { 0 }
483 \__unravel_tex_primitive:nnn { limits              } { limit_switch } { 1 }
484 \__unravel_tex_primitive:nnn { nolimits            } { limit_switch } { 2 }
485 \__unravel_tex_primitive:nnn { above              } { above } { 0 }
486 \__unravel_tex_primitive:nnn { over               } { above } { 1 }
487 \__unravel_tex_primitive:nnn { atop               } { above } { 2 }
488 \__unravel_tex_primitive:nnn { abovewithdelims    } { above } { 3 }
489 \__unravel_tex_primitive:nnn { overwithdelims     } { above } { 4 }
490 \__unravel_tex_primitive:nnn { atopwithdelims     } { above } { 5 }
491 \__unravel_tex_primitive:nnn { displaystyle       } { math_style } { 0 }
492 \__unravel_tex_primitive:nnn { textstyle          } { math_style } { 2 }
493 \__unravel_tex_primitive:nnn { scriptstyle        } { math_style } { 4 }
494 \__unravel_tex_primitive:nnn { scriptscriptstyle  } { math_style } { 6 }
495 \__unravel_tex_primitive:nnn { mathchoice         } { math_choice } { 0 }
496 \__unravel_tex_primitive:nnn { nonscript          } { non_script } { 0 }
497 \__unravel_tex_primitive:nnn { vcenter            } { vcenter } { 0 }
498 \__unravel_tex_primitive:nnn { lowercase          } { case_shift } { 256 }
499 \__unravel_tex_primitive:nnn { uppercase          } { case_shift } { 512 }
500 \__unravel_tex_primitive:nnn { message            } { message } { 0 }
501 \__unravel_tex_primitive:nnn { errmessage         } { message } { 1 }
502 \__unravel_tex_primitive:nnn { openout            } { extension } { 0 }
503 \__unravel_tex_primitive:nnn { write              } { extension } { 1 }
504 \__unravel_tex_primitive:nnn { closeout           } { extension } { 2 }
505 \__unravel_tex_primitive:nnn { special            } { extension } { 3 }
506 \__unravel_tex_primitive:nnn { immediate          } { extension } { 4 }
507 \__unravel_tex_primitive:nnn { setlanguage        } { extension } { 5 }
508 \__unravel_tex_primitive:nnn { pdfliteral         } { extension } { 6 }
509 \__unravel_tex_primitive:nnn { pdfobj             } { extension } { 7 }
510 \__unravel_tex_primitive:nnn { pdfrefobj          } { extension } { 8 }
511 \__unravel_tex_primitive:nnn { pdfxform           } { extension } { 9 }
512 \__unravel_tex_primitive:nnn { pdfrefxform        } { extension } { 10 }
513 \__unravel_tex_primitive:nnn { pdfximage          } { extension } { 11 }

```

```

514 \__unravel_tex_primitive:nnn { pdfrefximage      } { extension } { 12 }
515 \__unravel_tex_primitive:nnn { pdfannot        } { extension } { 13 }
516 \__unravel_tex_primitive:nnn { pdfstartlink   } { extension } { 14 }
517 \__unravel_tex_primitive:nnn { pdfendlink     } { extension } { 15 }
518 \__unravel_tex_primitive:nnn { pdfoutline     } { extension } { 16 }
519 \__unravel_tex_primitive:nnn { pdfdest        } { extension } { 17 }
520 \__unravel_tex_primitive:nnn { pdfthread      } { extension } { 18 }
521 \__unravel_tex_primitive:nnn { pdfstartthread } { extension } { 19 }
522 \__unravel_tex_primitive:nnn { pdfendthread   } { extension } { 20 }
523 \__unravel_tex_primitive:nnn { pdfsavepos     } { extension } { 21 }
524 \__unravel_tex_primitive:nnn { pdfinfo        } { extension } { 22 }
525 \__unravel_tex_primitive:nnn { pdfcatalog     } { extension } { 23 }
526 \__unravel_tex_primitive:nnn { pdfnames       } { extension } { 24 }
527 \__unravel_tex_primitive:nnn { pdffontattr    } { extension } { 25 }
528 \__unravel_tex_primitive:nnn { pdfincludechars } { extension } { 26 }
529 \__unravel_tex_primitive:nnn { pdfmapfile     } { extension } { 27 }
530 \__unravel_tex_primitive:nnn { pdfmapline     } { extension } { 28 }
531 \__unravel_tex_primitive:nnn { pdftrailer     } { extension } { 29 }
532 \__unravel_tex_primitive:nnn { pdfresettimer  } { extension } { 30 }
533 \__unravel_tex_primitive:nnn { pdffontexpand  } { extension } { 31 }
534 \__unravel_tex_primitive:nnn { pdfsetrandomseed } { extension } { 32 }
535 \__unravel_tex_primitive:nnn { pdfsnaprefpoint } { extension } { 33 }
536 \__unravel_tex_primitive:nnn { pdfsnapyp      } { extension } { 34 }
537 \__unravel_tex_primitive:nnn { pdfsnapypcomp  } { extension } { 35 }
538 \__unravel_tex_primitive:nnn { pdfglyphtounicode } { extension } { 36 }
539 \__unravel_tex_primitive:nnn { pdfcolorstack  } { extension } { 37 }
540 \__unravel_tex_primitive:nnn { pdfsetmatrix    } { extension } { 38 }
541 \__unravel_tex_primitive:nnn { pdfsave         } { extension } { 39 }
542 \__unravel_tex_primitive:nnn { pdfrestore      } { extension } { 40 }
543 \__unravel_tex_primitive:nnn { pdfnobuiltintounicode } { extension } { 41 }
544 \__unravel_tex_primitive:nnn { openin           } { in_stream } { 1 }
545 \__unravel_tex_primitive:nnn { closein            } { in_stream } { 0 }
546 \__unravel_tex_primitive:nnn { begingroup       } { begin_group } { 0 }
547 \__unravel_tex_primitive:nnn { endgroup         } { end_group } { 0 }
548 \__unravel_tex_primitive:nnn { omit             } { omit } { 0 }
549 \__unravel_tex_primitive:nnn { ~                 } { ex_space } { 0 }
550 \__unravel_tex_primitive:nnn { noboundary          } { no_boundary } { 0 }
551 \__unravel_tex_primitive:nnn { radical           } { radical } { 0 }
552 \__unravel_tex_primitive:nnn { endcsname         } { end_cs_name } { 0 }
553 \__unravel_tex_primitive:nnn { lastpenalty       } { last_item } { 0 }
554 \__unravel_tex_primitive:nnn { lastkern           } { last_item } { 1 }
555 \__unravel_tex_primitive:nnn { lastskip          } { last_item } { 2 }
556 \__unravel_tex_primitive:nnn { lastnodetype     } { last_item } { 3 }
557 \__unravel_tex_primitive:nnn { inputlineno     } { last_item } { 4 }
558 \__unravel_tex_primitive:nnn { badness          } { last_item } { 5 }
559 \__unravel_tex_primitive:nnn { pdftexversion   } { last_item } { 6 }
560 \__unravel_tex_primitive:nnn { pdflastobj       } { last_item } { 7 }
561 \__unravel_tex_primitive:nnn { pdflastxform     } { last_item } { 8 }
562 \__unravel_tex_primitive:nnn { pdflastximage    } { last_item } { 9 }
563 \__unravel_tex_primitive:nnn { pdflastximagepages } { last_item } { 10 }

```

```

564 \__unravel_tex_primitive:nnn { pdflastannot          } { last_item } { 11 }
565 \__unravel_tex_primitive:nnn { pdflastxpos          } { last_item } { 12 }
566 \__unravel_tex_primitive:nnn { pdflastypos          } { last_item } { 13 }
567 \__unravel_tex_primitive:nnn { pdfretval            } { last_item } { 14 }
568 \__unravel_tex_primitive:nnn { pdflastximagecolordepth } { last_item } { 15 }
569 \__unravel_tex_primitive:nnn { pdfelapsedtime       } { last_item } { 16 }
570 \__unravel_tex_primitive:nnn { pdfshellescape     } { last_item } { 17 }
571 \__unravel_tex_primitive:nnn { pdfrandomseed       } { last_item } { 18 }
572 \__unravel_tex_primitive:nnn { pdflastlink         } { last_item } { 19 }
573 \__unravel_tex_primitive:nnn { eTeXversion         } { last_item } { 20 }
574 \__unravel_tex_primitive:nnn { currentgrouplevel   } { last_item } { 21 }
575 \__unravel_tex_primitive:nnn { currentgroupstype    } { last_item } { 22 }
576 \__unravel_tex_primitive:nnn { currentiflevel       } { last_item } { 23 }
577 \__unravel_tex_primitive:nnn { currentiftypetype    } { last_item } { 24 }
578 \__unravel_tex_primitive:nnn { currentifbranch      } { last_item } { 25 }
579 \__unravel_tex_primitive:nnn { gluestretchorder     } { last_item } { 26 }
580 \__unravel_tex_primitive:nnn { glueshrinkorder      } { last_item } { 27 }
581 \__unravel_tex_primitive:nnn { fontcharwd          } { last_item } { 28 }
582 \__unravel_tex_primitive:nnn { fontcharht          } { last_item } { 29 }
583 \__unravel_tex_primitive:nnn { fontchardp          } { last_item } { 30 }
584 \__unravel_tex_primitive:nnn { fontcharic          } { last_item } { 31 }
585 \__unravel_tex_primitive:nnn { parshapelength       } { last_item } { 32 }
586 \__unravel_tex_primitive:nnn { parshapeindent       } { last_item } { 33 }
587 \__unravel_tex_primitive:nnn { parshapedimen        } { last_item } { 34 }
588 \__unravel_tex_primitive:nnn { gluestretch         } { last_item } { 35 }
589 \__unravel_tex_primitive:nnn { glueshrink          } { last_item } { 36 }
590 \__unravel_tex_primitive:nnn { mutoglupe           } { last_item } { 37 }
591 \__unravel_tex_primitive:nnn { glueto_mu           } { last_item } { 38 }
592 \__unravel_tex_primitive:nnn { numexpr             } { last_item } { 39 }
593 \__unravel_tex_primitive:nnn { dimexpr             } { last_item } { 40 }
594 \__unravel_tex_primitive:nnn { glueexpr            } { last_item } { 41 }
595 \__unravel_tex_primitive:nnn { muexpr              } { last_item } { 42 }
596 \__unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
597 \__unravel_tex_primitive:nnn { output               } { assign_toks } { 1 }
598 \__unravel_tex_primitive:nnn { everypar             } { assign_toks } { 2 }
599 \__unravel_tex_primitive:nnn { everymath            } { assign_toks } { 3 }
600 \__unravel_tex_primitive:nnn { everydisplay         } { assign_toks } { 4 }
601 \__unravel_tex_primitive:nnn { everyhbox            } { assign_toks } { 5 }
602 \__unravel_tex_primitive:nnn { everyvbox            } { assign_toks } { 6 }
603 \__unravel_tex_primitive:nnn { everyjob             } { assign_toks } { 7 }
604 \__unravel_tex_primitive:nnn { everycr             } { assign_toks } { 8 }
605 \__unravel_tex_primitive:nnn { errhelp              } { assign_toks } { 9 }
606 \__unravel_tex_primitive:nnn { pdfpagesattr        } { assign_toks } { 10 }
607 \__unravel_tex_primitive:nnn { pdfpageattr         } { assign_toks } { 11 }
608 \__unravel_tex_primitive:nnn { pdfpageresources     } { assign_toks } { 12 }
609 \__unravel_tex_primitive:nnn { pdfpkmode           } { assign_toks } { 13 }
610 \__unravel_tex_primitive:nnn { everyeof             } { assign_toks } { 14 }
611 \__unravel_tex_primitive:nnn { pretolerance        } { assign_int } { 0 }
612 \__unravel_tex_primitive:nnn { tolerance           } { assign_int } { 1 }
613 \__unravel_tex_primitive:nnn { linepenalty          } { assign_int } { 2 }

```

```

614 \_ unravel\_tex\_primitive:nnn { hyphenpenalty } { assign\_int } { 3 }
615 \_ unravel\_tex\_primitive:nnn { exhyphenpenalty } { assign\_int } { 4 }
616 \_ unravel\_tex\_primitive:nnn { clubpenalty } { assign\_int } { 5 }
617 \_ unravel\_tex\_primitive:nnn { widowpenalty } { assign\_int } { 6 }
618 \_ unravel\_tex\_primitive:nnn { displaywidowpenalty } { assign\_int } { 7 }
619 \_ unravel\_tex\_primitive:nnn { brokenpenalty } { assign\_int } { 8 }
620 \_ unravel\_tex\_primitive:nnn { binoppenalty } { assign\_int } { 9 }
621 \_ unravel\_tex\_primitive:nnn { relpenalty } { assign\_int } { 10 }
622 \_ unravel\_tex\_primitive:nnn { predisplaypenalty } { assign\_int } { 11 }
623 \_ unravel\_tex\_primitive:nnn { postdisplaypenalty } { assign\_int } { 12 }
624 \_ unravel\_tex\_primitive:nnn { interlinepenalty } { assign\_int } { 13 }
625 \_ unravel\_tex\_primitive:nnn { doublehyphendemerits } { assign\_int } { 14 }
626 \_ unravel\_tex\_primitive:nnn { finalhyphendemerits } { assign\_int } { 15 }
627 \_ unravel\_tex\_primitive:nnn { adjdemerits } { assign\_int } { 16 }
628 \_ unravel\_tex\_primitive:nnn { mag } { assign\_int } { 17 }
629 \_ unravel\_tex\_primitive:nnn { delimiterfactor } { assign\_int } { 18 }
630 \_ unravel\_tex\_primitive:nnn { looseness } { assign\_int } { 19 }
631 \_ unravel\_tex\_primitive:nnn { time } { assign\_int } { 20 }
632 \_ unravel\_tex\_primitive:nnn { day } { assign\_int } { 21 }
633 \_ unravel\_tex\_primitive:nnn { month } { assign\_int } { 22 }
634 \_ unravel\_tex\_primitive:nnn { year } { assign\_int } { 23 }
635 \_ unravel\_tex\_primitive:nnn { showboxbreadth } { assign\_int } { 24 }
636 \_ unravel\_tex\_primitive:nnn { showboxdepth } { assign\_int } { 25 }
637 \_ unravel\_tex\_primitive:nnn { hbadness } { assign\_int } { 26 }
638 \_ unravel\_tex\_primitive:nnn { vbadness } { assign\_int } { 27 }
639 \_ unravel\_tex\_primitive:nnn { pausing } { assign\_int } { 28 }
640 \_ unravel\_tex\_primitive:nnn { tracingonline } { assign\_int } { 29 }
641 \_ unravel\_tex\_primitive:nnn { tracingmacros } { assign\_int } { 30 }
642 \_ unravel\_tex\_primitive:nnn { tracingstats } { assign\_int } { 31 }
643 \_ unravel\_tex\_primitive:nnn { tracingparagraphs } { assign\_int } { 32 }
644 \_ unravel\_tex\_primitive:nnn { tracingpages } { assign\_int } { 33 }
645 \_ unravel\_tex\_primitive:nnn { tracingoutput } { assign\_int } { 34 }
646 \_ unravel\_tex\_primitive:nnn { tracinglostchars } { assign\_int } { 35 }
647 \_ unravel\_tex\_primitive:nnn { tracingcommands } { assign\_int } { 36 }
648 \_ unravel\_tex\_primitive:nnn { tracingrestores } { assign\_int } { 37 }
649 \_ unravel\_tex\_primitive:nnn { uchyph } { assign\_int } { 38 }
650 \_ unravel\_tex\_primitive:nnn { outputpenalty } { assign\_int } { 39 }
651 \_ unravel\_tex\_primitive:nnn { maxdeadcycles } { assign\_int } { 40 }
652 \_ unravel\_tex\_primitive:nnn { hangafter } { assign\_int } { 41 }
653 \_ unravel\_tex\_primitive:nnn { floatingpenalty } { assign\_int } { 42 }
654 \_ unravel\_tex\_primitive:nnn { globaldefs } { assign\_int } { 43 }
655 \_ unravel\_tex\_primitive:nnn { fam } { assign\_int } { 44 }
656 \_ unravel\_tex\_primitive:nnn { escapechar } { assign\_int } { 45 }
657 \_ unravel\_tex\_primitive:nnn { defaultshyphenchar } { assign\_int } { 46 }
658 \_ unravel\_tex\_primitive:nnn { defaultskewchar } { assign\_int } { 47 }
659 \_ unravel\_tex\_primitive:nnn { endlinedchar } { assign\_int } { 48 }
660 \_ unravel\_tex\_primitive:nnn { newlinedchar } { assign\_int } { 49 }
661 \_ unravel\_tex\_primitive:nnn { language } { assign\_int } { 50 }
662 \_ unravel\_tex\_primitive:nnn { lefthyphenmin } { assign\_int } { 51 }
663 \_ unravel\_tex\_primitive:nnn { righthyphenmin } { assign\_int } { 52 }

```

```

664 \_unravel_tex_primitive:nnn { holdinginserts      } { assign_int } { 53 }
665 \_unravel_tex_primitive:nnn { errorcontegtlines  } { assign_int } { 54 }
666 \_unravel_tex_primitive:nnn { pdfoutput        } { assign_int } { 55 }
667 \_unravel_tex_primitive:nnn { pdfcompresslevel } { assign_int } { 56 }
668 \_unravel_tex_primitive:nnn { pdfdecimaldigits } { assign_int } { 57 }
669 \_unravel_tex_primitive:nnn { pdfmovechars     } { assign_int } { 58 }
670 \_unravel_tex_primitive:nnn { pdfimageresolution } { assign_int } { 59 }
671 \_unravel_tex_primitive:nnn { pdfpkresolution   } { assign_int } { 60 }
672 \_unravel_tex_primitive:nnn { pdfuniquerename  } { assign_int } { 61 }
673 \_unravel_tex_primitive:nnn
674   { pdfoptionalwaysusepdfpagebox  } { assign_int } { 62 }
675 \_unravel_tex_primitive:nnn
676   { pdfoptionpdfinclusionerrorlevel } { assign_int } { 63 }
677 \_unravel_tex_primitive:nnn
678   { pdfoptionpdfminorversion       } { assign_int } { 64 }
679 \_unravel_tex_primitive:nnn { pdfminorversion   } { assign_int } { 64 }
680 \_unravel_tex_primitive:nnn { pdfforcepagebox   } { assign_int } { 65 }
681 \_unravel_tex_primitive:nnn { pdfpagebox        } { assign_int } { 66 }
682 \_unravel_tex_primitive:nnn
683   { pdfinclusionerrorlevel } { assign_int } { 67 }
684 \_unravel_tex_primitive:nnn { pdfgamma          } { assign_int } { 68 }
685 \_unravel_tex_primitive:nnn { pdfimagegamma     } { assign_int } { 69 }
686 \_unravel_tex_primitive:nnn { pdfimagehicolor   } { assign_int } { 70 }
687 \_unravel_tex_primitive:nnn { pdfimageapplygamma } { assign_int } { 71 }
688 \_unravel_tex_primitive:nnn { pdfadjustspacing  } { assign_int } { 72 }
689 \_unravel_tex_primitive:nnn { pdfprotrudechars  } { assign_int } { 73 }
690 \_unravel_tex_primitive:nnn { pdftracingfonts   } { assign_int } { 74 }
691 \_unravel_tex_primitive:nnn { pdfobjcompresslevel } { assign_int } { 75 }
692 \_unravel_tex_primitive:nnn
693   { pdfadjustinterwordglue } { assign_int } { 76 }
694 \_unravel_tex_primitive:nnn { pdfprependkern    } { assign_int } { 77 }
695 \_unravel_tex_primitive:nnn { pdfappendkern     } { assign_int } { 78 }
696 \_unravel_tex_primitive:nnn { pdfgentounicode   } { assign_int } { 79 }
697 \_unravel_tex_primitive:nnn { pdfdraftmode      } { assign_int } { 80 }
698 \_unravel_tex_primitive:nnn { pdfinclusioncopyfonts } { assign_int } { 81 }
699 \_unravel_tex_primitive:nnn { tracingassigns    } { assign_int } { 82 }
700 \_unravel_tex_primitive:nnn { tracinggroups     } { assign_int } { 83 }
701 \_unravel_tex_primitive:nnn { tracingifs        } { assign_int } { 84 }
702 \_unravel_tex_primitive:nnn { tracingcantokens  } { assign_int } { 85 }
703 \_unravel_tex_primitive:nnn { tracingnesting    } { assign_int } { 86 }
704 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
705 \_unravel_tex_primitive:nnn { lastlinefit       } { assign_int } { 88 }
706 \_unravel_tex_primitive:nnn { savingvdiscards   } { assign_int } { 89 }
707 \_unravel_tex_primitive:nnn { savinghyphcodes   } { assign_int } { 90 }
708 \_unravel_tex_primitive:nnn { TeXxETstate      } { assign_int } { 91 }
709 \_unravel_tex_primitive:nnn { parindent         } { assign_dimen } { 0 }
710 \_unravel_tex_primitive:nnn { mathsurround      } { assign_dimen } { 1 }
711 \_unravel_tex_primitive:nnn { lineskiplimit     } { assign_dimen } { 2 }
712 \_unravel_tex_primitive:nnn { hsize             } { assign_dimen } { 3 }
713 \_unravel_tex_primitive:nnn { vsize             } { assign_dimen } { 4 }

```

```

714 \__unravel_tex_primitive:nnn { maxdepth           } { assign_dimen } { 5 }
715 \__unravel_tex_primitive:nnn { splitmaxdepth      } { assign_dimen } { 6 }
716 \__unravel_tex_primitive:nnn { boxmaxdepth      } { assign_dimen } { 7 }
717 \__unravel_tex_primitive:nnn { hfuzz            } { assign_dimen } { 8 }
718 \__unravel_tex_primitive:nnn { vfuzz            } { assign_dimen } { 9 }
719 \__unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
720 \__unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
721 \__unravel_tex_primitive:nnn { scriptspace      } { assign_dimen } { 12 }
722 \__unravel_tex_primitive:nnn { predisplaysize   } { assign_dimen } { 13 }
723 \__unravel_tex_primitive:nnn { displaywidth     } { assign_dimen } { 14 }
724 \__unravel_tex_primitive:nnn { displayindent    } { assign_dimen } { 15 }
725 \__unravel_tex_primitive:nnn { overfullrule     } { assign_dimen } { 16 }
726 \__unravel_tex_primitive:nnn { hangindent       } { assign_dimen } { 17 }
727 \__unravel_tex_primitive:nnn { hoffset          } { assign_dimen } { 18 }
728 \__unravel_tex_primitive:nnn { voffset          } { assign_dimen } { 19 }
729 \__unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
730 \__unravel_tex_primitive:nnn { pdfhorigin       } { assign_dimen } { 21 }
731 \__unravel_tex_primitive:nnn { pdfvorigin       } { assign_dimen } { 22 }
732 \__unravel_tex_primitive:nnn { pdfpagewidth    } { assign_dimen } { 23 }
733 \__unravel_tex_primitive:nnn { pdfpageheight   } { assign_dimen } { 24 }
734 \__unravel_tex_primitive:nnn { pdflinkmargin   } { assign_dimen } { 25 }
735 \__unravel_tex_primitive:nnn { pdfdestmargin   } { assign_dimen } { 26 }
736 \__unravel_tex_primitive:nnn { pdfthreadmargin } { assign_dimen } { 27 }
737 \__unravel_tex_primitive:nnn { pdffirstlineheight } { assign_dimen } { 28 }
738 \__unravel_tex_primitive:nnn { pdfastlinedepth  } { assign_dimen } { 29 }
739 \__unravel_tex_primitive:nnn { pdfeachlineheight } { assign_dimen } { 30 }
740 \__unravel_tex_primitive:nnn { pdfeachlinedepth  } { assign_dimen } { 31 }
741 \__unravel_tex_primitive:nnn { pdfignoreddimen  } { assign_dimen } { 32 }
742 \__unravel_tex_primitive:nnn { pdfpxdimen       } { assign_dimen } { 33 }
743 \__unravel_tex_primitive:nnn { lineskip         } { assign_glue  } { 0 }
744 \__unravel_tex_primitive:nnn { baselineskip     } { assign_glue  } { 1 }
745 \__unravel_tex_primitive:nnn { parskip          } { assign_glue  } { 2 }
746 \__unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue  } { 3 }
747 \__unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue  } { 4 }
748 \__unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
749 \__unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
750 \__unravel_tex_primitive:nnn { leftskip         } { assign_glue  } { 7 }
751 \__unravel_tex_primitive:nnn { rightskip        } { assign_glue  } { 8 }
752 \__unravel_tex_primitive:nnn { topskip          } { assign_glue  } { 9 }
753 \__unravel_tex_primitive:nnn { splittopskip    } { assign_glue  } { 10 }
754 \__unravel_tex_primitive:nnn { tabskip         } { assign_glue  } { 11 }
755 \__unravel_tex_primitive:nnn { spaceskip       } { assign_glue  } { 12 }
756 \__unravel_tex_primitive:nnn { xspaceskip      } { assign_glue  } { 13 }
757 \__unravel_tex_primitive:nnn { parfillskip     } { assign_glue  } { 14 }
758 \__unravel_tex_primitive:nnn { thinmuskip      } { assign_mu_glue } { 15 }
759 \__unravel_tex_primitive:nnn { medmuskip       } { assign_mu_glue } { 16 }
760 \__unravel_tex_primitive:nnn { thickmuskip     } { assign_mu_glue } { 17 }
761 \__unravel_tex_primitive:nnn { fontdimen       } { assign_font_dimen } { 0 }
762 \__unravel_tex_primitive:nnn { hyphenchar      } { assign_font_int  } { 0 }
763 \__unravel_tex_primitive:nnn { skewchar        } { assign_font_int  } { 1 }

```



```

764 \__unravel_tex_primitive:nnn { lrcode           } { assign_font_int } { 2 }
765 \__unravel_tex_primitive:nnn { rrcode           } { assign_font_int } { 3 }
766 \__unravel_tex_primitive:nnn { ercode           } { assign_font_int } { 4 }
767 \__unravel_tex_primitive:nnn { tagcode          } { assign_font_int } { 5 }
768 \__unravel_tex_primitive:nnn { pdfnoligatures   } { assign_font_int } { 6 }
769 \__unravel_tex_primitive:nnn { knbscode          } { assign_font_int } { 7 }
770 \__unravel_tex_primitive:nnn { stbscode          } { assign_font_int } { 8 }
771 \__unravel_tex_primitive:nnn { shbscode          } { assign_font_int } { 9 }
772 \__unravel_tex_primitive:nnn { knbccode          } { assign_font_int } { 10 }
773 \__unravel_tex_primitive:nnn { knaccode          } { assign_font_int } { 11 }
774 \__unravel_tex_primitive:nnn { spacefactor      } { set_aux } { 102 }
775 \__unravel_tex_primitive:nnn { prevdepth         } { set_aux } { 1 }
776 \__unravel_tex_primitive:nnn { prevgraf          } { set_prev_graf } { 0 }
777 \__unravel_tex_primitive:nnn { pagegoal          } { set_page_dimen } { 0 }
778 \__unravel_tex_primitive:nnn { pagetotal         } { set_page_dimen } { 1 }
779 \__unravel_tex_primitive:nnn { pagestretch       } { set_page_dimen } { 2 }
780 \__unravel_tex_primitive:nnn { pagefilstretch    } { set_page_dimen } { 3 }
781 \__unravel_tex_primitive:nnn { pagefillstretch   } { set_page_dimen } { 4 }
782 \__unravel_tex_primitive:nnn { pagefilllstretch  } { set_page_dimen } { 5 }
783 \__unravel_tex_primitive:nnn { pageshrink        } { set_page_dimen } { 6 }
784 \__unravel_tex_primitive:nnn { pagedepth         } { set_page_dimen } { 7 }
785 \__unravel_tex_primitive:nnn { deadcycles         } { set_page_int } { 0 }
786 \__unravel_tex_primitive:nnn { insertpenalties   } { set_page_int } { 1 }
787 \__unravel_tex_primitive:nnn { interactionmode    } { set_page_int } { 2 }
788 \__unravel_tex_primitive:nnn { wd                    } { set_box_dimen } { 1 }
789 \__unravel_tex_primitive:nnn { dp                    } { set_box_dimen } { 2 }
790 \__unravel_tex_primitive:nnn { ht                    } { set_box_dimen } { 3 }
791 \__unravel_tex_primitive:nnn { parshape              } { set_shape } { 0 }
792 \__unravel_tex_primitive:nnn { interlinepenalties    } { set_shape } { 1 }
793 \__unravel_tex_primitive:nnn { clubpenalties         } { set_shape } { 2 }
794 \__unravel_tex_primitive:nnn { widowpenalties        } { set_shape } { 3 }
795 \__unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
796 \__unravel_tex_primitive:nnn { catcode              } { def_code } { 0 }
797 \__unravel_tex_primitive:nnn { lccode                } { def_code } { 256 }
798 \__unravel_tex_primitive:nnn { uccode                } { def_code } { 512 }
799 \__unravel_tex_primitive:nnn { sfcode                } { def_code } { 768 }
800 \__unravel_tex_primitive:nnn { mathcode              } { def_code } { 1024 }
801 \__unravel_tex_primitive:nnn { delcode                } { def_code } { 1591 }
802 \__unravel_tex_primitive:nnn { textfont              } { def_family } { -48 }
803 \__unravel_tex_primitive:nnn { scriptfont            } { def_family } { -32 }
804 \__unravel_tex_primitive:nnn { scriptscriptfont      } { def_family } { -16 }
805 \__unravel_tex_primitive:nnn { nullfont               } { set_font } { 0 }
806 \__unravel_tex_primitive:nnn { font                  } { def_font } { 0 }
807 \__unravel_tex_primitive:nnn { count                  } { register } { 1 000 000 }
808 \__unravel_tex_primitive:nnn { dimen                  } { register } { 2 000 000 }
809 \__unravel_tex_primitive:nnn { skip                   } { register } { 3 000 000 }
810 \__unravel_tex_primitive:nnn { muskip                 } { register } { 4 000 000 }
811 \__unravel_tex_primitive:nnn { advance                } { advance } { 0 }
812 \__unravel_tex_primitive:nnn { multiply               } { multiply } { 0 }
813 \__unravel_tex_primitive:nnn { divide                 } { divide } { 0 }

```

```

814 \__unravel_tex_primitive:nnn { long           } { prefix } { 1 }
815 \__unravel_tex_primitive:nnn { outer         } { prefix } { 2 }
816 \__unravel_tex_primitive:nnn { global       } { prefix } { 4 }
817 \__unravel_tex_primitive:nnn { protected  } { prefix } { 8 }
818 \__unravel_tex_primitive:nnn { let         } { let } { 0 }
819 \__unravel_tex_primitive:nnn { futurelet  } { let } { 1 }
820 \__unravel_tex_primitive:nnn { chardef     } { shorthand_def } { 0 }
821 \__unravel_tex_primitive:nnn { mathchardef } { shorthand_def } { 1 }
822 \__unravel_tex_primitive:nnn { countdef    } { shorthand_def } { 2 }
823 \__unravel_tex_primitive:nnn { dimendef    } { shorthand_def } { 3 }
824 \__unravel_tex_primitive:nnn { skipdef     } { shorthand_def } { 4 }
825 \__unravel_tex_primitive:nnn { muskipdef   } { shorthand_def } { 5 }
826 \__unravel_tex_primitive:nnn { toksdef     } { shorthand_def } { 6 }
827 \__unravel_tex_primitive:nnn { read        } { read_to_cs } { 0 }
828 \__unravel_tex_primitive:nnn { readline    } { read_to_cs } { 1 }
829 \__unravel_tex_primitive:nnn { def         } { def } { 0 }
830 \__unravel_tex_primitive:nnn { gdef        } { def } { 1 }
831 \__unravel_tex_primitive:nnn { edef       } { def } { 2 }
832 \__unravel_tex_primitive:nnn { xdef       } { def } { 3 }
833 \__unravel_tex_primitive:nnn { setbox     } { set_box } { 0 }
834 \__unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
835 \__unravel_tex_primitive:nnn { patterns   } { hyph_data } { 1 }
836 \__unravel_tex_primitive:nnn { batchmode  } { set_interaction } { 0 }
837 \__unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
838 \__unravel_tex_primitive:nnn { scrollmode  } { set_interaction } { 2 }
839 \__unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
840 \__unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
841 \__unravel_tex_primitive:nnn { pdfcopyfont } { pdf_copy_font } { 0 }
842 \__unravel_tex_primitive:nnn { undefined  } { undefined_cs } { 0 }
843 \__unravel_tex_primitive:nnn { ndefined   } { undefined_cs } { 0 }
844 \__unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
845 \__unravel_tex_primitive:nnn { unless     } { expand_after } { 1 }
846 \__unravel_tex_primitive:nnn { pdfprimitive } { no_expand } { 1 }
847 \__unravel_tex_primitive:nnn { noexpand   } { no_expand } { 0 }
848 \__unravel_tex_primitive:nnn { input      } { input } { 0 }
849 \__unravel_tex_primitive:nnn { endinput   } { input } { 1 }
850 \__unravel_tex_primitive:nnn { scantokens } { input } { 2 }
851 \__unravel_tex_primitive:nnn { if         } { if_test } { 0 }
852 \__unravel_tex_primitive:nnn { ifcat      } { if_test } { 1 }
853 \__unravel_tex_primitive:nnn { ifnum     } { if_test } { 2 }
854 \__unravel_tex_primitive:nnn { ifdim     } { if_test } { 3 }
855 \__unravel_tex_primitive:nnn { ifodd     } { if_test } { 4 }
856 \__unravel_tex_primitive:nnn { ifvmode   } { if_test } { 5 }
857 \__unravel_tex_primitive:nnn { ifhmode   } { if_test } { 6 }
858 \__unravel_tex_primitive:nnn { ifmmode   } { if_test } { 7 }
859 \__unravel_tex_primitive:nnn { ifinner   } { if_test } { 8 }
860 \__unravel_tex_primitive:nnn { ifvoid    } { if_test } { 9 }
861 \__unravel_tex_primitive:nnn { ifhbox    } { if_test } { 10 }
862 \__unravel_tex_primitive:nnn { ifvbox    } { if_test } { 11 }
863 \__unravel_tex_primitive:nnn { ifx       } { if_test } { 12 }

```

```

864 \__unravel_tex_primitive:nnn { ifeof           } { if_test } { 13 }
865 \__unravel_tex_primitive:nnn { iftrue          } { if_test } { 14 }
866 \__unravel_tex_primitive:nnn { iffalse        } { if_test } { 15 }
867 \__unravel_tex_primitive:nnn { ifcase         } { if_test } { 16 }
868 \__unravel_tex_primitive:nnn { ifdefined     } { if_test } { 17 }
869 \__unravel_tex_primitive:nnn { ifcsname       } { if_test } { 18 }
870 \__unravel_tex_primitive:nnn { iffontchar    } { if_test } { 19 }
871 \__unravel_tex_primitive:nnn { ifincsname    } { if_test } { 20 }
872 \__unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
873 \__unravel_tex_primitive:nnn { ifpdfabsnum   } { if_test } { 22 }
874 \__unravel_tex_primitive:nnn { ifpdfabsdim   } { if_test } { 23 }
875 \__unravel_tex_primitive:nnn { fi            } { fi_or_else } { 2 }
876 \__unravel_tex_primitive:nnn { else           } { fi_or_else } { 3 }
877 \__unravel_tex_primitive:nnn { or             } { fi_or_else } { 4 }
878 \__unravel_tex_primitive:nnn { csname         } { cs_name   } { 0 }
879 \__unravel_tex_primitive:nnn { number        } { convert   } { 0 }
880 \__unravel_tex_primitive:nnn { romannumeral   } { convert   } { 1 }
881 \__unravel_tex_primitive:nnn { string         } { convert   } { 2 }
882 \__unravel_tex_primitive:nnn { meaning       } { convert   } { 3 }
883 \__unravel_tex_primitive:nnn { fontname       } { convert   } { 4 }
884 \__unravel_tex_primitive:nnn { eTeXrevision  } { convert   } { 5 }
885 \__unravel_tex_primitive:nnn { pdftexrevision } { convert   } { 6 }
886 \__unravel_tex_primitive:nnn { pdftexbanner  } { convert   } { 7 }
887 \__unravel_tex_primitive:nnn { pdffontname   } { convert   } { 8 }
888 \__unravel_tex_primitive:nnn { pdffontobjnum } { convert   } { 9 }
889 \__unravel_tex_primitive:nnn { pdffontsize   } { convert   } { 10 }
890 \__unravel_tex_primitive:nnn { pdfpageref    } { convert   } { 11 }
891 \__unravel_tex_primitive:nnn { pdfxformname  } { convert   } { 12 }
892 \__unravel_tex_primitive:nnn { pdfescapestring } { convert   } { 13 }
893 \__unravel_tex_primitive:nnn { pdfescapename } { convert   } { 14 }
894 \__unravel_tex_primitive:nnn { leftmarginkern } { convert   } { 15 }
895 \__unravel_tex_primitive:nnn { rightmarginkern } { convert   } { 16 }
896 \__unravel_tex_primitive:nnn { pdfstrcmp    } { convert   } { 17 }
897 \__unravel_tex_primitive:nnn { pdfcolorstackinit } { convert   } { 18 }
898 \__unravel_tex_primitive:nnn { pdfescapehex  } { convert   } { 19 }
899 \__unravel_tex_primitive:nnn { pdfunescapehex } { convert   } { 20 }
900 \__unravel_tex_primitive:nnn { pdfcreationdate } { convert   } { 21 }
901 \__unravel_tex_primitive:nnn { pdffilemoddate } { convert   } { 22 }
902 \__unravel_tex_primitive:nnn { pdffilesize  } { convert   } { 23 }
903 \__unravel_tex_primitive:nnn { pdfmdfivesum } { convert   } { 24 }
904 \__unravel_tex_primitive:nnn { pdffiledump  } { convert   } { 25 }
905 \__unravel_tex_primitive:nnn { pdfmatch     } { convert   } { 26 }
906 \__unravel_tex_primitive:nnn { pdflastmatch  } { convert   } { 27 }
907 \__unravel_tex_primitive:nnn { pdfuniformdeviate } { convert   } { 28 }
908 \__unravel_tex_primitive:nnn { pdfnormaldeviate } { convert   } { 29 }
909 \__unravel_tex_primitive:nnn { pdfinsertht  } { convert   } { 30 }
910 \__unravel_tex_primitive:nnn { pdfximagebbox } { convert   } { 31 }
911 \__unravel_tex_primitive:nnn { jobname      } { convert   } { 32 }
912 \__unravel_tex_primitive:nnn { the          } { the } { 0 }
913 \__unravel_tex_primitive:nnn { unexpanded   } { the } { 1 }

```

```

914 \__unravel_tex_primitive:nnn { detokenize      } { the } { 5 }
915 \__unravel_tex_primitive:nnn { topmark        } { top_bot_mark } { 0 }
916 \__unravel_tex_primitive:nnn { firstmark     } { top_bot_mark } { 1 }
917 \__unravel_tex_primitive:nnn { botmark       } { top_bot_mark } { 2 }
918 \__unravel_tex_primitive:nnn { splitfirstmark } { top_bot_mark } { 3 }
919 \__unravel_tex_primitive:nnn { splitbotmark  } { top_bot_mark } { 4 }
920 \__unravel_tex_primitive:nnn { topmarks      } { top_bot_mark } { 5 }
921 \__unravel_tex_primitive:nnn { firstmarks    } { top_bot_mark } { 6 }
922 \__unravel_tex_primitive:nnn { botmarks      } { top_bot_mark } { 7 }
923 \__unravel_tex_primitive:nnn { splitfirstmarks } { top_bot_mark } { 8 }
924 \__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.

```

925 \cs_new_protected_nopar:Npn \__unravel_get_next:
926   {
927   \__unravel_input_if_empty:TF
928     { \__unravel_exit:w }
929     {
930       \__unravel_input_gpop:N \l__unravel_head_gtl
931       \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
932       \gtl_if_tl:NTF \l__unravel_head_gtl
933         {
934           \tl_set:Nx \l__unravel_head_tl
935             { \gtl_head:N \l__unravel_head_gtl }
936         }
937         { \tl_clear:N \l__unravel_head_tl }
938       }
939   }
940 \cs_new_protected_nopar:Npn \__unravel_get_next_aux:w
941   { \cs_set_eq:NN \l__unravel_head_token }

```

*(End definition for `\__unravel_get_next`:. This function is documented on page ??.)*

`\__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`.

```

942 \cs_new_protected_nopar:Npn \__unravel_get_token:
943   {
944     \__unravel_get_next:
945     \__unravel_set_cmd:
946   }

```

(End definition for `\__unravel_get_token:`.)

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

```

947 \cs_new_protected_nopar:Npn \__unravel_set_cmd:
948   {
949     \__unravel_set_cmd_aux_meaning:
950     \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
951     { }
952     {
953       \__unravel_token_if_expandable:NTF \l__unravel_head_token
954       {
955         \token_if_macro:NTF \l__unravel_head_token
956         { \__unravel_set_cmd_aux_macro: }
957         { \__unravel_set_cmd_aux_unknown: }
958       }
959       {
960         \token_if_cs:NTF \l__unravel_head_token
961         { \__unravel_set_cmd_aux_cs: }
962         { \__unravel_set_cmd_aux_char: }
963       }
964     }
965   }

```

(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.

```

966 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_meaning:
967   {
968     \tl_set:Nx \l__unravel_head_meaning_tl
969     {
970       \exp_after:wN \__unravel_strip_escape:w
971       \token_to_meaning:N \l__unravel_head_token

```

```

972     \tl_to_str:n { : }
973   }
974   \tl_set:Nx \l__unravel_head_meaning_tl
975     {
976     \exp_after:wN \__unravel_set_cmd_aux_meaning:w
977     \l__unravel_head_meaning_tl \q_stop
978   }
979 }
980 \use:x
981 {
982   \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
983     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
984 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_meaning:. This function is documented on page ??.)

\\_\_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
985 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
986   {
987   \cs_if_exist:cTF { c__unravel_tex_#1_tl }
988     {
989     \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
990       { c__unravel_tex_#1_tl }
991     #2
992   }
993 }
994 \cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
995 \cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
996   {
997   \int_set:Nn \l__unravel_head_cmd_int {#1}
998   \int_set:Nn \l__unravel_head_char_int {#2}
999 }

```

(End definition for \\_\_unravel\_set\_cmd\_aux\_primitive:nTF and \\_\_unravel\_set\_cmd\_aux\_primitive:oTF. These functions are documented on page ??.)

\\_\_unravel\_set\_cmd\_aux\_macro: The token is a macro. For now we do not test if the macro is long/outer.

```

1000 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_macro:
1001   {
1002   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1003   \int_zero:N \l__unravel_head_char_int
1004   }

```

(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.

```

1005 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_unknown:
1006   {
1007   \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
1008     \c__unravel_tex_relax_tl
1009   \msg_error:nxx { unravel } { unknown-primitive }

```

```

1010     { \l__unravel_head_meaning_tl }
1011   }
(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.

```

1012 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_cs:
1013   {
1014     \tl_if_in:NoTF \l__unravel_head_meaning_tl
1015       { \tl_to_str:n { elect-font } }
1016     {
1017       \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1018         \c__unravel_tex_nullfont_tl
1019     }
1020     { \__unravel_set_cmd_aux_numeric: }
1021   }
(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.

```

1022 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_numeric:
1023   {
1024     \tl_set:Nx \l__unravel_tmpa_tl
1025     {
1026       \exp_after:wN \__unravel_set_cmd_aux_numeric:N
1027         \l__unravel_head_meaning_tl + 0
1028     }
1029     \exp_after:wN \__unravel_set_cmd_aux_numeric:w
1030       \l__unravel_tmpa_tl \q_stop
1031   }
1032 \cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
1033   {
1034     \if_int_compare:w '#1 < 'A \exp_stop_f:
1035       \exp_not:N \q_mark
1036       \exp_after:wN \use_i:nn
1037     \fi:
1038     #1 \__unravel_set_cmd_aux_numeric:N
1039   }
1040 \cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop

```

```

1041 {
1042   \str_case:nnF {#1}
1043   {
1044     { char }      { \__unravel_set_cmd_aux_given:n { char_given } }
1045     { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
1046   }
1047   {
1048     \__unravel_set_cmd_aux_primitive:nTF {#1}
1049     { }
1050     { \__unravel_set_cmd_aux_unknown: }
1051     \int_add:Nn \l__unravel_head_char_int { 100 000 }
1052   }
1053   \int_add:Nn \l__unravel_head_char_int {#2}
1054 }
1055 \cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
1056 {
1057   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
1058   \int_zero:N \l__unravel_head_char_int
1059 }

```

*(End definition for \\_\_unravel\_set\_cmd\_aux\_numeric:, \\_\_unravel\_set\_cmd\_aux\_numeric:w, and \\_\_unravel\_set\_cmd\_aux\_... These functions are documented on page ??.)*

\\_\_unravel\_set\_cmd\_aux\_char:  
 \\_\_unravel\_set\_cmd\_aux\_char:w

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 ‘.

```

1060 \cs_new_protected_nopar:Npn \__unravel_set_cmd_aux_char:
1061 {
1062   \tl_set:Nx \l__unravel_head_meaning_tl
1063   { \token_to_meaning:N \l__unravel_head_token }
1064   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1065   { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1066   \exp_after:wN \__unravel_set_cmd_aux_char:w
1067   \l__unravel_head_meaning_tl \q_stop
1068   \exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1069   { ‘ \__unravel_token_to_char:N \l__unravel_head_token }
1070 }
1071 \cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1072 {
1073   \int_set:Nn \l__unravel_head_cmd_int
1074   { \__unravel_tex_use:n { #1_char } }
1075 }

```

*(End definition for \\_\_unravel\_set\_cmd\_aux\_char:. This function is documented on page ??.)*



## 2.5 Manipulating the input

### 2.5.1 Elementary operations

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

```
1076 \cs_new_nopar:Npn \__unravel_input_to_str:
1077   {
1078     \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1079     \__unravel_input_to_str_aux:n
1080   }
1081 \cs_new:Npn \__unravel_input_to_str_aux:n #1
1082   { \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.

```
1083 \cs_new_protected:Npn \__unravel_input_if_empty:TF
1084   {
1085     \int_compare:nNnTF \g__unravel_input_int = \c_zero
1086     { \use_i:nn }
1087     {
1088       \gtl_if_empty:cTF
1089       { g__unravel_input_ \int_use:N \g__unravel_input_int_gtl }
1090       {
1091         \int_gdecr:N \g__unravel_input_int
1092         \__unravel_input_if_empty:TF
1093       }
1094       {
1095         \__unravel_input_split:
1096         \use_ii:nn
1097       }
1098     }
1099   }
```

*(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

```
1100 \cs_new_protected_nopar:Npn \__unravel_input_split:
1101   {
1102     \int_compare:nNnT \g__unravel_input_int = \c_one
1103     {
1104       \exp_args:Nc \__unravel_input_split_aux:N
1105       { g__unravel_input_1_gtl }
1106     }
1107   }
1108 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1109   {
1110     \gtl_if_tl:NT #1
```

```

1111     {
1112         \gtl_if_head_is_N_type:NT #1
1113         {
1114             \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1115             \exp_last_unbraced:Nx \__unravel_input_split_auxii:N
1116             { \tl_head:N \l__unravel_input_tmpa_tl }
1117         }
1118     }
1119 }
1120 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1121 {
1122     \token_if_parameter:NF #1
1123     {
1124         \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1125         { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1126     \group_begin:
1127         \cs_set:Npn \__unravel_input_split_auxiii:w
1128             ##1 \__unravel_input_split_end: { + 1 }
1129         \int_gset:Nn \g__unravel_input_int
1130             { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1131     \group_end:
1132     \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1133     \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1134     }
1135 }
1136 \cs_new_nopar:Npn \__unravel_input_split_end: { }
1137 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1138     #1 \__unravel_input_split_end:
1139     {
1140     \gtl_gset:cn
1141     { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1142     \int_gdecr:N \g__unravel_input_tmpa_int
1143     }

```

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

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

```

1144 \cs_new_protected_nopar:Npn \__unravel_input_gset:n
1145     {
1146     \int_gset_eq:NN \g__unravel_input_int \c_one
1147     \gtl_gset:cn { g__unravel_input_1_gtl }
1148     }

```

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

\\_\_unravel\_input\_get:N

```

1149 \cs_new_protected:Npn \__unravel_input_get:N #1
1150     {
1151     \__unravel_input_if_empty:TF
1152     { \gtl_set:Nn #1 { \q_no_value } }
1153     {

```

```

1154     \gtl_get_left:cN
1155     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1156   }
1157 }

```

*(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.

```

1158 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1159 {
1160   \__unravel_input_if_empty:TF
1161   { \gtl_set:Nn #1 { \q_no_value } }
1162   {
1163     \gtl_gpop_left:cN
1164     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1165   }
1166 }

```

*(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.

```

1167 \cs_new_protected_nopar:Npn \__unravel_input_merge:
1168 {
1169   \int_gdecr:N \g__unravel_input_int
1170   \gtl_gconcat:ccc
1171   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1172   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1173   { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1174   \gtl_gclear:c
1175   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1176 }

```

*(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**

```

1177 \prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
1178 {
1179   \int_compare:nNnTF \g__unravel_input_int = \c_zero
1180   { \prg_return_false: }
1181   {
1182     \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1183     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1184   }
1185 }
1186 \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2

```

```

1187 {
1188   \gtl_gpop_left_item:NNTF #1#2
1189   { \prg_return_true: }
1190   {
1191     \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero
1192     { \prg_return_false: }
1193     {
1194       \int_compare:nNnTF \g__unravel_input_int = \c_one
1195       { \prg_return_false: }
1196       {
1197         \__unravel_input_merge:
1198         \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1199         {
1200           g__unravel_input_
1201           \int_use:N \g__unravel_input_int_gtl
1202         }
1203         #2
1204       }
1205     }
1206   }
1207 }

```

(End definition for \\_\_unravel\_input\_gpop\_item:N. This function is documented on page ??.)

\\_\_unravel\_input\_gpop\_tl:N

```

1208 \cs_new_protected:Npn \__unravel_input_gpop_tl:N #1
1209 { \tl_clear:N #1 \__unravel_input_gpop_tl_aux:N #1 }
1210 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:N #1
1211 {
1212   \int_compare:nNnF \g__unravel_input_int = \c_zero
1213   {
1214     \exp_args:Nc \__unravel_input_gpop_tl_aux:NN
1215     { g__unravel_input_ \int_use:N \g__unravel_input_int_gtl } #1
1216   }
1217 }
1218 \cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
1219 {
1220   \gtl_if_tl:NNTF #1
1221   {
1222     \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1223     \gtl_gclear:N #1
1224     \int_gdecr:N \g__unravel_input_int
1225     \__unravel_input_gpop_tl_aux:N #2
1226   }
1227   {
1228     \int_compare:nNnTF \g__unravel_input_int > \c_one
1229     { \int_compare:nNnTF { \gtl_extra_end:N #1 } > \c_zero }
1230     { \use_i:nn }
1231     {
1232       \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1233       \gtl_gpop_left_tl:N #1

```

```

1234     }
1235     {
1236     \__unravel_input_merge:
1237     \__unravel_input_gpop_tl_aux:N #2
1238     }
1239     }
1240 }

```

(End definition for \\_\_unravel\_input\_gpop\_tl:N.)

\\_\_unravel\_back\_input:n Insert a token list back into the input.

```

\__unravel_back_input:x 1241 \cs_new_protected_nopar:Npn \__unravel_back_input:n
1242 {
1243 \int_gincr:N \g__unravel_input_int
1244 \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1245 }
1246 \cs_generate_variant:Nn \__unravel_back_input:n { x , V , o }

```

(End definition for \\_\_unravel\_back\_input:n and \\_\_unravel\_back\_input:x.)

\\_\_unravel\_back\_input\_gtl:N Insert a generalized token list back into the input.

```

1247 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1248 {
1249 \gtl_if_tl:NTF #1
1250 { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1251 {
1252 \gtl_gconcat:cNc
1253 { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1254 #1
1255 { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1256 }
1257 }

```

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

\\_\_unravel\_back\_input: Insert the last token read back into the input stream.

```

1258 \cs_new_protected_nopar:Npn \__unravel_back_input:
1259 { \__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.

```

1260 \cs_new_protected_nopar:Npn \__unravel_back_input_tl_o:
1261 {
1262 \tl_set:Nx \l__unravel_tmpa_tl
1263 { \exp_args:NW \exp_not:o \l__unravel_head_tl }
1264 \__unravel_back_input:V \l__unravel_tmpa_tl
1265 \__unravel_print_done:x
1266 { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1267 }

```

(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.

```
1268 \cs_new_protected_nopar:Npn \__unravel_insert_relax:
1269   {
1270     \__unravel_back_input:
1271     \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1272     \__unravel_back_input:
1273     \__unravel_print_action:
1274   }
(End definition for \__unravel_insert_relax:.)
```

`\__unravel_insert_group_begin_error:`

```
1275 \cs_new_protected_nopar:Npn \__unravel_insert_group_begin_error:
1276   {
1277     \msg_error:nn { unravel } { missing-lbrace }
1278     \__unravel_back_input:
1279     \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1280     \__unravel_back_input:
1281     \__unravel_print_action:
1282   }
(End definition for \__unravel_insert_group_begin_error:.)
```

`\__unravel_insert_dollar_error:`

```
1283 \cs_new_protected_nopar:Npn \__unravel_insert_dollar_error:
1284   {
1285     \__unravel_back_input:
1286     \__unravel_back_input:n { $ } % $
1287     \msg_error:nn { unravel } { missing-dollar }
1288     \__unravel_print_action:
1289   }
(End definition for \__unravel_insert_dollar_error:.)
```

## 2.5.3 Macro calls

```
\__unravel_macro_prefix:N
\__unravel_macro_parameter:N
  \__unravel_macro_replacement:N
1290 \group_begin:
1291   \char_set_lccode:nn { ' . } { ': }
1292   \tex_lowercase:D
1293   {
1294     \cs_new:Npn \__unravel_macro_split_do:NN #1
1295       {
1296         \exp_after:wN \__unravel_macro_split_do:wN
1297         \token_to_meaning:N #1 \q_mark { } . -> \q_mark \use_none:nmmm
1298         \q_stop
1299       }
1300     \cs_new:Npn \__unravel_macro_split_do:wN
```

```

1301         #1 . #2 -> #3 \q_mark #4 #5 \q_stop #6
1302         { #4 #6 {#1} {#2} {#3} }
1303     }
1304 \group_end:
1305 \cs_new:Npn \__unravel_macro_prefix:N #1
1306   { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1307 \cs_new:Npn \__unravel_macro_parameter:N #1
1308   { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1309 \cs_new:Npn \__unravel_macro_replacement:N #1
1310   { \__unravel_macro_split_do:NN #1 \use_iii:nnn }
(End definition for \__unravel_macro_prefix:N, \__unravel_macro_parameter:N, and \__unravel_macro_replacement:N.)

```

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

```

1311 \cs_new_protected_nopar:Npn \__unravel_macro_call:
1312   {
1313     \bool_if:NTF \g__unravel_speedup_macros_bool
1314       {
1315         \tl_set:Nx \l__unravel_tmpa_tl
1316           { ^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1317         \tl_if_in:NVTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1318           { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1319       }
1320     { \__unravel_macro_call_safe: }
1321     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1322     \__unravel_print_done:x { \g__unravel_action_text_str }
1323   }
1324 \cs_new_protected_nopar:Npn \__unravel_macro_call_safe:
1325   {
1326     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1327     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1328   }
1329 \cs_new_protected_nopar:Npn \__unravel_macro_call_quick:
1330   {
1331     \exp_after:wN \__unravel_macro_call_quick_loop:NN \l__unravel_tmpa_tl
1332     { ? \use_none_delimit_by_q_stop:w } \q_stop
1333   }
1334 \cs_new_protected_nopar:Npn \__unravel_macro_call_quick_loop:NN #1#2
1335   {
1336     \use_none:n #2
1337     \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1338     { \msg_error:nn { unravel } { runaway-macro-parameter } }
1339     \tl_put_right:Nx \l__unravel_head_tl
1340       { { \exp_not:V \l__unravel_tmpa_tl } }
1341     \__unravel_macro_call_quick_loop:NN
1342   }

```

(End definition for `\__unravel_macro_call:`. This function is documented on page ??.)

## 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).

```
1343 \cs_new_protected_nopar:Npn \__unravel_expand:
1344   {
1345     \__unravel_set_action_text:
1346     \bool_if:NT \l__unravel_debug_bool
1347       {
1348         \__unravel_set_cmd:
1349         \iow_term:x { Exp:~\int_use:N \l__unravel_head_cmd_int }
1350       }
1351     \token_if_macro:NTF \l__unravel_head_token
1352     { \__unravel_macro_call: }
1353     { \__unravel_expand_nonmacro: }
1354   }
```

*(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.

```
1355 \cs_new_protected_nopar:Npn \__unravel_expand_nonmacro:
1356   {
1357     \__unravel_set_cmd_aux_meaning:
1358     \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1359     {
1360       \cs_if_exist_use:cF
1361       { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1362       { \msg_error:nxx { unravel } { internal } { expandable } }
1363     }
1364     {
1365       \msg_error:nxx { unravel } { unknown-primitive }
1366       { \l__unravel_head_meaning_tl }
1367       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
1368       \__unravel_print_action:
1369     }
1370   }
```

*(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.

```
1371 \cs_new_protected_nopar:Npn \__unravel_get_x_next:
1372   {
1373     \__unravel_get_next:
1374     \__unravel_token_if_expandable:NT \l__unravel_head_token
1375     {
1376       \__unravel_expand:
```



```

1377         \__unravel_get_x_next:
1378     }
1379 }
(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.

```

1380 \cs_new_protected_nopar:Npn \__unravel_get_x_or_protected:
1381 {
1382     \__unravel_get_next:
1383     \__unravel_token_if_protected:NF \l__unravel_head_token
1384     {
1385         \__unravel_expand:
1386         \__unravel_get_x_or_protected:
1387     }
1388 }
(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.

```

1389 \cs_new_protected_nopar:Npn \__unravel_get_x_non_blank:
1390 {
1391     \__unravel_get_x_next:
1392     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1393     { \__unravel_get_x_non_blank: }
1394 }
(End definition for \__unravel_get_x_non_blank:.)

```

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

```

1395 \cs_new_protected_nopar:Npn \__unravel_get_x_non_relax:
1396 {
1397     \__unravel_get_x_next:
1398     \token_if_eq_meaning:NNT \l__unravel_head_token \scan_stop:
1399     { \__unravel_get_x_non_relax: }
1400     {
1401         \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1402         { \__unravel_get_x_non_relax: }
1403     }
1404 }
(End definition for \__unravel_get_x_non_relax:.)

```

`\__unravel_skip_optional_space:`

```

1405 \cs_new_protected_nopar:Npn \__unravel_skip_optional_space:
1406 {
1407     \__unravel_get_x_next:
1408     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1409     { \__unravel_back_input: }
1410 }

```

(End definition for `\__unravel_skip_optional_space:`.)

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

```
1411 \cs_new_protected_nopar:Npn \__unravel_scan_optional_equals:
1412   {
1413     \__unravel_get_x_non_blank:
1414     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1415       { \__unravel_prev_input:n { = } }
1416       {
1417         \__unravel_prev_input_silent:n { = }
1418         \__unravel_back_input:
1419       }
1420   }
```

(End definition for `\__unravel_scan_optional_equals:`.)

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

```
1421 \cs_new_protected_nopar:Npn \__unravel_scan_left_brace:
1422   {
1423     \__unravel_get_x_non_relax:
1424     \token_if_eq_catcode:NMF \l__unravel_head_token \c_group_begin_token
1425       { \__unravel_insert_group_begin_error: }
1426   }
```

(End definition for `\__unravel_scan_left_brace:`.)

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

```

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

1427 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1428   { \__unravel_scan_keyword:nTF {#1} { } { } }
1429 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1430   { T , F , TF }
1431   {
1432     \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl
1433     \__unravel_scan_keyword_loop:NNN \c_true_bool
1434     #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1435   }
1436 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1437   {
1438     \quark_if_recursion_tail_stop_do:nn {#2}
1439     { \__unravel_scan_keyword_true: }
1440     \quark_if_recursion_tail_stop_do:nn {#3}
1441     { \msg_error:nnx { unravel } { internal } { odd-keyword-length } }
1442     \__unravel_get_x_next:
1443     \__unravel_scan_keyword_test:NNTF #2#3
1444     {
1445       \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1446       \__unravel_scan_keyword_loop:NNN \c_false_bool
1447     }
1448     {
1449       \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
1450       { \__unravel_scan_keyword_false:w }
1451       \bool_if:NF #1
1452       { \__unravel_scan_keyword_false:w }
1453       \__unravel_scan_keyword_loop:NNN #1#2#3
1454     }
1455   }
1456 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
1457   { TF }
1458   {
1459     \__unravel_gtl_if_head_is_definable:NMF \l__unravel_head_gtl
1460     { \prg_return_false: }
1461     {
1462       \str_if_eq_x:nnTF
1463       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
1464       { \prg_return_true: }
1465     }
1466     \str_if_eq_x:nnTF
1467     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
1468     { \prg_return_true: }
1469     { \prg_return_false: }
1470   }
1471 }
1472 }

```

```

1473 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_true:
1474 {
1475   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
1476   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
1477   \prg_return_true:
1478 }
1479 \cs_new_protected_nopar:Npn \__unravel_scan_keyword_false:w
1480 #1 \q_recursion_stop
1481 {
1482   \__unravel_back_input:
1483   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
1484   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
1485   \prg_return_false:
1486 }

```

(End definition for \\_\_unravel\_scan\_keyword:n. This function is documented on page ??.)

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

```

1487 \cs_new_protected_nopar:Npn \__unravel_scan_font_ident:
1488 {
1489   \__unravel_get_x_non_blank:
1490   \__unravel_set_cmd:
1491   \int_case:nnF \l__unravel_head_cmd_int
1492   {
1493     { \__unravel_tex_use:n { def_font } }
1494     { \__unravel_prev_input:V \l__unravel_head_tl }
1495     { \__unravel_tex_use:n { letterspace_font } }
1496     { \__unravel_prev_input:V \l__unravel_head_tl }
1497     { \__unravel_tex_use:n { pdf_copy_font } }
1498     { \__unravel_prev_input:V \l__unravel_head_tl }
1499     { \__unravel_tex_use:n { set_font } }
1500     { \__unravel_prev_input:V \l__unravel_head_tl }
1501     { \__unravel_tex_use:n { def_family } }
1502     {
1503       \__unravel_prev_input:V \l__unravel_head_tl
1504       \__unravel_scan_int:
1505     }
1506   }
1507   {
1508     \msg_error:nn { unravel } { missing-font-id }
1509     \__unravel_prev_input:n { \tex_nullfont:D }
1510     \__unravel_back_error:
1511   }
1512 }

```

(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).

```

1513 \cs_new_protected_nopar:Npn \__unravel_scan_font_int:
1514 {
1515   \int_case:nnF \l__unravel_head_char_int

```

```

1516     {
1517         { 0 } { \__unravel_scan_font_ident: }
1518         { 1 } { \__unravel_scan_font_ident: }
1519         { 6 } { \__unravel_scan_font_ident: }
1520     }
1521     { \__unravel_scan_font_ident: \__unravel_scan_int: }
1522 }

```

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

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

```

1523 \cs_new_protected_nopar:Npn \__unravel_scan_font_dimen:
1524 {
1525     \__unravel_scan_int:
1526     \__unravel_scan_font_ident:
1527 }

```

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

\\_\_unravel\_scan\_something\_internal:n Receives an (explicit) “level” argument:

\\_\_unravel\_scan\_something\_aux:nwn

- int\_val=0 for integer values;
- dimen\_val=1 for dimension values;
- glue\_val=2 for glue specifications;
- mu\_val=3 for math glue specifications;
- ident\_val=4 for font identifiers (this never happens);
- tok\_val=5 for token lists.

Scans something internal, and places its value, converted to the given level, to the right of the last item of \g\_\_unravel\_prev\_input\_seq, then sets \g\_\_unravel\_val\_level\_int to the found level (level before conversion, so this may be higher than requested). Get in one go the information about what level is produced by the given token once it has received all its operands (head of \l\_\_unravel\_tmpa\_tl), and about what to do to find those operands (tail of \l\_\_unravel\_tmpa\_tl). If the first token is not between min\_internal=68 and max\_internal=89, this step claims a level of 8. If the level that will be produced is 4 or 5, but the argument #1 is not, or if the level is 8 (exercise: check that the conditional indeed checks for this case, given that  $\varepsilon$ -TeX rounds “to nearest, ties away from zero”), then complain. Otherwise, fetch arguments if there are any: the scanning is performed after placing the current token in a new level of prev\_input and telling the user about it. Once done with this step, \l\_\_unravel\_head\_tl contains the tokens found. Convert them to the wanted level with \\_\_unravel\_thing\_use:nN (at this stage, TeX may complain about a missing number or incompatible glue units), and place the result in prev\_input. Finally, tell the user the tokens that have been found and their value (and, if there was a single token, its meaning).

```

1528 \cs_new_protected:Npn \__unravel_scan_something_internal:n #1
1529 {

```

```

1530 \__unravel_set_cmd:
1531 \__unravel_set_action_text:
1532 \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
1533 \exp_after:wN \__unravel_scan_something_aux:nwn
1534   \l__unravel_tmpa_tl \q_stop {#1}
1535 }
1536 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
1537 {
1538   \int_compare:nNnTF
1539     { ( #1 + \c_two ) / \c_four } > { ( #3 + \c_two ) / \c_four }
1540     { \__unravel_back_input: }
1541     {
1542       \tl_if_empty:nF {#2}
1543       {
1544         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
1545         \__unravel_print_action:
1546         #2
1547         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1548       }
1549     }
1550   \tl_set:Nx \l__unravel_tmpa_tl
1551     { \__unravel_thing_use:nnN {#1} {#3} \l__unravel_head_tl }
1552   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
1553   \__unravel_set_action_text:
1554   \__unravel_set_action_text:x
1555     { \g__unravel_action_text_str = ~ \tl_to_str:N \l__unravel_tmpa_tl }
1556   \int_compare:nNnF {#3} > { 3 } { \__unravel_print_action: }
1557   \int_gset:Nn \g__unravel_val_level_int {#1}
1558 }

```

(End definition for \\_\_unravel\_scan\_something\_internal:n. This function is documented on page ??.)

\\_\_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 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.

```

1559 \cs_new_nopar:Npn \__unravel_thing_case:
1560 {
1561   \int_case:nnF \l__unravel_head_cmd_int
1562   {
1563     { 68 } { 0 } % char_given
1564     { 69 } { 0 } % math_given
1565     { 70 } { \__unravel_thing_last_item: } % last_item
1566     { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
1567     { 72 } { 5 } % assign_toks
1568     { 73 } { 0 } % assign_int
1569     { 74 } { 1 } % assign_dimen
1570     { 75 } { 2 } % assign_glue

```

```

1571     { 76 } { 3                               } % assign_mu_glue
1572     { 77 } { 1 \_unravel_scan_font_dimen:   } % assign_font_dimen
1573     { 78 } { 0 \_unravel_scan_font_int:     } % assign_font_int
1574     { 79 } { 0                               } % set_aux
1575     { 80 } { 0                               } % set_prev_graf
1576     { 81 } { 1                               } % set_page_dimen
1577     { 82 } { 0                               } % set_page_int
1578     { 83 } { 1 \_unravel_scan_int:         } % set_box_dimen
1579     { 84 } { 0 \_unravel_scan_int:         } % set_shape
1580     { 85 } { 0 \_unravel_scan_int:         } % def_code
1581     { 86 } { 4 \_unravel_scan_font_ident:   } % def_family
1582     { 87 } { 4 \_unravel_scan_font_ident:   } % set_font
1583     { 88 } { 4 \_unravel_scan_font_ident:   } % def_font
1584     { 89 } { \_unravel_thing_register:     } % register
1585   }
1586   { 8 }
1587 }
1588 \cs_new_nopar:Npn \_unravel_thing_last_item:
1589 {
1590   \int_compare:nNnTF \l_unravel_head_char_int < { 26 }
1591   {
1592     \int_case:nnF \l_unravel_head_char_int
1593     {
1594       { 1 } { 1 } % lastkern
1595       { 2 } { 2 } % lastskip
1596     }
1597     { 0 } % other integer parameters
1598   }
1599   {
1600     \int_case:nnF \l_unravel_head_char_int
1601     {
1602       { 26 } { 0 \_unravel_scan_normal_glue: } % gluestretchorder
1603       { 27 } { 0 \_unravel_scan_normal_glue: } % glueshrinkorder
1604       { 28 } % fontcharwd
1605       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1606       { 29 } % fontcharht
1607       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1608       { 30 } % fontchardp
1609       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1610       { 31 } % fontcharic
1611       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
1612       { 32 } { 1 \_unravel_scan_int: } % parshaplength
1613       { 33 } { 1 \_unravel_scan_int: } % parshapeindent
1614       { 34 } { 1 \_unravel_scan_int: } % parshapedimen
1615       { 35 } { 1 \_unravel_scan_normal_glue: } % gluestretch
1616       { 36 } { 1 \_unravel_scan_normal_glue: } % glueshrink
1617       { 37 } { 2 \_unravel_scan_mu_glue: } % mutoglu
1618       { 38 } { 3 \_unravel_scan_normal_glue: } % glutomu
1619       { 39 } % numepr
1620       { 0 \_unravel_scan_expr:N \_unravel_scan_int: }

```

```

1621         { 40 } % dimexpr
1622         { 1 \_unravel_scan_expr:N \_unravel_scan_normal_dimen: }
1623         { 41 } % glueexpr
1624         { 2 \_unravel_scan_expr:N \_unravel_scan_normal_glue: }
1625         { 42 } % muexpr
1626         { 3 \_unravel_scan_expr:N \_unravel_scan_mu_glue: }
1627     }
1628     { }
1629 }
1630 }

```

```

1631 \cs_new_nopar:Npn \_unravel_thing_register:
1632 {
1633     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
1634     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = \c_zero
1635     { \_unravel_scan_int: }
1636 }

```

*(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.

```

1637 \cs_new_protected:Npn \_unravel_scan_toks_register:
1638 {
1639     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
1640     { \_unravel_scan_int: }
1641 }

```

*(End definition for \\_unravel\_scan\_toks\_register:.)*

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

```

1642 \cs_new:Npn \_unravel_thing_use:nnN #1#2
1643 {
1644     \int_case:nnF { \int_min:nn {#1} {#2} }
1645     {
1646         { 0 } \int_eval:n
1647         { 1 } \dim_eval:n
1648         { 2 } \skip_eval:n
1649         { 3 } \muskip_eval:n
1650     }
1651     { \tex_the:D }
1652 }

```

*(End definition for \\_unravel\_thing\_use:nnN.)*

```

\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
1653 \cs_new_protected:Npn \_unravel_scan_expr:N #1
1654 { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
1655 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
1656 {
1657     \_unravel_get_x_non_blank:
1658     \_unravel_scan_factor:N #1
1659     \_unravel_scan_expr_op:NN #1#2

```



```

1660 }
1661 \cs_new_protected:Npn \__unravel_scan_expr_op:NN #1#2
1662 {
1663   \__unravel_get_x_non_blank:
1664   \tl_case:NnF \l__unravel_head_tl
1665   {
1666     \c__unravel_plus_tl
1667     {
1668       \__unravel_prev_input:V \l__unravel_head_tl
1669       \__unravel_scan_expr_aux:NN #1#2
1670     }
1671     \c__unravel_minus_tl
1672     {
1673       \__unravel_prev_input:V \l__unravel_head_tl
1674       \__unravel_scan_expr_aux:NN #1#2
1675     }
1676     \c__unravel_times_tl
1677     {
1678       \__unravel_prev_input:V \l__unravel_head_tl
1679       \__unravel_get_x_non_blank:
1680       \__unravel_scan_factor:N \__unravel_scan_int:
1681       \__unravel_scan_expr_op:NN #1#2
1682     }
1683     \c__unravel_over_tl
1684     {
1685       \__unravel_prev_input:V \l__unravel_head_tl
1686       \__unravel_get_x_non_blank:
1687       \__unravel_scan_factor:N \__unravel_scan_int:
1688       \__unravel_scan_expr_op:NN #1#2
1689     }
1690     \c__unravel_rp_tl
1691     {
1692       \bool_if:NTF #2
1693       { \__unravel_prev_input:V \l__unravel_head_tl }
1694       { \__unravel_back_input: }
1695     }
1696   }
1697   {
1698     \bool_if:NTF #2
1699     {
1700       \msg_error:nn { unravel } { missing-rparen }
1701       \__unravel_back_input:
1702       \__unravel_prev_input:V \c__unravel_rp_tl
1703     }
1704     {
1705       \token_if_eq_meaning:NnF \l__unravel_head_token \scan_stop:
1706       { \__unravel_back_input: }
1707     }
1708   }
1709 }

```

```

1710 \cs_new_protected:Npn \__unravel_scan_factor:N #1
1711 {
1712   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
1713   {
1714     \__unravel_prev_input:V \l__unravel_head_tl
1715     \__unravel_scan_expr_aux:NN #1 \c_true_bool
1716   }
1717   {
1718     \__unravel_back_input:
1719     #1
1720   }
1721 }

```

(End definition for \\_\_unravel\_scan\_expr:N. This function is documented on page ??.)

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

```

1722 \cs_new_protected_nopar:Npn \__unravel_scan_signs:
1723 {
1724   \__unravel_get_x_non_blank:
1725   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
1726   {
1727     \__unravel_prev_input:V \l__unravel_head_tl
1728     \__unravel_scan_signs:
1729   }
1730   {
1731     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
1732     {
1733       \__unravel_prev_input:V \l__unravel_head_tl
1734       \__unravel_scan_signs:
1735     }
1736   }
1737 }

```

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

```

\__unravel_scan_int:
\__unravel_scan_int_char:
\__unravel_scan_int_lq:
\__unravel_scan_int_explicit:n
1738 \cs_new_protected_nopar:Npn \__unravel_scan_int:
1739 {
1740   \__unravel_scan_signs:
1741   \__unravel_set_cmd:
1742   \int_compare:nNnTF
1743   \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
1744   { \__unravel_scan_int_char: }
1745   {
1746     \int_compare:nNnTF
1747     \l__unravel_head_cmd_int
1748     > { \__unravel_tex_use:n { max_internal } }
1749     { \__unravel_scan_int_char: }
1750     { \__unravel_scan_something_internal:n { 0 } }
1751   }

```

```

1752 }
1753 \cs_new_protected_nopar:Npn \__unravel_scan_int_char:
1754 {
1755   \tl_case:NnF \l__unravel_head_tl
1756   {
1757     \c__unravel_lq_tl { \__unravel_scan_int_lq: }
1758     \c__unravel_rq_tl
1759     {
1760       \__unravel_prev_input:V \l__unravel_head_tl
1761       \__unravel_get_x_next:
1762       \__unravel_scan_int_explicit:n { ' }
1763     }
1764     \c__unravel_dq_tl
1765     {
1766       \__unravel_prev_input:V \l__unravel_head_tl
1767       \__unravel_get_x_next:
1768       \__unravel_scan_int_explicit:n { " }
1769     }
1770   }
1771   { \__unravel_scan_int_explicit:n { } }
1772 }
1773 \cs_new_protected_nopar:Npn \__unravel_scan_int_lq:
1774 {
1775   \__unravel_get_next:
1776   \__unravel_gtl_if_head_is_definable:Nf \l__unravel_head_gtl
1777   {
1778     \tl_set:Nx \l__unravel_head_tl
1779     { \__unravel_token_to_char:N \l__unravel_head_token }
1780   }
1781   \tl_set:Nx \l__unravel_tmpa_tl
1782   { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
1783   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
1784   \__unravel_print_action:x
1785   { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
1786   \__unravel_skip_optional_space:
1787 }
1788 \cs_new_protected:Npn \__unravel_scan_int_explicit:n #1
1789 {
1790   \if_int_compare:w \c_one
1791   < #1 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
1792   \exp_after:wN \use_i:nn
1793   \else:
1794   \exp_after:wN \use_ii:nn
1795   \fi:
1796   {
1797     \__unravel_prev_input:V \l__unravel_head_tl
1798     \__unravel_get_x_next:
1799     \__unravel_scan_int_explicit:n {#1}
1800   }
1801   {

```

```

1802     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1803     { \__unravel_back_input: }
1804   }
1805 }

```

(End definition for \\_\_unravel\_scan\_int:. This function is documented on page ??.)

\\_\_unravel\_scan\_normal\_dimen:

```

1806 \cs_new_protected_nopar:Npn \__unravel_scan_normal_dimen:
1807 { \__unravel_scan_dimen:NN \c_false_bool \c_false_bool }

```

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

\\_\_unravel\_scan\_dimen:NN Quoth tex.web. “*mu is true if the finite units must be ‘mu’, while mu is false if ‘mu’ units are disallowed; inf is true if the infinite units ‘fil’, ‘fill’, ‘filll’ are permitted.*” The function here has the same first two parameters as TeX’s scan\_dimen, but omits the third, as the shortcut is provided as a separate function, \\_\_unravel\_scan\_dimen\_unit:NN.

```

1808 \cs_new_protected:Npn \__unravel_scan_dimen:NN #1#2
1809 {
1810   \__unravel_scan_signs:
1811   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
1812   \__unravel_set_cmd:
1813   \int_compare:nNnTF
1814     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
1815     { \__unravel_scan_dimen_char:NN #1#2 }
1816     {
1817       \int_compare:nNnTF
1818         \l__unravel_head_cmd_int
1819         > { \__unravel_tex_use:n { max_internal } }
1820         { \__unravel_scan_dimen_char:NN #1#2 }
1821         {
1822           \bool_if:NTF #1
1823           {
1824             \__unravel_scan_something_internal:n { 3 }
1825             \int_case:nnF \g__unravel_val_level_int
1826             {
1827               { 0 } { \__unravel_scan_dim_unit:NN #1 #2 }
1828               { 3 } { }
1829             }
1830             {
1831               \msg_error:nn { unravel } { incompatible-units }
1832               % ^^A todo: error recovery
1833             }
1834           }
1835           {
1836             \__unravel_scan_something_internal:n { 2 }
1837             \int_case:nnF \g__unravel_val_level_int
1838             {
1839               { 0 } { \__unravel_scan_dim_unit:NN #1#2 }
1840               { 3 } % ^^A todo: error recovery
1841               { \msg_error:nn { unravel } { incompatible-units } }

```

```

1842         }
1843         { }
1844     }
1845 }
1846 }
1847 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1848 \__unravel_prev_input_silent:V \l__unravel_head_tl
1849 }
1850 \cs_new_protected:Npn \__unravel_scan_dimen_char:NN #1#2
1851 {
1852     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
1853     { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
1854     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
1855     {
1856         \__unravel_prev_input:n { . }
1857         \__unravel_scan_decimal_loop:
1858     }
1859     {
1860         \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl
1861         {
1862             \__unravel_back_input:
1863             \__unravel_scan_int:
1864             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
1865             { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
1866             \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
1867             {
1868                 \__unravel_input_gpop:N \l__unravel_tmpb_gtl
1869                 \__unravel_prev_input:n { . }
1870                 \__unravel_scan_decimal_loop:
1871             }
1872         }
1873     }
1874     \__unravel_back_input:
1875     \__unravel_scan_int:
1876 }
1877 }
1878 \__unravel_scan_dim_unit:NN #1#2
1879 }
1880 \cs_new_protected:Npn \__unravel_scan_dim_unit:NN #1#2
1881 {
1882     \bool_if:NT #2
1883     {
1884         \__unravel_scan_keyword:nT { fF iI lL }
1885         {
1886             \__unravel_scan_inf_unit_loop:
1887             \__unravel_break:w
1888         }
1889     }
1890     \__unravel_get_x_non_blank:
1891     \__unravel_set_cmd:

```

```

1892 \int_compare:nNnTF
1893   \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
1894   { \__unravel_back_input: }
1895   {
1896     \int_compare:nNnTF
1897       \l__unravel_head_cmd_int
1898       > { \__unravel_tex_use:n { max_internal } }
1899       { \__unravel_back_input: }
1900       {
1901         \seq_gput_right:Nn \g__unravel_prev_input_seq { }
1902         \bool_if:NTF #1
1903           { \__unravel_scan_something_internal:n { 3 } }
1904           { \__unravel_scan_something_internal:n { 2 } }
1905         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
1906         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
1907         \tl_set:Nx \l__unravel_tmpa_tl
1908         {
1909           \bool_if:NTF #1 \muskip_eval:n \skip_eval:n
1910           {
1911             \l__unravel_tmpa_tl
1912             \bool_if:NTF #1 \etex_muexpr:D \etex_glueexpr:D
1913             \l__unravel_head_tl
1914           }
1915         }
1916         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_tmpa_tl
1917         \__unravel_break:w
1918       }
1919     }
1920 \bool_if:NT #1
1921 {
1922   \__unravel_scan_keyword:nT { mM uU } { \__unravel_break:w }
1923   \msg_error:nn { unravel } { missing-mudim }
1924   \__unravel_break:w
1925 }
1926 \__unravel_scan_keyword:nT { eE mM } { \__unravel_break:w }
1927 \__unravel_scan_keyword:nT { eE xX } { \__unravel_break:w }
1928 \__unravel_scan_keyword:nT { pP xX } { \__unravel_break:w }
1929 \__unravel_scan_keyword:n { tT rR uU eE }
1930 \__unravel_scan_keyword:nT { pP tT } { \__unravel_break:w }
1931 \__unravel_scan_keyword:nT { iI nN } { \__unravel_break:w }
1932 \__unravel_scan_keyword:nT { pP cC } { \__unravel_break:w }
1933 \__unravel_scan_keyword:nT { cC mM } { \__unravel_break:w }
1934 \__unravel_scan_keyword:nT { mM mM } { \__unravel_break:w }
1935 \__unravel_scan_keyword:nT { bB pP } { \__unravel_break:w }
1936 \__unravel_scan_keyword:nT { dD dD } { \__unravel_break:w }
1937 \__unravel_scan_keyword:nT { cC cC } { \__unravel_break:w }
1938 \__unravel_scan_keyword:nT { nN dD } { \__unravel_break:w }
1939 \__unravel_scan_keyword:nT { nN cC } { \__unravel_break:w }
1940 \__unravel_scan_keyword:nT { sS pP } { \__unravel_break:w }
1941 \__unravel_break_point:

```

```

1942 }
1943 \cs_new_protected_nopar:Npn \__unravel_scan_inf_unit_loop:
1944 { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
1945 \cs_new_protected_nopar:Npn \__unravel_scan_decimal_loop:
1946 {
1947   \__unravel_get_x_next:
1948   \tl_if_empty:NTF \l__unravel_head_tl
1949     { \use_ii:nn }
1950     { \tl_if_in:nVTF { 0123456789 } \l__unravel_head_tl }
1951     {
1952       \__unravel_prev_input:V \l__unravel_head_tl
1953       \__unravel_scan_decimal_loop:
1954     }
1955     {
1956       \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
1957       { \__unravel_back_input: }
1958       \__unravel_prev_input_silent:n { ~ }
1959     }
1960 }

```

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

```

\__unravel_scan_normal_glue:
  \__unravel_scan_mu_glue:

```

```

1961 \cs_new_protected_nopar:Npn \__unravel_scan_normal_glue:
1962 { \__unravel_scan_glue:n { 2 } }
1963 \cs_new_protected_nopar:Npn \__unravel_scan_mu_glue:
1964 { \__unravel_scan_glue:n { 3 } }

```

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

```

\__unravel_scan_glue:n

```

```

1965 \cs_new_protected:Npn \__unravel_scan_glue:n #1
1966 {
1967   \int_compare:nNnTF {#1} = { 2 }
1968     { \__unravel_scan_glue_aux:nN {#1} \c_false_bool }
1969     { \__unravel_scan_glue_aux:nN {#1} \c_true_bool }
1970 }
1971 \cs_new_protected:Npn \__unravel_scan_glue_aux:nN #1#2
1972 {
1973   \__unravel_scan_signs:
1974   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
1975   \__unravel_set_cmd:
1976   \int_compare:nNnTF
1977     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
1978     { \__unravel_back_input: \__unravel_scan_dimen:NN #2 \c_false_bool }
1979     {
1980       \int_compare:nNnTF
1981         \l__unravel_head_cmd_int
1982         > { \__unravel_tex_use:n { max_internal } }
1983         {
1984           \__unravel_back_input:

```

```

1985     \_unravel_scan_dimen:NN #2 \c_false_bool
1986   }
1987   {
1988     \_unravel_scan_something_internal:n {#1}
1989     \int_case:nnF \g__unravel_val_level_int
1990     {
1991       { 0 } { \_unravel_scan_dimen:NN #2 \c_false_bool }
1992       { 1 } { \bool_if:NT #2 { \msg_error: } } % ^^A todo: ??
1993     }
1994     {
1995       \int_compare:nNnF \g__unravel_val_level_int = {#1}
1996       { \msg_error:nn { unravel } { incompatible-units } }
1997       \_unravel_break:w
1998     }
1999   }
2000 }
2001 \_unravel_scan_keyword:nT { pP lL uU sS }
2002   { \_unravel_scan_dimen:NN #2 \c_true_bool }
2003 \_unravel_scan_keyword:nT { mM iI nN uU sS }
2004   { \_unravel_scan_dimen:NN #2 \c_true_bool }
2005 \_unravel_break_point:
2006 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2007 \_unravel_prev_input_silent:V \l__unravel_head_tl
2008 }

```

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

\\_unravel\_scan\_file\_name:

```

2009 \cs_new_protected_nopar:Npn \_unravel_scan_file_name:
2010 {
2011   \bool_gset_true:N \g__unravel_name_in_progress_bool
2012   \_unravel_get_x_non_blank:
2013   \_unravel_scan_file_name_loop:
2014   \bool_gset_false:N \g__unravel_name_in_progress_bool
2015   \_unravel_prev_input_silent:n { ~ }
2016 }
2017 \cs_new_protected_nopar:Npn \_unravel_scan_file_name_loop:
2018 {
2019   \_unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2020   { \_unravel_back_input: }
2021   {
2022     \tl_set:Nx \l__unravel_tmpa_tl
2023     { \_unravel_token_to_char:N \l__unravel_head_token }
2024     \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl
2025     {
2026       \_unravel_prev_input_silent:V \l__unravel_tmpa_tl
2027       \_unravel_get_x_next:
2028       \_unravel_scan_file_name_loop:
2029     }
2030   }
2031 }

```



(End definition for `\_unravel_scan_file_name:`.)

`\_unravel_scan_r_token:` This is analogous to TeX's `get_r_token`. We store in `\l_unravel_defined_tl` the token which we found, as this is what will be defined by the next assignment.

```
2032 \cs_new_protected_nopar:Npn \_unravel_scan_r_token:
2033   {
2034     \bool_do_while:nn
2035       { \tl_if_eq_p:NN \l_unravel_head_tl \c_space_tl }
2036       { \_unravel_get_next: }
2037     \_unravel_gtl_if_head_is_definable:NF \l_unravel_head_gtl
2038     {
2039       \msg_error:nn { unravel } { missing-cs }
2040       \_unravel_back_input:
2041       \tl_set:Nn \l_unravel_head_tl { \_unravel_inaccessible:w }
2042     }
2043     \_unravel_prev_input_silent:V \l_unravel_head_tl
2044     \tl_set_eq:NN \l_unravel_defined_tl \l_unravel_head_tl
2045   }
```

(End definition for `\_unravel_scan_r_token:`.)

`\_unravel_scan_toks_to_str:`

```
2046 \cs_new_protected:Npn \_unravel_scan_toks_to_str:
2047   {
2048     \seq_gput_right:Nn \g_unravel_prev_input_seq { }
2049     \_unravel_scan_toks:NN \c_false_bool \c_true_bool
2050     \seq_gpop_right:NN \g_unravel_prev_input_seq \l_unravel_tmpa_tl
2051     \_unravel_prev_input_silent:x
2052     { { \exp_after:wN \tl_to_str:n \l_unravel_tmpa_tl } }
2053   }
```

(End definition for `\_unravel_scan_toks_to_str:`.)

`\_unravel_scan_toks:NN`

```
2054 \cs_new_protected:Npn \_unravel_scan_toks:NN #1#2
2055   {
2056     \bool_if:NT #1 { \_unravel_scan_param: }
2057     \_unravel_scan_left_brace:
2058     \bool_if:NTF #2
2059       { \_unravel_scan_group_x:N #1 }
2060       { \_unravel_scan_group_n:N #1 }
2061   }
```

(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`.

```
2062 \cs_new_protected_nopar:Npn \_unravel_scan_param:
2063   {
2064     \tl_clear:N \l_unravel_tmpa_tl
```

```

2065 \__unravel_scan_param_aux:
2066 \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
2067 \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2068 }
2069 \cs_new_protected_nopar:Npn \__unravel_scan_param_aux:
2070 {
2071 \__unravel_get_next:
2072 \tl_concat:NNN \l__unravel_tmpa_tl
2073 \l__unravel_tmpa_tl \l__unravel_head_tl
2074 \tl_if_empty:NTF \l__unravel_head_tl
2075 { \__unravel_back_input: } { \__unravel_scan_param_aux: }
2076 }

```

(End definition for \\_\_unravel\_scan\_param:. This function is documented on page ??.)

\\_\_unravel\_scan\_group\_n:N

```

2077 \cs_new_protected:Npn \__unravel_scan_group_n:N #1
2078 {
2079 \__unravel_back_input:
2080 \__unravel_input_gpop_item:NF \l__unravel_head_tl
2081 {
2082 \msg_error:nn { unravel } { runaway-text }
2083 \__unravel_exit:w
2084 }
2085 \tl_set:Nx \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
2086 \bool_if:NT #1
2087 { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
2088 \__unravel_prev_input_silent:V \l__unravel_head_tl
2089 }

```

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

\\_\_unravel\_scan\_group\_x:N

```

2090 \cs_new_protected:Npn \__unravel_scan_group_x:N #1
2091 {
2092 \__unravel_input_gpop_tl:N \l__unravel_head_tl
2093 \__unravel_back_input:V \l__unravel_head_tl
2094 \bool_if:NTF #1
2095 {
2096 \__unravel_prev_input_silent:V \c_left_brace_str
2097 \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2098 \__unravel_scan_group_xdef:n { 1 }
2099 }
2100 {
2101 \seq_gput_right:NV \g__unravel_prev_input_seq \c_empty_gtl
2102 \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2103 \__unravel_scan_group_x:n { 1 }
2104 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpb_gtl
2105 \__unravel_prev_input_silent:x
2106 { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2107 }
2108 }

```

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

\\_unravel\_scan\_group\_xdef:n

```
2109 \cs_new_protected:Npn \_unravel_scan_group_xdef:n #1
2110 {
2111   \_unravel_get_token_x:N \c_true_bool
2112   \tl_if_empty:NTF \l__unravel_head_tl
2113   {
2114     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2115     {
2116       \_unravel_prev_input_silent:V \c_left_brace_str
2117       \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2118       \_unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2119     }
2120     {
2121       \_unravel_prev_input_silent:V \c_right_brace_str
2122       \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2123       \int_compare:nNnF {#1} = \c_one
2124       { \_unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2125     }
2126   }
2127   {
2128     \_unravel_prev_input_silent:V \l__unravel_head_tl
2129     \tl_put_right:Nx \l__unravel_defining_tl
2130     { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2131     \_unravel_scan_group_xdef:n {#1}
2132   }
2133 }
2134 \cs_generate_variant:Nn \_unravel_scan_group_xdef:n { f }
```

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

\\_unravel\_scan\_group\_x:n

```
2135 \cs_new_protected:Npn \_unravel_scan_group_x:n #1
2136 {
2137   \_unravel_get_token_x:N \c_false_bool
2138   \_unravel_prev_input_gtl:N \l__unravel_head_gtl
2139   \tl_if_empty:NTF \l__unravel_head_tl
2140   {
2141     \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2142     { \_unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2143     {
2144       \int_compare:nNnF {#1} = \c_one
2145       { \_unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2146     }
2147   }
2148   { \_unravel_scan_group_x:n {#1} }
2149 }
2150 \cs_generate_variant:Nn \_unravel_scan_group_x:n { f }
```

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

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

```
2151 \cs_new_protected:Npn \__unravel_get_token_x:N #1
2152 {
2153   \__unravel_get_next:
2154   \__unravel_token_if_protected:NF \l__unravel_head_token
2155   {
2156     \__unravel_set_cmd:
2157     \int_compare:nNnTF
2158     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { the } }
2159     {
2160       \__unravel_get_the:
2161       \bool_if:NTF #1
2162       {
2163         \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
2164         \__unravel_prev_input:V \l__unravel_head_tl
2165       }
2166       {
2167         \gtl_set:Nx \l__unravel_tmpb_gtl { \l__unravel_head_tl }
2168         \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
2169         \__unravel_print_action:
2170       }
2171     }
2172     { \__unravel_expand: }
2173     \__unravel_get_token_x:N #1
2174   }
2175 }
```

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

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

```
2176 \cs_new_protected_nopar:Npn \__unravel_scan_alt_rule:
2177 {
2178   \__unravel_scan_keyword:nTF { wWiIdDtThH }
2179   {
2180     \__unravel_scan_normal_dimen:
2181     \__unravel_scan_alt_rule:
2182   }
2183   {
2184     \__unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2185     {
2186       \__unravel_scan_normal_dimen:
2187       \__unravel_scan_alt_rule:
2188     }
2189     {
2190       \__unravel_scan_keyword:nT { dDeEpPtThH }
2191       {
2192         \__unravel_scan_normal_dimen:
2193         \__unravel_scan_alt_rule:
2194       }
2195     }
2196 }
```

```

2196     }
2197   }
(End definition for \_unravel_scan_alt_rule:.)

```

`\_unravel_scan_spec:` Some T<sub>E</sub>X primitives accept the keywords `to` and `spread`, followed by a dimension.

```

2198 \cs_new_protected_nopar:Npn \_unravel_scan_spec:
2199   {
2200     \_unravel_scan_keyword:nTF { tT oO } { \_unravel_scan_normal_dimen: }
2201     {
2202       \_unravel_scan_keyword:nT { sS pP rR eE aA dD }
2203       { \_unravel_scan_normal_dimen: }
2204     }
2205     \_unravel_scan_left_brace:
2206   }
(End definition for \_unravel_scan_spec:.)

```

## 2.8 Working with boxes

`\_unravel_do_box:N` When this procedure is called, the last item in `\g\_unravel_prev_input_seq` is

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

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

```

2207 \cs_new_protected:Npn \_unravel_do_box:N #1
2208   {
2209     \_unravel_get_x_non_relax:
2210     \_unravel_set_cmd:
2211     \int_compare:nNnTF
2212       \l__unravel_head_cmd_int = { \_unravel_tex_use:n { make_box } }
2213       { \_unravel_do_begin_box:N #1 }
2214       {
2215         \bool_if:NTF #1
2216         {
2217           \_unravel_cs_case:NnF \l__unravel_head_token
2218           {
2219             \tex_hrulerule:D { \_unravel_do_leaders_rule: }
2220             \tex_vrulerule:D { \_unravel_do_leaders_rule: }
2221           }
2222           { \_unravel_do_box_error: }
2223         }
2224         { \_unravel_do_box_error: }

```

```

2225     }
2226   }
(End definition for \_unravel_do_box:N.)

```

`\_unravel_do_box_error:` Put the (non-`make_box`) command back into the input and complain. Then recover by throwing away the action (last item of `\g__unravel_prev_input_seq`). For some reason (this appears to be what `TEX` does), there is no need to remove the after assignment token here.

```

2227 \cs_new_protected_nopar:Npn \_unravel_do_box_error:
2228   {
2229     \_unravel_back_input:
2230     \msg_error:nn { unravel } { missing-box }
2231     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2232     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2233   }
(End definition for \_unravel_do_box_error:.)

```

`\_unravel_do_begin_box:N` We have just found a `make_box` command and placed it into the last item of `\g__unravel_prev_input_seq`. If it is “simple” (`\box<int>`, `\copy<int>`, `\lastbox`, `\vsplit<int>` to `<dim>`) then we grab its operands, then call `\_unravel_do_simple_box:N` to finish up. If it is `\vtop` or `\vbox` or `\hbox`, we need to work harder.

```

2234 \cs_new_protected:Npn \_unravel_do_begin_box:N #1
2235   {
2236     \_unravel_prev_input:V \l__unravel_head_tl
2237     \int_case:nnTF \l__unravel_head_char_int
2238       {
2239         { 0 } { \_unravel_scan_int: } % box
2240         { 1 } { \_unravel_scan_int: } % copy
2241         { 2 } { } % lastbox
2242         { 3 } % vsplit
2243         {
2244           \_unravel_scan_int:
2245           \_unravel_scan_keyword:nF { tT o0 }
2246           {
2247             \msg_error:nn { unravel } { missing-to }
2248             \_unravel_prev_input:n { to }
2249           }
2250           \_unravel_scan_normal_dimen:
2251         }
2252       }
2253     { \_unravel_do_simple_box:N #1 }
2254     { \_unravel_do_box_explicit:N #1 }
2255   }
(End definition for \_unravel_do_begin_box:N.)

```

`\_unravel_do_simple_box:N` For leaders, we need to fetch a glue. In all cases, retrieve the box construction (such as `\raise3pt\vsplit7to5em`). Finally, let `TEX` run the code and print what we have done.

```

2256 \cs_new_protected:Npn \_unravel_do_simple_box:N #1

```

```

2257 {
2258   \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2259   {
2260     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2261     \tl_use:N \l__unravel_head_tl \scan_stop:
2262     \gtl_put_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2263     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2264   }
2265 }

```

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

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

```

2266 \cs_new_protected_nopar:Npn \__unravel_do_leaders_fetch_skip:
2267 {
2268   \__unravel_get_x_non_relax:
2269   \__unravel_set_cmd:
2270   \int_compare:nNnTF \l__unravel_head_cmd_int
2271   = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2272   {
2273     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2274     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2275     \__unravel_do_append_glue:
2276   }
2277   {
2278     \__unravel_back_input:
2279     \msg_error:nn { unravel } { improper-leaders }
2280     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2281     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2282   }
2283 }

```

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

\\_\_unravel\_do\_box\_explicit:N

At this point, the last item in \g\_\_unravel\_prev\_input\_seq is typically \setbox0\hbox or \raise 3pt\hbox. Scan for keywords to and spread and a left brace. Install a hook in \everyhbox or \everyvbox (whichever T<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 \g\_\_unravel\_prev\_input\_seq). 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 booleans: the top is true if the innermost box is part of leaders.

```

2284 \cs_new_protected:Npn \__unravel_do_box_explicit:N #1
2285 {
2286   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hbox:D
2287   { \__unravel_box_hook:N \tex_everyhbox:D }
2288   { \__unravel_box_hook:N \tex_everyvbox:D }
2289   % ^^A todo: TeX calls |normal_paragraph| here.

```

```

2290 \__unravel_scan_spec:
2291 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2292 \__unravel_set_action_text:x
2293   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2294 \seq_push:Nx \l__unravel_leaders_box_seq
2295   { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2296 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2297 \gtl_gconcat:NNN \g__unravel_output_gtl
2298   \g__unravel_output_gtl \c_group_begin_gtl
2299 \tl_use:N \l__unravel_head_tl
2300   \c_group_begin_token \__unravel_box_hook_end:
2301 }

```

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

```

\__unravel_box_hook:N
\__unravel_box_hook:w
\__unravel_box_hook_end:
2302 \cs_new_protected:Npn \__unravel_box_hook:N #1
2303   {
2304   \tl_set:NV \l__unravel_tmpa_tl #1
2305   \str_if_eq_x:nnF
2306     { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
2307     {
2308       \exp_args:Nx #1
2309       {
2310         \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
2311         \exp_not:V #1
2312       }
2313     }
2314   \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
2315     {
2316       \exp_args:No #1 {##1}
2317       \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2318       \__unravel_print_action:
2319       \__unravel_back_input:o {##1}
2320       \__unravel_set_action_text:x
2321       { \token_to_meaning:N #1 \tl_to_str:o {##1} }
2322       \tl_if_empty:oF {##1} { \__unravel_print_action: }
2323     }
2324   }
2325   \cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
2326   \cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:

```

(End definition for \\_\_unravel\_box\_hook:N. This function is documented on page ??.)

\\_\_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.

```

2327 \cs_new_protected_nopar:Npn \__unravel_do_leaders_rule:
2328   {
2329   \__unravel_prev_input:V \l__unravel_head_tl
2330   \__unravel_scan_alt_rule:

```



```

2331   \__unravel_do_leaders_fetch_skip:
2332   }
(End definition for \__unravel_do_leaders_rule:.)

```

## 2.9 Paragraphs

\\_unravel\_charcode\_if\_safe:nTF

```

2333 \prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
2334 {
2335   \bool_if:nTF
2336   {
2337     \int_compare_p:n { #1 = '!' }
2338     || \int_compare_p:n { ' ' <= #1 <= '[' }
2339     || \int_compare_p:n { #1 = ']' }
2340     || \int_compare_p:n { ' ' <= #1 <= 'z' }
2341   }
2342   { \prg_return_true: }
2343   { \prg_return_false: }
2344 }
(End definition for \__unravel_charcode_if_safe:nTF.)

```

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

```

2345 \group_begin:
2346 \char_set_catcode_other:n { 0 }
2347 \cs_new_protected:Npn \__unravel_char:n #1
2348 {
2349   \tex_char:D #1 \scan_stop:
2350   \__unravel_charcode_if_safe:nTF {#1}
2351   {
2352     \group_begin:
2353     \char_set_lccode:nn { 0 } {#1}
2354     \tex_lowercase:D
2355     { \group_end: \tl_set:Nn \l__unravel_tmpa_tl { ^^@ } }
2356   }
2357   {
2358     \tl_set:Nx \l__unravel_tmpa_tl
2359     { \exp_not:N \char \int_eval:n {#1} ~ }
2360   }
2361   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2362   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2363 }
2364 \group_end:
2365 \cs_generate_variant:Nn \__unravel_char:n { V , x }
(End definition for \__unravel_char:n, \__unravel_char:V, and \__unravel_char:x.)

```

\\_unravel\_char\_in\_mmode:n  
\\_unravel\_char\_in\_mmode:V  
\\_unravel\_char\_in\_mmode:x

```

2366 \group_begin:
2367 \char_set_catcode_other:n { 0 }
2368 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1

```

```

2369 {
2370   \int_compare:nNnTF { \tex_mathcode:D #1 } = { "8000 }
2371     { % math active
2372       \group_begin:
2373         \char_set_lccode:nn { 0 } { \l__unravel_tmpa_tl }
2374         \tex_lowercase:D
2375         { \group_end: \gtl_set:Nn \l__unravel_head_gtl { ^^@ } }
2376         \__unravel_back_input:
2377       }
2378     { \__unravel_char:n {#1} }
2379   }
2380 \group_end:
2381 \cs_generate_variant:Nn \__unravel_char_in_mmode:n { V , x }
(End definition for \__unravel_char_in_mmode:n, \__unravel_char_in_mmode:V, and \__unravel_char_in_mmode:x.)

```

`\__unravel_mathchar:n`  
`\__unravel_mathchar:x`

```

2382 \cs_new_protected:Npn \__unravel_mathchar:n #1
2383 {
2384   \tex_mathchar:D #1 \scan_stop:
2385   \tl_set:Nx \l__unravel_tmpa_tl
2386     { \exp_not:N \mathchar \int_eval:n {#1} ~ }
2387   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2388   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2389 }
2390 \cs_generate_variant:Nn \__unravel_mathchar:n { x }
(End definition for \__unravel_mathchar:n and \__unravel_mathchar:x.)

```

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

```

2391 \cs_new_protected:Npn \__unravel_new_graf:N #1
2392 {
2393   \tl_set:NV \l__unravel_tmpa_tl \tex_everypar:D
2394   \tex_everypar:D { }
2395   \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2396   \exp_args:NV \tex_everypar:D \l__unravel_tmpa_tl
2397   \__unravel_back_input:V \l__unravel_tmpa_tl
2398   \__unravel_print_action:x
2399     {
2400       \g__unravel_action_text_str \c_space_tl : ~
2401       \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2402     }
2403 }
(End definition for \__unravel_new_graf:N.)

```

`\__unravel_end_graf:`

```

2404 \cs_new_protected_nopar:Npn \__unravel_end_graf:
2405 { \mode_if_horizontal:T { \__unravel_normal_paragraph: } }

```

(End definition for `\__unravel_end_graf:.`)

`\__unravel_normal_paragraph:`

```
2406 \cs_new_protected_nopar:Npn \__unravel_normal_paragraph:
2407   {
2408     \tex_par:D
2409     \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2410     \__unravel_print_action:x { Paragraph~end. }
2411   }
```

(End definition for `\__unravel_normal_paragraph:.`)

`\__unravel_build_page:`

```
2412 \cs_new_protected_nopar:Npn \__unravel_build_page:
2413   {
2414   }
```

(End definition for `\__unravel_build_page:.`)

## 2.10 Groups

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

```
2415 \cs_new_protected_nopar:Npn \__unravel_handle_left_brace:
2416   {
2417     \int_case:nnF \etex_currentgrouptype:D
2418     {
2419       { 1 } { \__unravel_end_simple_group: } % simple
2420       { 2 } { \__unravel_end_box_group: } % hbox
2421       { 3 } { \__unravel_end_box_group: } % adjusted_hbox
2422       { 4 } { \__unravel_end_graf: \__unravel_end_box_group: } % vbox
2423       { 5 } { \__unravel_end_graf: \__unravel_end_box_group: } % vtop
2424       { 6 } { \__unravel_end_align_group: } % align
2425       { 7 } { \__unravel_end_no_align_group: } % no_align
2426       { 8 } { \__unravel_end_output_group: } % output
2427       { 9 } { \__unravel_end_math_group: } % math
2428       { 10 } { \__unravel_end_disc_group: } % disc
2429       { 11 } { \__unravel_end_graf: \__unravel_end_simple_group: } % insert
2430       { 12 } { \__unravel_end_graf: \__unravel_end_simple_group: } % vcenter
2431       { 13 } { \__unravel_end_math_choice_group: } % math_choice
2432     }
2433     { % bottom_level, semi_simple, math_shift, math_left
2434       \__unravel_back_input:
2435       \l__unravel_head_token
2436       \__unravel_print_action:
2437     }
2438   }
```

(End definition for `\__unravel_handle_left_brace:.`)

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

```

2439 \cs_new_protected_nopar:Npn \__unravel_end_simple_group:
2440 {
2441   \l__unravel_head_token
2442   \gtl_gconcat:NNN \g__unravel_output_gtl
2443   \g__unravel_output_gtl \c_group_end_gtl
2444   \__unravel_print_action:
2445 }

```

*(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.

```

2446 \cs_new_protected_nopar:Npn \__unravel_end_box_group:
2447 {
2448   \seq_pop:N \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
2449   \str_if_eq_x:nmTF \l__unravel_tmpa_tl { Z }
2450   { \__unravel_end_simple_group: }
2451   {
2452     \__unravel_get_x_non_relax:
2453     \__unravel_set_cmd:
2454     \int_compare:nNnTF \l__unravel_head_cmd_int
2455     = { \__unravel_tex_use:n { \l__unravel_tmpa_tl skip } }
2456     {
2457       \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
2458       \__unravel_do_append_glue:
2459     }
2460     {
2461       \__unravel_back_input:
2462       \c_group_end_token \group_begin: \group_end:
2463       \__unravel_print_action:
2464     }
2465   }
2466 }

```

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

`\__unravel_off_save:`

```

2467 \cs_new_protected_nopar:Npn \__unravel_off_save:
2468 {
2469   \int_compare:nNnTF \etex_currentgrouptype:D = { 0 }
2470   { % bottom-level
2471     \msg_error:nnx { unravel } { extra-close }
2472     { \token_to_meaning:N \l__unravel_head_token }
2473   }
2474   {
2475     \__unravel_back_input:
2476     \int_case:nnF \etex_currentgrouptype:D
2477     {
2478       { 14 } % semi_simple_group

```

```

2479         { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
2480         { 15 } % math_shift_group
2481         { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
2482         { 16 } % math_left_group
2483         { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
2484     }
2485     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
2486     \__unravel_back_input:
2487     \msg_error:nnx { unravel } { off-save }
2488     { \gtl_to_str:N \l__unravel_head_gtl }
2489 }
2490 }

```

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

## 2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
2491 \cs_new_protected:Npn \__unravel_mode_math:n #1
2492 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
2493 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
2494 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
2495 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
2496 {
2497     \mode_if_math:TF
2498     { \__unravel_insert_dollar_error: }
2499     { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
2500 }
2501 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
2502 {
2503     \mode_if_vertical:TF
2504     { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
2505     {#1}
2506 }

```

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

\\_\_unravel\_head\_for\_vmode: See TeX's head\_for\_vmode.

```

2507 \cs_new_protected_nopar:Npn \__unravel_head_for_vmode:
2508 {
2509     \mode_if_inner:TF
2510     {
2511         \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hruler:D
2512         {
2513             \msg_error:nn { unravel } { hruler-bad-mode }
2514             \__unravel_print_action:
2515         }
2516         { \__unravel_off_save: }
2517     }
2518     {
2519         \__unravel_back_input:

```

```

2520         \gtl_set:Nn \l__unravel_head_gtl { \par }
2521         \__unravel_back_input:
2522     }
2523 }

```

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

## 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.

```

2524 \cs_new_protected_nopar:Npn \__unravel_do_step:
2525 {
2526     \__unravel_set_action_text:
2527     \bool_if:NT \l__unravel_debug_bool
2528         { \iow_term:x { Cmd:~\int_use:N \l__unravel_head_cmd_int } }
2529     \cs_if_exist_use:cF
2530         { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
2531         { \msg_error:nnx { unravel } { internal } { unknown-command } }
2532 }

```

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

## 2.13 Commands

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

### 2.13.1 Characters: from 0 to 15

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

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

- `blank_char=10` for blank spaces (catcode 10).
- `the_char=11` for letters (catcode 11).
- `other_char=12` for other characters (catcode 12).
- `par_end=13` for `\par`.
- `stop=14` for `\end` and `\dump`.
- `delim_num=15` for `\delimiter`.

Not implemented at all: `endv`.

`\relax` does nothing.

```
2533 \__unravel_new_tex_cmd:nn { relax } % 0
2534 { \__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.

```
2535 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
2536 {
2537   \gtl_gconcat:NNN \g__unravel_output_gtl
2538   \g__unravel_output_gtl \c_group_begin_gtl
2539   \__unravel_print_action:
2540   \l__unravel_head_token
2541 }
```

```
2542 \__unravel_new_tex_cmd:nn { end-group_char } % 2
2543 { \__unravel_handle_left_brace: }
```

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

```
2544 \__unravel_new_tex_cmd:nn { math_char } % 3
2545 {
2546   \__unravel_mode_non_vertical:n
2547   {
2548     \mode_if_math:TF
2549     {
2550       \int_compare:nNnTF
2551       \etex_currentgrouplevel:D = { 15 } % math_shift_group
2552       { \__unravel_after_math: }
2553       { \__unravel_off_save: }
2554     }
2555     {
2556       \__unravel_get_next:
2557       \token_if_eq_catcode:NNTF
2558       \l__unravel_head_token \c_math_toggle_token
2559       {
2560         \mode_if_inner:TF
2561         { \__unravel_back_input: \__unravel_goto_inner_math: }
2562         { \__unravel_goto_display_math: }
2563       }
2564       { \__unravel_back_input: \__unravel_goto_inner_math: }
```

```

2565     }
2566   }
2567 }

```

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.

```

2568 \__unravel_new_tex_cmd:nn { alignment_char } % 4
2569 { \l__unravel_head_token \__unravel_print_action: }
2570 \__unravel_new_tex_cmd:nn { car_ret } % 5
2571 { \l__unravel_head_token \__unravel_print_action: }
2572 \__unravel_new_tex_cmd:nn { macro_char } % 6
2573 { \l__unravel_head_token \__unravel_print_action: }
2574 \__unravel_new_tex_cmd:nn { superscript_char } % 7
2575 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2576 \__unravel_new_tex_cmd:nn { subscript_char } % 8
2577 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
2578 \cs_new_protected_nopar:Npn \__unravel_sub_sup:
2579 {
2580   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2581   \__unravel_print_action:
2582   \__unravel_scan_math:
2583   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2584   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2585   \tl_use:N \l__unravel_head_tl \scan_stop:
2586   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2587 }
2588 \__unravel_new_tex_cmd:nn { endv } % 9
2589 { \msg_error:nn { unravel } { not-implemented } { alignments } }

```

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

```

2590 \__unravel_new_tex_cmd:nn { blank_char } % 10
2591 {
2592   \mode_if_horizontal:T
2593   {
2594     \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
2595     \l__unravel_head_token
2596   }
2597   \__unravel_print_action:
2598 }

```

Letters and other characters leave vertical mode.

```

2599 \__unravel_new_tex_cmd:nn { the_char } % 11
2600 {
2601   \__unravel_mode_non_vertical:n
2602   {
2603     \tl_set:Nx \l__unravel_tmpa_tl
2604     { ' \__unravel_token_to_char:N \l__unravel_head_token }
2605     \mode_if_math:TF

```



```

2606         { \_unravel_char_in_mmode:V \l\_unravel_tmpa_tl }
2607         { \_unravel_char:V \l\_unravel_tmpa_tl }
2608     }
2609 }
2610 \_unravel_new_eq_tex_cmd:nn { other_char } { the_char }           % 12
2611 \_unravel_new_tex_cmd:nn { par_end }                               % 13
2612 {
2613     \_unravel_mode_non_math:n
2614     {
2615         \mode_if_vertical:TF
2616         { \_unravel_normal_paragraph: }
2617         {
2618             % if align_state<0 then off_save;
2619             \_unravel_end_graf:
2620             \mode_if_vertical:T
2621             { \mode_if_inner:F { \_unravel_build_page: } }
2622         }
2623     }
2624 }
2625 \_unravel_new_tex_cmd:nn { stop }                                   % 14
2626 {
2627     \_unravel_mode_vertical:n
2628     {
2629         \mode_if_inner:TF
2630         { \_unravel_forbidden_case: }
2631         {
2632             % ^^A todo: unless its_all_over
2633             \int_gdecr:N \g\_unravel_ends_int
2634             \int_compare:nNnTF \g\_unravel_ends_int > \c_zero
2635             {
2636                 \_unravel_back_input:
2637                 \_unravel_back_input:n
2638                 {
2639                     \tex_hbox:D to \tex_hsize:D { }
2640                     \tex_vfill:D
2641                     \tex_penalty:D - '10000000000 ~
2642                 }
2643                 \_unravel_build_page:
2644                 \_unravel_print_action:x { End-everything! }
2645             }
2646             {
2647                 \_unravel_print_outcome:
2648                 \l\_unravel_head_token
2649             }
2650         }
2651     }
2652 }
2653 \_unravel_new_tex_cmd:nn { delim_num }                             % 15
2654 {

```

```

2655 \__unravel_mode_math:n
2656 {
2657   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2658   \__unravel_print_action:
2659   \__unravel_scan_int:
2660   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2661   \tl_use:N \l__unravel_head_tl \scan_stop:
2662   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2663 }
2664 }

```

### 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.

```

2665 \__unravel_new_tex_cmd:nn { char_num } % 16
2666 {
2667   \__unravel_mode_non_vertical:n

```

```

2668     {
2669       \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2670       \__unravel_print_action:
2671       \__unravel_scan_int:
2672       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2673       \mode_if_math:TF
2674         { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
2675         { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
2676     }
2677 }

```

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.

```

2678 \__unravel_new_tex_cmd:nn { math_char_num } % 17
2679 {
2680   \__unravel_mode_math:n
2681   {
2682     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2683     \__unravel_print_action:
2684     \__unravel_scan_int:
2685     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2686     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
2687   }
2688 }

```

```

2689 \__unravel_new_tex_cmd:nn { mark } % 18
2690 {
2691   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2692   \__unravel_print_action:
2693   \int_compare:nNnF \l__unravel_head_char_int = \c_zero
2694     { \__unravel_scan_int: }
2695   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2696   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2697   \seq_gpop_right:Nn \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2698   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2699   \__unravel_print_action:x
2700     { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
2701   \tl_put_right:Nx \l__unravel_head_tl
2702     { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2703   \tl_use:N \l__unravel_head_tl
2704 }

```

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

```

2705 \__unravel_new_tex_cmd:nn { xray } % 19

```

```

2706 {
2707   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2708   \__unravel_print_action:
2709   \int_case:nnF \l__unravel_head_char_int
2710   {
2711     { 0 }
2712     { % show
2713       \__unravel_get_next:
2714       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2715       \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl
2716     }
2717     { 2 }
2718     { % showthe
2719       \__unravel_get_x_next:
2720       \__unravel_scan_something_internal:n { 5 }
2721       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2722       \exp_args:Nx \etex_showtokens:D
2723         { \tl_tail:N \l__unravel_head_tl }
2724     }
2725   }
2726   { % no operand for showlists, showgroups, showifs
2727     \int_compare:nNnT \l__unravel_head_char_int = \c_one % showbox
2728     { \__unravel_scan_int: }
2729     \int_compare:nNnT \l__unravel_head_char_int = \c_five % showtokens
2730     { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
2731     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2732     \tl_use:N \l__unravel_head_tl \scan_stop:
2733   }
2734 }

```

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

```

2735 \__unravel_new_tex_cmd:nn { make_box } % 20
2736 {
2737   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
2738   \__unravel_back_input:
2739   \__unravel_do_box:N \c_false_bool
2740 }

```

\\_\_unravel\_do\_move: Scan a dimension and a box, and perform the shift, printing the appropriate action.

```

2741 \cs_new_protected_nopar:Npn \__unravel_do_move:
2742 {
2743   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2744   \__unravel_print_action:
2745   \__unravel_scan_normal_dimen:
2746   \__unravel_do_box:N \c_false_bool
2747 }

```

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

hmove=21 for \moveright and \moveleft.

```

2748 \__unravel_new_tex_cmd:nn { hmove } % 21

```

```

2749 {
2750   \mode_if_vertical:TF
2751   { \__unravel_do_move: } { \__unravel_forbidden_case: }
2752 }
    vmove=22 for \lower and \raise.
2753 \__unravel_new_tex_cmd:nn { vmove } % 22
2754 {
2755   \mode_if_vertical:TF
2756   { \__unravel_forbidden_case: } { \__unravel_do_move: }
2757 }

```

`\__unravel_do_unpackage:`

```

2758 \cs_new_protected_nopar:Npn \__unravel_do_unpackage:
2759 {
2760   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2761   \__unravel_print_action:
2762   \__unravel_scan_int:
2763   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2764   \tl_use:N \l__unravel_head_tl \scan_stop:
2765   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2766 }

```

*(End definition for \\_\_unravel\_do\_unpackage:.)*

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

```

2767 \__unravel_new_tex_cmd:nn { un_hbox } % 23
2768 { \__unravel_mode_non_vertical:n { \__unravel_do_unpackage: } }

```

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

```

2769 \__unravel_new_tex_cmd:nn { un_vbox } % 24

```

```

2770 {
2771   \__unravel_mode_vertical:n
2772   {
2773     \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
2774     { \l__unravel_head_token \__unravel_print_action: }
2775     { \__unravel_do_unpackage: }
2776   }
2777 }

```

`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.

```

2778 \__unravel_new_tex_cmd:nn { remove_item } % 25
2779 { \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.

```

2780 \cs_new_protected_nopar:Npn \__unravel_do_append_glue:

```

```

2781 {
2782   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
2783     { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
2784     {
2785       \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2786       \__unravel_print_action:
2787       \exp_args:Nf \__unravel_scan_glue:n
2788         { \int_eval:n { \l__unravel_head_char_int - 2 } }
2789       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2790       \tl_use:N \l__unravel_head_tl \scan_stop:
2791       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2792     }
2793 }

```

(End definition for \\_\_unravel\_do\_append\_glue:.)

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

```

2794 \__unravel_new_tex_cmd:nn { hskip } % 26
2795 { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }
      vskip=27 for \vfil, \vfill, \vss, \vfilneg, \vskip.
2796 \__unravel_new_tex_cmd:nn { vskip } % 27
2797 { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }
      mskip=28 for \mskip (5).
2798 \__unravel_new_tex_cmd:nn { mskip } % 28
2799 { \__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.

```

2800 \cs_new_protected_nopar:Npn \__unravel_do_append_kern:
2801 {
2802   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2803   \__unravel_print_action:
2804   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
2805     { \__unravel_scan_dimen:NN \c_true_bool \c_false_bool }
2806     { \__unravel_scan_dimen:NN \c_false_bool \c_false_bool }
2807   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2808   \tl_use:N \l__unravel_head_tl \scan_stop:
2809   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2810 }

```

(End definition for \\_\_unravel\_do\_append\_kern:.)

kern=29 for \kern (1).

```

2811 \__unravel_new_tex_cmd:nn { kern } % 29
2812 { \__unravel_do_append_kern: }
      mkern=30 for \mkern (99).
2813 \__unravel_new_tex_cmd:nn { mkern } % 30
2814 { \__unravel_mode_math:n { \__unravel_do_append_kern: } }

```

```

leader_ship=31 for \shipout (99), \leaders (100), \cleaders (101), \xleaders (102).
2815 \__unravel_new_tex_cmd:nn { leader_ship } % 31
2816 {
2817   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2818   \__unravel_print_action:
2819   \__unravel_do_box:N \c_true_bool
2820 }

```

### 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

```

2821 \__unravel_new_tex_cmd:nn { halign } % 32
2822 { \msg_fatal:nxx { unravel } { not-implemented } { halign } }
2823 \__unravel_new_tex_cmd:nn { valign } % 33
2824 { \msg_fatal:nxx { unravel } { not-implemented } { valign } }
2825 \__unravel_new_tex_cmd:nn { no_align } % 34
2826 { \msg_fatal:nxx { unravel } { not-implemented } { noalign } }

```

```

2827 \__unravel_new_tex_cmd:nn { vrule } % 35
2828 { \__unravel_mode_non_vertical:n { \__unravel_do_rule: } }
2829 \__unravel_new_tex_cmd:nn { hrule } % 36
2830 { \__unravel_mode_vertical:n { \__unravel_do_rule: } }
2831 \cs_new_protected_nopar:Npn \__unravel_do_rule:
2832 {
2833   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2834   \__unravel_print_action:
2835   \__unravel_scan_alt_rule:
2836   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2837   \tl_use:N \l__unravel_head_tl \scan_stop:
2838   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2839 }

2840 \__unravel_new_tex_cmd:nn { insert } % 37
2841 { \__unravel_begin_insert_or_adjust: }
2842 \__unravel_new_tex_cmd:nn { vadjust } % 38
2843 {
2844   \mode_if_vertical:TF
2845   { \__unravel_forbidden_case: } { \__unravel_begin_insert_or_adjust: }
2846 }

2847 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
2848 {
2849   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
2850   {
2851     \__unravel_get_x_non_blank:
2852     \__unravel_set_cmd:
2853     \__unravel_do_step:
2854   }
2855   { \msg_error:nn { unravel } { not-implemented } { pdfprimitive } }
2856 }

2857 \__unravel_new_tex_cmd:nn { after_assignment } % 40
2858 {
2859   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
2860   \__unravel_get_next:
2861   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
2862   \__unravel_print_action:x
2863   {
2864     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
2865     \gtl_to_str:N \l__unravel_head_gtl
2866   }
2867 }

2868 \__unravel_new_tex_cmd:nn { after_group } % 41
2869 { \msg_error:nxx { unravel } { not-implemented } { aftergroup } }

See \__unravel_do_append_glue:.

2870 \__unravel_new_tex_cmd:nn { break_penalty } % 42
2871 {
2872   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl

```



```

2873 \__unravel_print_action:
2874 \__unravel_scan_int:
2875 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2876 \tl_use:N \l__unravel_head_tl \scan_stop:
2877 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2878 }

2879 \__unravel_new_tex_cmd:nn { start_par } % 43
2880 {
2881 \mode_if_vertical:TF
2882 {
2883 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
2884 { \__unravel_new_graf:N \c_false_bool }
2885 { \__unravel_new_graf:N \c_true_bool }
2886 }
2887 {
2888 \int_compare:nNnF \l__unravel_head_char_int = { 1 } % indent
2889 {
2890 \tex_hbox:D width \tex_parindent:D { }
2891 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2892 }
2893 \__unravel_print_action:
2894 }
2895 }

2896 \__unravel_new_tex_cmd:nn { ital_corr } % 44
2897 {
2898 \mode_if_vertical:TF { \__unravel_forbidden_case: }
2899 { \l__unravel_head_token \__unravel_print_action: }
2900 }

\__unravel_do_accent:

2901 \cs_new_protected_nopar:Npn \__unravel_do_accent:
2902 {
2903 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2904 \__unravel_print_action:
2905 \__unravel_scan_int:
2906 \__unravel_do_assignments:
2907 \bool_if:nTF
2908 {
2909 \token_if_eq_catcode_p:NN
2910 \l__unravel_head_token \c_catcode_letter_token
2911 ||
2912 \token_if_eq_catcode_p:NN
2913 \l__unravel_head_token \c_catcode_other_token
2914 ||
2915 \int_compare_p:nNn
2916 \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
2917 }
2918 { \__unravel_prev_input:V \l__unravel_head_tl }
2919 {

```

```

2920     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
2921     {
2922         \__unravel_prev_input:V \l__unravel_head_tl
2923         \__unravel_scan_int:
2924     }
2925     { \__unravel_break:w }
2926 }
2927 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2928 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2929 \tl_use:N \l__unravel_head_tl \scan_stop:
2930 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2931 \__unravel_break_point:
2932 }

```

(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).

```

2933 \cs_new_protected_nopar:Npn \__unravel_do_math_accent:
2934 {
2935     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2936     \__unravel_print_action:
2937     \__unravel_scan_int:
2938     \__unravel_scan_math:
2939     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2940     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2941     \tl_use:N \l__unravel_head_tl \scan_stop:
2942     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2943 }

```

(End definition for `\__unravel_do_math_accent:`)

```

2944 \__unravel_new_tex_cmd:nn { accent } % 45
2945 {
2946     \__unravel_mode_non_vertical:n
2947     {
2948         \mode_if_math:TF
2949         { \__unravel_do_math_accent: } { \__unravel_do_accent: }
2950     }
2951 }
2952 \__unravel_new_tex_cmd:nn { math_accent } % 46
2953 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
2954 \__unravel_new_tex_cmd:nn { discretionary } % 47
2955 { \msg_error:nxx { unravel } { not-implemented } { discretionary } }

```

#### 2.13.4 Maths: from 48 to 56

- eq\_no=48
- left\_right=49
- math\_comp=50

- limit\_switch=51
- above=52
- math\_style=53
- math\_choice=54
- non\_script=55
- vcenter=56

```

2956 \__unravel_new_tex_cmd:nn { eq_no } % 48
2957 { \msg_error:nxx { unravel } { not-implemented } { eqno } }
2958 \__unravel_new_tex_cmd:nn { left_right } % 49
2959 { \msg_error:nxx { unravel } { not-implemented } { left/right } }
2960 \__unravel_new_tex_cmd:nn { math_comp } % 50
2961 { \msg_error:nxx { unravel } { not-implemented } { math-comp } }
2962 \__unravel_new_tex_cmd:nn { limit_switch } % 51
2963 { \msg_error:nxx { unravel } { not-implemented } { limits } }
2964 \__unravel_new_tex_cmd:nn { above } % 52
2965 { \msg_error:nxx { unravel } { not-implemented } { above } }
2966 \__unravel_new_tex_cmd:nn { math_style } % 53
2967 { \msg_error:nxx { unravel } { not-implemented } { math-style } }
2968 \__unravel_new_tex_cmd:nn { math_choice } % 54
2969 { \msg_error:nxx { unravel } { not-implemented } { math-choice } }
2970 \__unravel_new_tex_cmd:nn { non_script } % 55
2971 { \msg_error:nxx { unravel } { not-implemented } { non-script } }
2972 \__unravel_new_tex_cmd:nn { vcenter } % 56
2973 { \msg_error:nxx { unravel } { not-implemented } { vcenter } }

```

### 2.13.5 From 57 to 70

- case\_shift=57
- message=58
- extension=59
- in\_stream=60
- begin\_group=61
- end\_group=62
- omit=63
- ex\_space=64

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

```

2974 \__unravel_new_tex_cmd:nn { case_shift } % 57
2975 {
2976   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2977   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
2978   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
2979   \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
2980 }
2981 \cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
2982 {
2983   #1 { \__unravel_back_input:n {#2} }
2984   \__unravel_print_action:x
2985     { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
2986 }
2987 \__unravel_new_tex_cmd:nn { message } % 58
2988 {
2989   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
2990   \__unravel_print_action:
2991   \__unravel_scan_toks_to_str:
2992   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
2993   \tl_use:N \l__unravel_head_tl
2994   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2995 }
2996
2997 Extensions are implemented in a later section.
2998
2999 \__unravel_new_tex_cmd:nn { extension } % 59
3000 {
3001   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3002   \__unravel_print_action:
3003   \__unravel_scan_extension_operands:
3004   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3005   \tl_use:N \l__unravel_head_tl \scan_stop:
3006   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3007 }
3008
3009 \__unravel_new_tex_cmd:nn { in_stream } % 60
3010 {
3011   \seq_put_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3012   \__unravel_print_action:
3013   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D

```

```

3010     {
3011         \__unravel_scan_int:
3012         \__unravel_scan_optional_equals:
3013         \__unravel_scan_file_name:
3014     }
3015     { \__unravel_scan_int: }
3016     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3017     \tl_use:N \l__unravel_head_tl \scan_stop:
3018     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3019 }
3020 \__unravel_new_tex_cmd:nn { begin_group } % 61
3021 {
3022     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3023     \__unravel_print_action:
3024     \l__unravel_head_token
3025 }
3026 \__unravel_new_tex_cmd:nn { end_group } % 62
3027 {
3028     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3029     \__unravel_print_action:
3030     \l__unravel_head_token
3031 }
3032 \__unravel_new_tex_cmd:nn { omit } % 63
3033 { \msg_error:nn { unravel } { not-implemented } { omit } }
3034 \__unravel_new_tex_cmd:nn { ex_space } % 64
3035 {
3036     \__unravel_mode_non_vertical:n
3037     { \l__unravel_head_token \__unravel_print_action: }
3038 }
3039 \__unravel_new_tex_cmd:nn { no_boundary } % 65
3040 {
3041     \__unravel_mode_non_vertical:n
3042     { \l__unravel_head_token \__unravel_print_action: }
3043 }
3044 \__unravel_new_tex_cmd:nn { radical } % 66
3045 {
3046     \__unravel_mode_math:n
3047     {
3048         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3049         \__unravel_print_action:
3050         \__unravel_scan_int:
3051         \__unravel_scan_math:
3052         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3053         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3054         \tl_use:N \l__unravel_head_tl \scan_stop:
3055         \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3056     }
3057 }

```

```

    Let TEX cause the error.
3058 \__unravel_new_tex_cmd:nn { end_cs_name } % 67
3059 { \l__unravel_head_token \__unravel_print_action: }

    See the_char and other_char.
3060 \__unravel_new_tex_cmd:nn { char_given } % 68
3061 {
3062   \__unravel_mode_non_vertical:n
3063   {
3064     \mode_if_math:TF
3065     { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
3066     { \__unravel_char:V \l__unravel_head_char_int }
3067   }
3068 }

    See math_char_num.
3069 \__unravel_new_tex_cmd:nn { math_given } % 69
3070 {
3071   \__unravel_mode_math:n
3072   { \__unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3073 }

3074 \__unravel_new_tex_cmd:nn { last_item } % 70
3075 { \__unravel_forbidden_case: }

```

### 2.13.6 Extensions

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

```

3076 \cs_new_protected_nopar:Npn \__unravel_scan_extension_operands:
3077 {
3078   \int_case:nnF \l__unravel_head_char_int
3079   {
3080     { 0 } % openout
3081     {
3082       \__unravel_scan_int:
3083       \__unravel_scan_optional_equals:
3084       \__unravel_scan_file_name:
3085     }
3086     { 1 } % write
3087     {
3088       \__unravel_scan_int:
3089       \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3090     }
3091     { 2 } % closeout
3092     { \__unravel_scan_int: }
3093     { 3 } % special
3094     { \__unravel_scan_toks_to_str: }
3095     { 4 } % immediate
3096     { \__unravel_scan_immediate_operands: }
3097     { 5 } % setlanguage

```

```

3098     {
3099         \mode_if_horizontal:TF
3100         { \_unravel_scan_int: }
3101         { \msg_error:nn { unravel } { invalid-mode } }
3102     }
3103 { 6 } % pdfliteral
3104     {
3105         \_unravel_scan_keyword:nF { dD iI rR eE cC tT }
3106         { \_unravel_scan_keyword:n { pP aA gG eE } }
3107         \_unravel_scan_pdf_ext_toks:
3108     }
3109 { 7 } % pdfobj
3110     {
3111         \_unravel_scan_keyword:nTF
3112         { rR eE sS eE rR vV eE oO bB jJ nN uU mM }
3113         { \_unravel_skip_optional_space: }
3114         {
3115             \_unravel_scan_keyword:nF { uU sS eE oO bB jJ nN uU mM }
3116             { \_unravel_scan_int: }
3117             \_unravel_scan_keyword:nT { sS tT rR eE aA mM }
3118             {
3119                 \_unravel_scan_keyword:nT { aA tT tT rR }
3120                 { \_unravel_scan_pdf_ext_toks: }
3121             }
3122             \_unravel_scan_keyword:n { fF iI lL eE }
3123             \_unravel_scan_pdf_ext_toks:
3124         }
3125     }
3126 { 8 } % pdfrefobj
3127     { \_unravel_scan_int: }
3128 { 9 } % pdfxform
3129     {
3130         \_unravel_scan_keyword:nT { aA tT tT rR }
3131         { \_unravel_scan_pdf_ext_toks: }
3132         \_unravel_scan_keyword:nTF { rR eE sS oO uU rR cC eE sS }
3133         { \_unravel_scan_pdf_ext_toks: }
3134         \_unravel_scan_int:
3135     }
3136 { 10 } % pdfrefxform
3137     { \_unravel_scan_int: }
3138 { 11 } % pdfximage
3139     { \_unravel_scan_image: }
3140 { 12 } % pdfrefximage
3141     { \_unravel_scan_int: }
3142 { 13 } % pdfannot
3143     {
3144         \_unravel_scan_keyword:nTF
3145         { rR eE sS eE rR vV eE oO bB jJ nN uU mM }
3146         { \_unravel_scan_optional_space: }
3147         {

```

```

3148         \_unravel_scan_keyword:nT { uU sS eE oO bB jJ nN uU mM }
3149         { \_unravel_scan_int: }
3150         \_unravel_scan_alt_rule:
3151         \_unravel_scan_pdf_ext_toks:
3152     }
3153 }
3154 { 14 } % pdfstartlink
3155 {
3156     \mode_if_vertical:TF
3157     { \msg_error:nn { unravel } { invalid-mode } }
3158     {
3159         \_unravel_scan_rule_attr:
3160         \_unravel_scan_action:
3161     }
3162 }
3163 { 15 } % pdfendlink
3164 {
3165     \mode_if_vertical:T
3166     { \msg_error:nn { unravel } { invalid-mode } }
3167 }
3168 { 16 } % pdfoutline
3169 {
3170     \_unravel_scan_keyword:nT { aA tT tT rR }
3171     { \_unravel_scan_pdf_ext_toks: }
3172     \_unravel_scan_action:
3173     \_unravel_scan_keyword:nT { cC oO uU nN tT }
3174     { \_unravel_scan_int: }
3175     \_unravel_scan_pdf_ext_toks:
3176 }
3177 { 17 } % pdfdest
3178 { \_unravel_scan_pdfdest_operands: }
3179 { 18 } % pdfthread
3180 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3181 { 19 } % pdfstartthread
3182 { \_unravel_scan_rule_attr: \_unravel_scan_thread_id: }
3183 { 20 } % pdfendthread
3184 { }
3185 { 21 } % pdfsavepos
3186 { }
3187 { 22 } % pdfinfo
3188 { \_unravel_scan_pdf_ext_toks: }
3189 { 23 } % pdfcatalog
3190 {
3191     \_unravel_scan_pdf_ext_toks:
3192     \_unravel_scan_keyword:n { oO pP eE nN aA cC tT iI oO nN }
3193     { \_unravel_scan_action: }
3194 }
3195 { 24 } % pdfnames
3196 { \_unravel_scan_pdf_ext_toks: }
3197 { 25 } % pdffontattr

```



```

3198     {
3199         \_unravel_scan_font_ident:
3200         \_unravel_scan_pdf_ext_toks:
3201     }
3202 { 26 } % pdfincludechars
3203     {
3204         \_unravel_scan_font_ident:
3205         \_unravel_scan_pdf_ext_toks:
3206     }
3207 { 27 } % pdfmapfile
3208     { \_unravel_scan_pdf_ext_toks: }
3209 { 28 } % pdfmapline
3210     { \_unravel_scan_pdf_ext_toks: }
3211 { 29 } % pdftrailer
3212     { \_unravel_scan_pdf_ext_toks: }
3213 { 30 } % pdfresettimer
3214     { }
3215 { 31 } % pdffontexpand
3216     {
3217         \_unravel_scan_font_ident:
3218         \_unravel_scan_optional_equals:
3219         \_unravel_scan_int:
3220         \_unravel_scan_int:
3221         \_unravel_scan_int:
3222         \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
3223         { \_unravel_skip_optional_space: }
3224     }
3225 { 32 } % pdfsetrandomseed
3226     { \_unravel_scan_int: }
3227 { 33 } % pdfsnaprefpoint
3228     { }
3229 { 34 } % pdfsnapy
3230     { \_unravel_scan_normal_glue: }
3231 { 35 } % pdfsnapycomp
3232     { \_unravel_scan_int: }
3233 { 36 } % pdfglyphtounicode
3234     {
3235         \_unravel_scan_pdf_ext_toks:
3236         \_unravel_scan_pdf_ext_toks:
3237     }
3238 { 37 } % pdfcolorstack
3239     { \_unravel_scan_pdfcolorstack_operands: }
3240 { 38 } % pdfsetmatrix
3241     { \_unravel_scan_pdf_ext_toks: }
3242 { 39 } % pdfsave
3243     { }
3244 { 40 } % pdfrestore
3245     { }
3246 { 41 } % pdfnobluiltintounicode
3247     { \_unravel_scan_font_ident: }

```

```

3248     }
3249     { } % no other cases.
3250 }
(End definition for \_unravel_scan_extension_operands:.)

```

\\_unravel\_scan\_pdfcolorstack\_operands:

```

3251 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstack_operands:
3252 {
3253   \_unravel_scan_int:
3254   \_unravel_scan_keyword:nF { sSeEtT }
3255   {
3256     \_unravel_scan_keyword:nF { pPuUsShH }
3257     {
3258       \_unravel_scan_keyword:nF { pPoOpP }
3259       {
3260         \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
3261         {
3262           \msg_error:nn { unravel }
3263           { color-stack-action-missing }
3264         }
3265       }
3266     }
3267   }
3268 }
(End definition for \_unravel_scan_pdfcolorstack_operands:.)

```

\\_unravel\_scan\_rule\_attr:

```

3269 \cs_new_protected_nopar:Npn \_unravel_scan_rule_attr:
3270 {
3271   \_unravel_scan_alt_rule:
3272   \_unravel_scan_keyword:nT { aA tT tT rR }
3273   { \_unravel_scan_pdf_ext_toks: }
3274 }
(End definition for \_unravel_scan_rule_attr:.)

```

\\_unravel\_scan\_action:

```

3275 \cs_new_protected_nopar:Npn \_unravel_scan_action:
3276 {
3277   \_unravel_scan_keyword:nTF { uUsSeErR }
3278   { \_unravel_scan_pdf_ext_toks: }
3279   {
3280     \_unravel_scan_keyword:nF { gGoOtToO }
3281     {
3282       \_unravel_scan_keyword:nF { tThHrReEaAdD }
3283       { \msg_error:nn { unravel } { action-type-missing } }
3284     }
3285   }
3286   \_unravel_scan_keyword:nT { fFiIlLeE }
3287   { \_unravel_scan_pdf_ext_toks: }

```

```

3288 \__unravel_scan_keyword:nTF { pPaAgGeE }
3289 {
3290   \__unravel_scan_int:
3291   \__unravel_scan_pdf_ext_toks:
3292 }
3293 {
3294   \__unravel_scan_keyword:nTF { nNaAmMeE }
3295   { \__unravel_scan_pdf_ext_toks: }
3296   {
3297     \__unravel_scan_keyword:nTF { nNuUmM }
3298     { \__unravel_scan_int: }
3299     { \msg_error:nn { unravel } { identifier-type-missing } }
3300   }
3301 }
3302 \__unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
3303 { \__unravel_skip_optional_space: }
3304 {
3305   \__unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
3306   { \__unravel_skip_optional_space: }
3307 }
3308 }

```

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

`\__unravel_scan_image:` Used by `\pdfximage`.

```

3309 \cs_new_protected_nopar:Npn \__unravel_scan_image:
3310 {
3311   \__unravel_scan_rule_attr:
3312   \__unravel_scan_keyword:nTF { nNaAmMeEdD }
3313   { \__unravel_scan_pdf_ext_toks: }
3314   {
3315     \__unravel_scan_keyword:nT { pPaAgGeE }
3316     { \__unravel_scan_int: }
3317   }
3318   \__unravel_scan_keyword:nT { cCoO1LoOrRsSpPaAcCeE }
3319   { \__unravel_scan_int: }
3320   \__unravel_scan_pdf_ext_toks:
3321 }

```

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

`\__unravel_scan_immediate_operands:`

```

3322 \cs_new_protected_nopar:Npn \__unravel_scan_immediate_operands:
3323 {
3324   \__unravel_get_x_next:
3325   \__unravel_set_cmd:
3326   \int_compare:nNnTF
3327   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
3328   {
3329     \int_compare:nNnTF
3330     \l__unravel_head_char_int < { 3 } % openout, write, closeout

```

```

3331     { \_unravel_scan_immediate_operands_aux: }
3332     {
3333         \int_case:nnF \l__unravel_head_char_int
3334         {
3335             { 7 } { \_unravel_scan_extension_operands_aux: } % pdfobj
3336             { 9 } { \_unravel_scan_extension_operands_aux: } % pdfxform
3337             { 11 } { \_unravel_scan_extension_operands_aux: } %pdfximage
3338         }
3339         { \_unravel_scan_immediate_operands_bad: }
3340     }
3341 }
3342 { \_unravel_scan_immediate_operands_bad: }
3343 }
3344 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_aux:
3345 {
3346     \_unravel_prev_input:V \l__unravel_head_tl
3347     \_unravel_scan_extension_operands:
3348 }
3349 \cs_new_protected_nopar:Npn \_unravel_scan_immediate_operands_bad:
3350 {
3351     \_unravel_back_input:
3352     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3353     \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
3354     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3355 }
3356

```

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

\\_unravel\_scan\_pdfdest\_operands:

```

3357 \cs_new_protected_nopar:Npn \_unravel_scan_pdfdest_operands:
3358 {
3359     \_unravel_scan_keyword:nTF { nNuUmM }
3360     { \_unravel_scan_int: }
3361     {
3362         \_unravel_scan_keyword:nTF { nNaAmMeE }
3363         { \_unravel_scan_pdf_ext_toks: }
3364         { \msg_error:nn { unravel } { identifier-type-missing } }
3365     }
3366     \_unravel_scan_keyword:nTF { xXyYzZ }
3367     {
3368         \_unravel_scan_keyword:nTF { zZoOoOmM }
3369         { \_unravel_scan_int: }
3370     }
3371     {
3372         \_unravel_scan_keyword:nF { fFiItTbBhH }
3373         {
3374             \_unravel_scan_keyword:nF { fFiItTbBvV }
3375             {
3376                 \_unravel_scan_keyword:nF { fFiItTbB }
3377                 {

```

```

3378     \_unravel_scan_keyword:nF { fFiItThHhH }
3379     {
3380         \_unravel_scan_keyword:nF { fFiItTvV }
3381         {
3382             \_unravel_scan_keyword:nTF
3383             { fFiItTrR }
3384             {
3385                 \_unravel_skip_optional_space:
3386                 \_unravel_scan_alt_rule:
3387                 \use_none:n
3388             }
3389             {
3390                 \_unravel_scan_keyword:nF
3391                 { fFiItT }
3392                 {
3393                     \msg_error:nn { unravel }
3394                     {
3395                         destination-type-missing
3396                     }
3397                 }
3398             }
3399         }
3400     }
3401 }
3402 }
3403 }
3404 }
3405 \_unravel_skip_optional_space:
3406 }

```

(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`.

```

3407 \cs_set_protected_nopar:Npn \_unravel_tmp:w
3408 {
3409     \seq_gput_right:Nn \g_unravel_prev_input_seq { }
3410     \_unravel_prefixed_command:
3411 }
3412 \int_step_inline:nnnn
3413 { \_unravel_tex_use:n { max_non_prefixed_command } + 1 }
3414 { 1 }
3415 { \_unravel_tex_use:n { max_command } }
3416 { \cs_new_eq:cN { \_unravel_cmd_#1: } \_unravel_tmp:w }

```

`\__unravel_prefixed_command:` Accumulated prefix codes so far are stored as the last item of `\g__unravel_prev_input_seq`.

```

3417 \cs_new_protected_nopar:Npn \__unravel_prefixed_command:
3418 {
3419   \int_while_do:nNnn
3420     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
3421     {
3422       \__unravel_prev_input:V \l__unravel_head_tl
3423       \__unravel_get_x_non_relax:
3424       \__unravel_set_cmd:
3425       \int_compare:nNnF \l__unravel_head_cmd_int
3426         > { \__unravel_tex_use:n { max_non_prefixed_command } }
3427         {
3428           \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3429           \msg_error:nnxx { unravel } { erroneous-prefixes }
3430             { \tl_to_str:N \l__unravel_tmpa_tl }
3431             { \tl_to_str:N \l__unravel_head_tl }
3432           \__unravel_back_input:
3433           \__unravel_omit_after_assignment:w
3434         }
3435       }
3436       % ^^A todo: Discard non-\global prefixes if they are irrelevant
3437       % ^^A todo: Adjust for the setting of \globaldefs
3438       \cs_if_exist_use:cF
3439         { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
3440         {
3441           \msg_error:nnx { unravel } { internal } { prefixed }
3442           \__unravel_omit_after_assignment:w
3443         }
3444       \__unravel_after_assignment:
3445     }

```

*(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 3446 \cs_new_protected_nopar:Npn \__unravel_after_assignment:
3447 {
3448   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
3449   \gtl_gclear:N \g__unravel_after_assignment_gtl
3450 }
3451 \cs_new_protected_nopar:Npn \__unravel_omit_after_assignment:w
3452   #1 \__unravel_after_assignment: { }

```

*(End definition for `\__unravel_after_assignment:`. This function is documented on page ??.)*

`\__unravel_prefixed_new:nn`

```

3453 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
3454 {

```

```

3455 \cs_new_protected_nopar:cpn
3456   { __unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
3457 }
(End definition for \__unravel_prefixed_new:nn.)

```

\\_\_unravel\_assign\_token:n

```

3458 \cs_new_protected:Npn \__unravel_assign_token:n #1
3459   {
3460     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3461     #1
3462     \tl_use:N \l__unravel_head_tl \scan_stop:
3463     \__unravel_print_assigned_token:
3464   }
(End definition for \__unravel_assign_token:n.)

```

\\_\_unravel\_assign\_register:

```

3465 \cs_new_protected_nopar:Npn \__unravel_assign_register:
3466   {
3467     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3468     \tl_use:N \l__unravel_head_tl \scan_stop:
3469     \__unravel_print_assigned_register:
3470   }
(End definition for \__unravel_assign_register:.)

```

\\_\_unravel\_assign\_value:nn

```

3471 \cs_new_protected:Npn \__unravel_assign_value:nn #1#2
3472   {
3473     \tl_if_empty:nF {#1}
3474     {
3475       \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3476       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3477       #1
3478       \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3479     }
3480     \__unravel_prev_input:V \l__unravel_head_tl
3481     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3482     \__unravel_scan_optional_equals:
3483     #2
3484     \__unravel_assign_register:
3485   }
(End definition for \__unravel_assign_value:nn.)

```

\\_\_unravel\_assign\_toks:

```

3486 \__unravel_prefixed_new:nn { toks_register } % 71
3487   {
3488     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3489     { % \toks
3490       \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3491       \__unravel_print_action:

```

```

3492     \_unravel_scan_int:
3493     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3494   }
3495   \_unravel_assign_toks:
3496 }
3497 \_unravel_prefixed_new:nn { assign_toks } % 72
3498 { \_unravel_assign_toks: }
3499 \cs_new_protected_nopar:Npn \_unravel_assign_toks:
3500 {
3501   \_unravel_prev_input_silent:V \l__unravel_head_tl
3502   \_unravel_print_action:
3503   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3504   \_unravel_scan_optional_equals:
3505   \_unravel_get_x_non_relax:
3506   \_unravel_set_cmd:
3507   \int_compare:nNnTF
3508     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { toks_register } }
3509   {
3510     \_unravel_prev_input:V \l__unravel_head_tl
3511     \int_compare:nNnT \l__unravel_head_char_int = \c_zero
3512     { \_unravel_scan_int: }
3513   }
3514   {
3515     \int_compare:nNnTF
3516       \l__unravel_head_cmd_int = { \_unravel_tex_use:n { assign_toks } }
3517       { \_unravel_prev_input:V \l__unravel_head_tl }
3518     {
3519       \_unravel_back_input:
3520       \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3521     }
3522   }
3523   \_unravel_assign_register:
3524 }
(End definition for \_unravel_assign_toks:.)
3525 \_unravel_prefixed_new:nn { assign_int } % 73
3526 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
3527 \_unravel_prefixed_new:nn { assign_dimen } % 74
3528 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
3529 \_unravel_prefixed_new:nn { assign_glue } % 75
3530 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_glue: } }
3531 \_unravel_prefixed_new:nn { assign_mu_glue } % 76
3532 { \_unravel_assign_value:nn { } { \_unravel_scan_mu_glue: } }
3533 \_unravel_prefixed_new:nn { assign_font_dimen } % 77
3534 {
3535   \_unravel_assign_value:nn
3536     { \_unravel_scan_int: \_unravel_scan_font_ident: }
3537     { \_unravel_scan_normal_dimen: }
3538 }
3539 \_unravel_prefixed_new:nn { assign_font_int } % 78

```



```

3540 {
3541   \__unravel_assign_value:nn
3542   { \__unravel_scan_font_int: } { \__unravel_scan_int: }
3543 }
3544 \__unravel_prefixed_new:nn { set_aux } % 79
3545 { % prevdepth = 1, spacefactor = 0
3546   \int_compare:nNnTF \l__unravel_head_char_int = \c_one
3547   { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3548   { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3549 }
3550 \__unravel_prefixed_new:nn { set_prev_graf } % 80
3551 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3552 \__unravel_prefixed_new:nn { set_page_dimen } % 81
3553 { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
3554 \__unravel_prefixed_new:nn { set_page_int } % 82
3555 { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
3556 \__unravel_prefixed_new:nn { set_box_dimen } % 83
3557 {
3558   \__unravel_assign_value:nn
3559   { \__unravel_scan_int: } { \__unravel_scan_normal_dimen: }
3560 }
3561 \__unravel_prefixed_new:nn { set_shape } % 84
3562 {
3563   \__unravel_assign_value:nn { \__unravel_scan_int: }
3564   {
3565     \prg_replicate:nn
3566     {
3567       \tl_if_head_eq_meaning:VNT
3568       \l__unravel_defined_tl \tex_parshape:D { \c_two * }
3569       \tl_tail:N \l__unravel_defined_tl
3570     }
3571     { \__unravel_scan_int: }
3572   }
3573 }
3574 \__unravel_prefixed_new:nn { def_code } % 85
3575 {
3576   \__unravel_assign_value:nn
3577   { \__unravel_scan_int: } { \__unravel_scan_int: }
3578 }
3579 \__unravel_prefixed_new:nn { def_family } % 86
3580 {
3581   \__unravel_assign_value:nn
3582   { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
3583 }
3584 \__unravel_prefixed_new:nn { set_font } % 87
3585 {
3586   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3587   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3588   \tl_use:N \l__unravel_head_tl \scan_stop:

```

```

3589 \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3590 \__unravel_print_action:
3591 }
3592 \__unravel_prefixed_new:nn { def_font } % 88
3593 {
3594 \__unravel_prev_input_silent:V \l__unravel_head_tl
3595 \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
3596 \__unravel_scan_r_token:
3597 \__unravel_print_action:x
3598 { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
3599 \__unravel_scan_optional_equals:
3600 \__unravel_scan_file_name:
3601 \bool_gset_true:N \g__unravel_name_in_progress_bool
3602 \__unravel_scan_keyword:nTF { aAtT }
3603 { \__unravel_scan_normal_dimen: }
3604 {
3605 \__unravel_scan_keyword:nT { sS cC aA lL eE dD }
3606 { \__unravel_scan_int: }
3607 }
3608 \bool_gset_false:N \g__unravel_name_in_progress_bool
3609 \__unravel_assign_token:n { }
3610 }

```

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

let, futurelet

```

3611 \__unravel_prefixed_new:nn { let } % 94
3612 {
3613 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3614 \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
3615 { % |let|
3616 \__unravel_scan_r_token:
3617 \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3618 \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3619 \__unravel_get_next:
3620 \bool_while_do:nn
3621 { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
3622 { \__unravel_get_next: }
3623 \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
3624 { \__unravel_get_next: }
3625 \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
3626 { \__unravel_get_next: }
3627 }
3628 { % |futurelet|
3629 \__unravel_scan_r_token:
3630 \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3631 \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3632 \__unravel_get_next:
3633 \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
3634 \__unravel_get_next:

```

```

3635     \__unravel_back_input:
3636     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
3637     \__unravel_back_input:
3638   }
3639   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3640   \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
3641   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3642   \use:x
3643   {
3644     \exp_not:V \l__unravel_head_tl
3645     \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
3646   }
3647   \__unravel_print_assigned_token:
3648 }

3649 \__unravel_prefixed_new:nn { shorthand_def } % 95
3650 {
3651   \__unravel_prev_input_silent:V \l__unravel_head_tl
3652   \tl_set:Nx \l__unravel_prev_action_tl
3653     { \tl_to_str:N \l__unravel_head_tl }
3654   \__unravel_scan_r_token:
3655   \__unravel_print_action:x
3656     { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
3657   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
3658   \__unravel_scan_optional_equals:
3659   \__unravel_scan_int:
3660   \__unravel_assign_token:n { }
3661 }

3662 \__unravel_prefixed_new:nn { read_to_cs } % 96
3663 {
3664   \__unravel_prev_input:V \l__unravel_head_tl
3665   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3666   \__unravel_scan_int:
3667   \__unravel_scan_keyword:nF { tTo0 }
3668   {
3669     \msg_error:nn { unravel } { missing-to }
3670     \__unravel_prev_input:n { to }
3671   }
3672   \__unravel_scan_r_token:
3673   \__unravel_assign_token:n { }
3674 }

3675 \__unravel_prefixed_new:nn { def } % 97
3676 {
3677   \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3678   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
3679   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
3680   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3681   \int_compare:nNnTF \l__unravel_head_char_int < \c_two
3682     { % def/gdef
3683       \__unravel_scan_r_token:

```

```

3684     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
3685     \__unravel_scan_toks:NN \c_true_bool \c_false_bool
3686   }
3687   { % edef/xdef
3688     \__unravel_scan_r_token:
3689     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
3690     \__unravel_scan_toks:NN \c_true_bool \c_true_bool
3691   }
3692   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3693   \__unravel_prev_input:V \l__unravel_head_tl
3694   \__unravel_assign_token:n
3695   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
3696 }

```

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

```

3697 \__unravel_prefixed_new:nn { set_box } % 98
3698 {
3699   \__unravel_prev_input:V \l__unravel_head_tl
3700   \__unravel_scan_int:
3701   \__unravel_scan_optional_equals:
3702   \bool_if:NTF \g__unravel_set_box_allowed_bool
3703   { \__unravel_do_box:N \c_false_bool }
3704   {
3705     \msg_error:nn { unravel } { improper-setbox }
3706     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3707     \__unravel_omit_after_assignment:w
3708   }
3709 }

```

\hyphenation and \patterns

```

3710 \__unravel_prefixed_new:nn { hyph_data } % 99
3711 {
3712   \__unravel_prev_input:V \l__unravel_head_tl
3713   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3714   \__unravel_assign_token:n { }
3715 }
3716 \__unravel_prefixed_new:nn { set_interaction } % 100
3717 {
3718   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3719   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
3720   \tl_use:N \l__unravel_head_tl \scan_stop:
3721   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3722 }
3723 \__unravel_prefixed_new:nn { letterspace_font } % 101
3724 {
3725   % ^^A todo...
3726   % new_letterspaced_font(a);

```

```

3727 \msg_error:nxx { unravel } { not-implemented } { letterspace-font }
3728 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3729 \__unravel_omit_after_assignment:w
3730 }
3731 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
3732 {
3733 % ^^A todo...
3734 % make_font_copy(a);
3735 \msg_error:nxx { unravel } { not-implemented } { pdf-copy-font }
3736 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3737 \__unravel_omit_after_assignment:w
3738 }

```

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

```

3739 \__unravel_prefixed_new:nn { register } % 89
3740 { \__unravel_do_register:N \c_zero }
3741 \__unravel_prefixed_new:nn { advance } % 90
3742 { \__unravel_do_operation:N \c_one }
3743 \__unravel_prefixed_new:nn { multiply } % 91
3744 { \__unravel_do_operation:N \c_two }
3745 \__unravel_prefixed_new:nn { divide } % 92
3746 { \__unravel_do_operation:N \c_three }

```

```

\__unravel_do_operation:N
  \__unravel_do_operation_fail:w

```

```

3747 \cs_new_protected:Npn \__unravel_do_operation:N #1
3748 {
3749   \__unravel_prev_input_silent:V \l__unravel_head_tl
3750   \__unravel_print_action:
3751   \__unravel_get_x_next:
3752   \__unravel_set_cmd:
3753   \int_compare:nNnTF
3754     \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
3755     {
3756       \int_compare:nNnTF
3757         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
3758         { \__unravel_do_register:N #1 }
3759         { \__unravel_do_operation_fail:w }
3760     }
3761     {
3762       \int_compare:nNnTF
3763         \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
3764         { \__unravel_do_operation_fail:w }
3765         {
3766           \__unravel_prev_input:V \l__unravel_head_tl
3767           \exp_args:NNf \__unravel_do_register_set:Nn #1
3768           {
3769             \int_eval:n
3770             {
3771               \l__unravel_head_cmd_int

```

```

3772         - \_unravel_tex_use:n { assign_toks }
3773     }
3774 }
3775 }
3776 }
3777 }
3778 \cs_new_protected_nopar:Npn \_unravel_do_operation_fail:w
3779 {
3780     \msg_error:nn { unravel } { after-advance }
3781     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3782     \_unravel_omit_after_assignment:w
3783 }

```

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

\\_unravel\_do\_register:N

\\_unravel\_do\_register\_aux:Nn

```

3784 \cs_new_protected:Npn \_unravel_do_register:N #1
3785 {
3786     \exp_args:NNV \_unravel_do_register_aux:Nn #1
3787     \l__unravel_head_char_int
3788 }
3789 \cs_new_protected:Npn \_unravel_do_register_aux:Nn #1#2
3790 {
3791     \int_compare:nNnTF { \tl_tail:n {#2} } = \c_zero
3792     {
3793         \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3794         \_unravel_print_action:
3795         \_unravel_scan_int:
3796         \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3797         \_unravel_prev_input_silent:V \l__unravel_head_tl
3798     }
3799     {
3800         \_unravel_prev_input_silent:V \l__unravel_head_tl
3801         \_unravel_print_action:
3802     }
3803     \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
3804     \exp_args:NNf \_unravel_do_register_set:Nn #1
3805     { \int_eval:n { #2 / 1 000 000 } }
3806 }

```

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

\\_unravel\_do\_register\_set:Nn

```

3807 \cs_new_protected:Npn \_unravel_do_register_set:Nn #1#2
3808 {
3809     \int_compare:nNnTF {#1} = \c_zero
3810     { % truly register command
3811         \_unravel_scan_optional_equals:
3812     }
3813     { % \advance, \multiply, \divide
3814         \_unravel_scan_keyword:nF { bByY }

```

```

3815     { \_unravel_prev_input_silent:n { by } }
3816   }
3817   \int_compare:nNnTF {#1} < \c_two
3818   {
3819     \int_case:nnF {#2}
3820     {
3821       { 1 } { \_unravel_scan_int:          } % count
3822       { 2 } { \_unravel_scan_normal_dimen: } % dim
3823       { 3 } { \_unravel_scan_normal_glue:  } % glue
3824       { 4 } { \_unravel_scan_mu_glue:     } % muglue
3825     }
3826     { \msg_error:nnx { unravel } { internal } { do-reg=#2 } }
3827   }
3828   { \_unravel_scan_int: }
3829   \_unravel_assign_register:
3830 }

```

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

The following is used for instance when making accents.

```

3831 \cs_new_protected_nopar:Npn \_unravel_do_assignments:
3832 {
3833   \_unravel_get_x_non_relax:
3834   \_unravel_set_cmd:
3835   \int_compare:nNnT
3836   \l__unravel_head_cmd_int
3837   > { \_unravel_tex_use:n { max_non_prefixed_command } }
3838   {
3839     \bool_gset_false:N \g__unravel_set_box_allowed_bool
3840     \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3841     \_unravel_prefixed_command:
3842     \bool_gset_true:N \g__unravel_set_box_allowed_bool
3843     \_unravel_do_assignments:
3844   }
3845 }

```

## 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.

```
3846 \__unravel_new_tex_expandable:nn { undefined_cs } % 103
3847 { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
```

```
\__unravel_expandafter:
  \__unravel_unless: 3848 \__unravel_new_tex_expandable:nn { expand_after } % 104
  \__unravel_unless_bad: 3849 {
3850   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
3851   { \__unravel_expandafter: } { \__unravel_unless: }
3852 }
3853 \cs_new_protected_nopar:Npn \__unravel_expandafter:
3854 {
3855   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
3856   \__unravel_get_next:
3857   \gtl_concat:NNN \l__unravel_head_gtl
3858   \l__unravel_tmpb_gtl \l__unravel_head_gtl
3859   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_gtl
3860   \__unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
3861   \__unravel_get_next:
3862   \__unravel_token_if_expandable:NTF \l__unravel_head_token
3863   { \__unravel_expand: }
3864   { \__unravel_back_input: }
3865   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_gtl
3866   \__unravel_set_action_text:x
3867   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
3868   \gtl_pop_left:N \l__unravel_head_gtl
3869   \__unravel_back_input:
3870   \__unravel_print_action:
3871 }
```



```

3872 \cs_new_protected_nopar:Npn \__unravel_unless:
3873 {
3874   \__unravel_get_token:
3875   \int_compare:nNnTF
3876     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
3877     {
3878       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
3879       { \__unravel_unless_bad: }
3880       {
3881         \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
3882         % \int_add:Nn \l__unravel_head_char_int { 32 }
3883         \__unravel_expand_nonmacro:
3884       }
3885     }
3886     { \__unravel_unless_bad: }
3887   }
3888 \cs_new_protected_nopar:Npn \__unravel_unless_bad:
3889 {
3890   \msg_error:nn { unravel } { bad-unless }
3891   \__unravel_back_input:
3892 }
(End definition for \__unravel_expandafter:, \__unravel_unless:, and \__unravel_unless_bad:.)

\__unravel_noexpand:
\__unravel_pdfprimitive:
3893 \__unravel_new_tex_expandable:nn { no_expand } % 105
3894 {
3895   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
3896   { \__unravel_noexpand: }
3897   { \__unravel_pdfprimitive: }
3898 }
3899 \cs_new_protected_nopar:Npn \__unravel_noexpand:
3900 {
3901   \__unravel_get_token:
3902   \__unravel_back_input:
3903   \__unravel_token_if_expandable:NT \l__unravel_head_token
3904   {
3905     \cs_gset_protected_nopar:Npx \__unravel_get_next:
3906     {
3907       \cs_gset_protected_nopar:Npn \__unravel_get_next:
3908       { \exp_not:o { \__unravel_get_next: } }
3909       \exp_not:o { \__unravel_get_next: }
3910       \exp_not:n { \cs_set_eq:NN \l__unravel_head_token \tex_relax:D }
3911     }
3912   }
3913 }
3914 \cs_new_protected_nopar:Npn \__unravel_pdfprimitive:
3915 { \msg_error:nnx { unravel } { not-implemented } { pdfprimitive } }
(End definition for \__unravel_noexpand: and \__unravel_pdfprimitive:.)

```

```

\__unravel_scantokens:
  \__unravel_input:
3916 \__unravel_new_tex_expandable:nn { input } % 106
3917 {
3918   \int_case:nnF \l__unravel_head_char_int
3919   {
3920     { 1 } { \__unravel_print_action: } % \endinput
3921     { 2 } { \__unravel_scantokens: } % \scantokens
3922   }
3923   { % 0=\input
3924     \bool_if:NTF \g__unravel_name_in_progress_bool
3925     { \__unravel_insert_relax: } { \__unravel_input: }
3926   }
3927 }
3928 \cs_new_protected_nopar:Npn \__unravel_scantokens:
3929 {
3930   \seq_gput_right:Nn \g__unravel_prev_input_seq { }
3931   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3932   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
3933   \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
3934   \__unravel_back_input:V \l__unravel_head_tl
3935   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
3936 }
3937 \cs_new_protected_nopar:Npn \__unravel_input:
3938 {
3939   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3940   \__unravel_scan_file_name:
3941   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3942   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
3943   \__unravel_tl_gset_input:Nno \g__unravel_tmpe_tl { } \l__unravel_tmpa_tl
3944   \__unravel_back_input:V \g__unravel_tmpe_tl
3945   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3946 }
(End definition for \__unravel_scantokens: and \__unravel_input:.)

```

```

\__unravel_csname_loop:
3947 \__unravel_new_tex_expandable:nn { cs_name } % 109
3948 {
3949   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3950   \__unravel_print_action:
3951   \__unravel_csname_loop:
3952   \__unravel_prev_input:V \l__unravel_head_tl
3953   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
3954   \__unravel_back_input_tl_o:
3955 }
3956 \cs_new_protected_nopar:Npn \__unravel_csname_loop:
3957 {
3958   \__unravel_get_x_next:
3959   \token_if_cs:NTF \l__unravel_head_token
3960   {

```

```

3961     \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
3962     {
3963         \msg_error:nn { unravel } { missing-endcsname }
3964         \__unravel_back_input:
3965         \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
3966     }
3967 }
3968 {
3969     \__unravel_prev_input_silent:x
3970     { \__unravel_token_to_char:N \l__unravel_head_token }
3971     \__unravel_csname_loop:
3972 }
3973 }
(End definition for \__unravel_csname_loop:.)
3974 \__unravel_new_tex_expandable:nn { convert } % 110
3975 {
3976     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
3977     \__unravel_print_action:
3978     \int_case:nn \l__unravel_head_char_int
3979     {
3980         0     \__unravel_scan_int:
3981         1     \__unravel_scan_int:
3982         2 { \__unravel_get_next: \__unravel_prev_input:V \l__unravel_head_tl }
3983         3 { \__unravel_get_next: \__unravel_prev_input:V \l__unravel_head_tl }
3984         4     \__unravel_scan_font_ident:
3985         8     \__unravel_scan_font_ident:
3986         9     \__unravel_scan_font_ident:
3987         { 10 } \__unravel_scan_font_ident:
3988         { 11 } \__unravel_scan_int:
3989         { 12 } \__unravel_scan_int:
3990         { 13 } \__unravel_scan_pdf_ext_toks:
3991         { 14 } \__unravel_scan_pdf_ext_toks:
3992         { 15 } \__unravel_scan_int:
3993         { 16 } \__unravel_scan_int:
3994         { 17 } \__unravel_scan_pdfstrcmp:
3995         { 18 } \__unravel_scan_pdfcolorstackinit:
3996         { 19 } \__unravel_scan_pdf_ext_toks:
3997         { 20 } \__unravel_scan_pdf_ext_toks:
3998         { 22 } \__unravel_scan_pdf_ext_toks:
3999         { 23 } \__unravel_scan_pdf_ext_toks:
4000         { 24 }
4001         {
4002             \__unravel_scan_keyword:n { fF iI lL eE }
4003             \__unravel_scan_pdf_ext_toks:
4004         }
4005         { 25 } \__unravel_scan_pdffiledump:
4006         { 26 } \__unravel_scan_pdfmatch:
4007         { 27 } \__unravel_scan_int:
4008         { 28 } \__unravel_scan_int:

```

```

4009         { 30 } \_unravel_scan_int:
4010         { 31 } \_unravel_scan_pdfimagebbox:
4011     }
4012     \seq_gpop_right:NN \g\_unravel_prev_input_seq \l\_unravel_head_tl
4013     \_unravel_back_input_tl_o:
4014 }
4015 \cs_new_protected_nopar:Npn \_unravel_scan_pdfstrcmp:
4016 {
4017     \_unravel_scan_toks_to_str:
4018     \_unravel_scan_toks_to_str:
4019 }
4020 \cs_new_protected_nopar:Npn \_unravel_scan_pdfimagebbox:
4021 { \_unravel_scan_int: \_unravel_scan_int: }
4022 \cs_new_protected_nopar:Npn \_unravel_scan_pdfcolorstackinit:
4023 {
4024     \_unravel_scan_keyword:nTF { pP aA gG eE }
4025     { \bool_set_true:N \l\_unravel_tmpa_bool }
4026     { \bool_set_false:N \l\_unravel_tmpb_bool }
4027     \_unravel_scan_keyword:nF { dD iI rR eE cC tT }
4028     { \_unravel_scan_keyword:n { pP aA gG eE } }
4029     \_unravel_scan_toks_to_str:
4030 }
4031 \cs_new_protected_nopar:Npn \_unravel_scan_pdffiledump:
4032 {
4033     \_unravel_scan_keyword:nT { oO fF fF sS eE tT } \_unravel_scan_int:
4034     \_unravel_scan_keyword:nT { lL eE nN gG tT hH } \_unravel_scan_int:
4035     \_unravel_scan_pdf_ext_toks:
4036 }
4037 \cs_new_protected_nopar:Npn \_unravel_scan_pdfmatch:
4038 {
4039     \_unravel_scan_keyword:n { iI cC aA sS eE }
4040     \_unravel_scan_keyword:nT { sS uU bB cC oO uU nN tT }
4041     { \_unravel_scan_int: }
4042     \_unravel_scan_pdf_ext_toks:
4043     \_unravel_scan_pdf_ext_toks:
4044 }

```

\\_unravel\_get\_the:

```

4045 \_unravel_new_tex_expandable:nn { the } % 111
4046 {
4047     \_unravel_get_the:
4048     \tl_set:Nx \l\_unravel_tmpa_tl { \exp_args:NV \exp_not:o \l\_unravel_head_tl }
4049     \_unravel_back_input:V \l\_unravel_tmpa_tl
4050     \_unravel_print_action:
4051 }
4052 \cs_new_protected_nopar:Npn \_unravel_get_the:
4053 {
4054     \seq_gput_right:NV \g\_unravel_prev_input_seq \l\_unravel_head_tl
4055     \_unravel_print_action:
4056     \int_if_odd:nTF \l\_unravel_head_char_int

```

```

4057 { % \unexpanded, \detokenize
4058   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4059   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4060   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4061 }
4062 { % \the
4063   \__unravel_get_x_next:
4064   \__unravel_scan_something_internal:n { 5 }
4065   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4066   \__unravel_set_action_text:x
4067   {
4068     \tl_head:N \l__unravel_head_tl
4069     => \tl_tail:N \l__unravel_head_tl
4070   }
4071   \tl_set:Nx \l__unravel_head_tl
4072   { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
4073 }
4074 }
(End definition for \__unravel_get_the:.)
4075 \__unravel_new_tex_expandable:nn { top_bot_mark } % 112
4076 { \__unravel_back_input_tl_o: }
4077 \__unravel_new_tex_expandable:nn { end_template } % 117
4078 {
4079   \msg_error:nn { unravel } { not-implemented } { end-template }
4080   \__unravel_back_input_tl_o:
4081 }

```

### 2.14.1 Conditionals

```

\__unravel_pass_text:
\__unravel_pass_text_done:w
4082 \cs_new_protected_nopar:Npn \__unravel_pass_text:
4083 {
4084   \__unravel_input_if_empty:TF
4085   { \__unravel_pass_text_empty: }
4086   {
4087     \__unravel_input_get:N \l__unravel_tmpb_gtl
4088     \if_true:
4089       \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one
4090       \exp_after:wN \__unravel_pass_text_done:w
4091       \fi:
4092       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4093       \exp_after:wN \__unravel_pass_text:
4094     \else:
4095       \use:c { fi: }
4096       \int_set_eq:NN \l__unravel_if_nesting_int \c_one
4097       \__unravel_input_gpop:N \l__unravel_tmpb_gtl
4098       \exp_after:wN \__unravel_pass_text_nested:
4099     \fi:

```

```

4100     }
4101   }
4102 \cs_new_protected_nopar:Npn \__unravel_pass_text_done:w
4103   {
4104     \__unravel_get_next:
4105     \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
4106     \else:
4107   }

```

(End definition for \\_\_unravel\_pass\_text:. This function is documented on page ??.)

\\_\_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:..

```

4108 \cs_new_protected_nopar:Npn \__unravel_pass_text_nested:
4109   {
4110     \__unravel_input_if_empty:TF
4111     { \__unravel_pass_text_empty: }
4112     {
4113       \__unravel_input_get:N \l__unravel_tmpb_gtl
4114       \if_true:
4115         \if_true:
4116         \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
4117         \int_decr:N \l__unravel_if_nesting_int
4118         \use_none:nnnnn
4119         \fi:
4120         \use_none:nnn
4121       \fi:
4122       \int_incr:N \l__unravel_if_nesting_int
4123       \fi:
4124       \__unravel_input_gpop:N \l__unravel_tmpa_gtl
4125       \int_compare:nNnTF \l__unravel_if_nesting_int = \c_zero
4126       { \__unravel_pass_text: }
4127       { \__unravel_pass_text_nested: }
4128     }
4129   }

```

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

\\_\_unravel\_pass\_text\_empty:

```

4130 \cs_new_protected_nopar:Npn \__unravel_pass_text_empty:

```

```

4131 {
4132   \msg_error:nn { unravel } { runaway-if }
4133   \__unravel_exit:w
4134 }

```

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

```

\__unravel_cond_push:
\__unravel_cond_pop:

```

```

4135 \cs_new_protected:Npn \__unravel_cond_push:
4136 {
4137   \tl_gput_left:Nx \g__unravel_if_limit_tl
4138   { { \int_use:N \g__unravel_if_limit_int } }
4139   \int_gincr:N \g__unravel_if_depth_int
4140   \int_gzero:N \g__unravel_if_limit_int
4141 }
4142 \cs_new_protected_nopar:Npn \__unravel_cond_pop:
4143 {
4144   \int_gset:Nn \g__unravel_if_limit_int
4145   { \tl_head:N \g__unravel_if_limit_tl }
4146   \tl_gset:Nx \g__unravel_if_limit_tl
4147   { \tl_tail:N \g__unravel_if_limit_tl }
4148   \int_gdecr:N \g__unravel_if_depth_int
4149 }

```

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

```

\__unravel_change_if_limit:nn

```

```

4150 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
4151 {
4152   \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
4153   { \int_gset:Nn \g__unravel_if_limit_int {#1} }
4154   {
4155     \tl_clear:N \l__unravel_tmpa_tl
4156     \prg_replicate:nn { \g__unravel_if_depth_int - #2 - \c_one }
4157     {
4158       \tl_put_right:Nx \l__unravel_tmpa_tl
4159       { { \tl_head:N \g__unravel_if_limit_tl } }
4160       \tl_gset:Nx \g__unravel_if_limit_tl
4161       { \tl_tail:N \g__unravel_if_limit_tl }
4162     }
4163     \tl_gset:Nx \g__unravel_if_limit_tl
4164     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
4165   }
4166 }

```

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

```

4167 \__unravel_new_tex_expandable:nn { if_test } % 107
4168 {
4169   \__unravel_cond_push:
4170   \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
4171 }

```

\\_unravel\_cond\_aux:nn

```
4172 \cs_new_protected:Npn \_unravel_cond_aux:n #1
4173 {
4174   \int_case:nnF \l__unravel_head_char_int
4175   {
4176     { 12 } { \_unravel_test_ifx:n {#1} }
4177     { 16 } { \_unravel_test_case:n {#1} }
4178     { 21 } { \_unravel_test_pdfprimitive:n {#1} } % ^^A todo and \unless
4179   }
4180   {
4181     \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4182     \_unravel_print_action:
4183     \int_case:nn \l__unravel_head_char_int
4184     {
4185       { 0 } { \_unravel_test_two_chars: } % if
4186       { 1 } { \_unravel_test_two_chars: } % ifcat
4187       { 2 } % ifnum
4188         { \_unravel_test_two_vals:N \_unravel_scan_int: }
4189       { 3 } % ifdim
4190         { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4191       { 4 } { \_unravel_scan_int: } % ifodd
4192       % { 5 } { } % ifvmode
4193       % { 6 } { } % ifhmode
4194       % { 7 } { } % ifmmode
4195       % { 8 } { } % ifinner
4196       { 9 } { \_unravel_scan_int: } % ifvoid
4197       { 10 } { \_unravel_scan_int: } % ifhbox
4198       { 11 } { \_unravel_scan_int: } % ifvbox
4199       { 13 } { \_unravel_scan_int: } % ifeof
4200       % { 14 } { } % iftrue
4201       % { 15 } { } % iffalse
4202       { 17 } { \_unravel_test_ifdefined: } % ifdefined
4203       { 18 } { \_unravel_test_ifcsname: } % ifcsname
4204       { 19 } % iffontchar
4205         { \_unravel_scan_font_ident: \_unravel_scan_int: }
4206       % { 20 } { } % ifincsname % ^^A todo: something?
4207       { 22 } % ifpdfabsnum
4208         { \_unravel_test_two_vals:N \_unravel_scan_int: }
4209       { 23 } % ifpdfabsdim
4210         { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
4211     }
4212     \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4213     \_unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4214     \l__unravel_head_tl \scan_stop:
4215     \exp_after:wN \_unravel_cond_true:n
4216   \else:
4217     \exp_after:wN \_unravel_cond_false:n
4218   \fi:
4219   {#1}
```



```

4220     }
4221   }
(End definition for \_unravel_cond_aux:nn.)

```

\\_unravel\_cond\_true:n

```

4222 \cs_new_protected:Npn \_unravel_cond_true:n #1
4223   {
4224     \_unravel_change_if_limit:nn { 3 } {#1} % wait for else/fi
4225     \_unravel_print_action:x { \g_unravel_action_text_str = true }
4226   }
(End definition for \_unravel_cond_true:n.)

```

\\_unravel\_cond\_false:n

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

```

4227 \cs_new_protected:Npn \_unravel_cond_false:n #1
4228   {
4229     \_unravel_cond_false_loop:n {#1}
4230     \_unravel_cond_false_common:
4231     \_unravel_print_action:x { \g_unravel_action_text_str = false }
4232   }
4233 \cs_new_protected:Npn \_unravel_cond_false_loop:n #1
4234   {
4235     \_unravel_pass_text:
4236     \int_compare:nNnTF \g_unravel_if_depth_int = {#1}
4237       {
4238         \token_if_eq_meaning:NNT \l_unravel_head_token \or:
4239         {
4240           \msg_error:nn { unravel } { extra-or }
4241           \_unravel_cond_false_loop:n {#1}
4242         }
4243       }
4244       {
4245         \token_if_eq_meaning:NNT \l_unravel_head_token \fi:
4246         { \_unravel_cond_pop: }
4247         \_unravel_cond_false_loop:n {#1}
4248       }
4249   }
4250 \cs_new_protected_nopar:Npn \_unravel_cond_false_common:
4251   {
4252     \token_if_eq_meaning:NNTF \l_unravel_head_token \fi:
4253     { \_unravel_cond_pop: }
4254     { \int_gset:Nn \g_unravel_if_limit_int { 2 } } % wait for fi
4255   }

```

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

\\_unravel\_test\_two\_vals:N

```

4256 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
4257   {
4258     #1
4259     \_unravel_get_x_non_blank:

```

```

4260 \tl_if_in:nVF { < = > } \l__unravel_head_tl
4261 {
4262   \msg_error:nn { unravel } { missing-equals }
4263   \__unravel_back_input:
4264   \tl_set:Nn \l__unravel_head_tl { = }
4265 }
4266 \__unravel_prev_input:V \l__unravel_head_tl
4267 #1
4268 }

```

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

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

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

```

4269 \cs_new_protected_nopar:Npn \__unravel_test_two_chars:
4270 {
4271   \__unravel_test_two_chars_aux:
4272   \__unravel_prev_input:V \l__unravel_head_tl
4273   \__unravel_test_two_chars_aux:
4274   \__unravel_prev_input:V \l__unravel_head_tl
4275 }
4276 \cs_new_protected_nopar:Npn \__unravel_test_two_chars_aux:
4277 {
4278   \__unravel_get_x_next:
4279   \gtl_if_tl:NF \l__unravel_head_gtl
4280   {
4281     \tl_set:Nx \l__unravel_head_tl
4282     {
4283       \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
4284       { \c_group_begin_token } { \c_group_end_token }
4285     }
4286   }
4287   \tl_put_left:Nn \l__unravel_head_tl { \exp_not:N } % ^^A todo: prettify.
4288 }

```

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

\\_\_unravel\_test\_ifx:n

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

```

4289 \cs_new_protected:Npn \__unravel_test_ifx:n #1
4290 {
4291   \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4292   \__unravel_print_action:
4293   \__unravel_get_next:
4294   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4295   \__unravel_get_next:
4296   \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_tmpa_tl
4297   \__unravel_set_action_text:x
4298   {
4299     Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
4300     \gtl_to_str:N \l__unravel_tmpb_gtl
4301     \gtl_to_str:N \l__unravel_head_gtl
4302   }

```

```

4303 \gtl_head_do:NN \l__unravel_tmpb_gtl \__unravel_test_ifx_aux:w
4304 \exp_after:wN \__unravel_cond_true:n
4305 \else:
4306 \exp_after:wN \__unravel_cond_false:n
4307 \fi:
4308 {#1}
4309 }
4310 \cs_new_nopar:Npn \__unravel_test_ifx_aux:w
4311 { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tpa_tl }
(End definition for \__unravel_test_ifx:n and \__unravel_test_ifx_aux:w.)

```

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn

```

```

4312 \cs_new_protected:Npn \__unravel_test_case:n #1
4313 {
4314 \seq_gput_right:NV \g__unravel_prev_input_seq \l__unravel_head_tl
4315 \__unravel_print_action:
4316 \bool_if:NT \l__unravel_debug_bool { \iow_term:n { {\ifcase level~#1} } }
4317 \__unravel_scan_int:
4318 \seq_get_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4319 \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
4320 % ^^A does text_case_aux use prev_input_seq?
4321 \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
4322 \seq_gpop_right:NN \g__unravel_prev_input_seq \l__unravel_head_tl
4323 \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4324 }
4325 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
4326 {
4327 \int_compare:nNnTF {#1} = \c_zero
4328 { \__unravel_change_if_limit:nn { 4 } {#2} }
4329 {
4330 \__unravel_pass_text:
4331 \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
4332 {
4333 \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
4334 {
4335 \exp_args:Nf \__unravel_test_case_aux:nn
4336 { \int_eval:n { #1 - 1 } } {#2}
4337 }
4338 { \__unravel_cond_false_common: }
4339 }
4340 {
4341 \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
4342 { \__unravel_cond_pop: }
4343 \__unravel_test_case_aux:nn {#1} {#2}
4344 }
4345 }
4346 }

```

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

\\_unravel\_test\_ifdefined:

```
4347 \cs_new_protected_nopar:Npn \_unravel_test_ifdefined:
4348 {
4349   \_unravel_input_if_empty:TF
4350   { \_unravel_pass_text_empty: }
4351   {
4352     \_unravel_input_gpop:N \l__unravel_tmpb_gtl
4353     \_unravel_set_action_text:x
4354     {
4355       Conditional:~ \tl_to_str:N \l__unravel_head_tl
4356       \gtl_to_str:N \l__unravel_tmpb_gtl
4357     }
4358     \_unravel_prev_input:x
4359     {
4360       \gtl_if_tl:NTF \l__unravel_tmpb_gtl
4361       { \gtl_head:N \l__unravel_tmpb_gtl }
4362       { \gtl_to_str:N \l__unravel_tmpb_gtl }
4363     }
4364   }
4365 }
```

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

\\_unravel\_test\_ifcname:

```
4366 \cs_new_protected_nopar:Npn \_unravel_test_ifcname:
4367 {
4368   \_unravel_csname_loop:
4369   \_unravel_prev_input:V \l__unravel_head_tl
4370 }
```

*(End definition for \\_unravel\_test\_ifcname:.)*

```
4371 \_unravel_new_tex_expandable:nn { fi_or_else } % 108
4372 {
4373   \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
4374   {
4375     \int_compare:nNnTF \g__unravel_if_limit_int = \c_zero
4376     {
4377       \int_compare:nNnTF \g__unravel_if_depth_int = \c_zero
4378       { \msg_error:nn { unravel } { extra-fi-or-else } }
4379       { \_unravel_insert_relax: }
4380     }
4381     { \msg_error:nn { unravel } { extra-fi-or-else } }
4382   }
4383   {
4384     \_unravel_set_action_text:
4385     \int_compare:nNnF \l__unravel_head_char_int = \c_two
4386     {
4387       \_unravel-fi-or-else-loop:
4388       \_unravel_set_action_text:x
4389       {
4390         \g__unravel_action_text_str \c_space_tl
```

```

4391         => ~ skipped ~ to ~ \tl_to_str:N \l__unravel_head_tl
4392     }
4393 }
4394 % ^^A todo: in this print_action the token itself is missing.
4395 \__unravel_print_action:
4396 \__unravel_cond_pop:
4397 }
4398 }
4399 \cs_new_protected_nopar:Npn \__unravel_fi_or_else_loop:
4400 {
4401     \int_compare:nNnF \l__unravel_head_char_int = \c_two
4402     {
4403         \__unravel_pass_text:
4404         \__unravel_set_cmd:
4405         \__unravel_fi_or_else_loop:
4406     }
4407 }

```

## 2.15 User interaction

### 2.15.1 Print

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

```

\__unravel_print:n
\__unravel_print:x

```

```

4408 \cs_new_eq:NN \__unravel_print:n \iow_term:n
4409 \cs_generate_variant:Nn \__unravel_print:n { x }

```

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

```

\__unravel_print_message:nn

```

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

```

4410 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
4411 { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_print:n }

```

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

```

\__unravel_set_action_text:x

```

```

4412 \cs_new_protected:Npn \__unravel_set_action_text:x #1
4413 {
4414     \group_begin:
4415     \int_set:Nn \tex_escapechar:D { 92 }
4416     \str_gset:Nx \g__unravel_action_text_str {#1}
4417     \group_end:
4418 }

```

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

`\__unravel_set_action_text:`

```
4419 \cs_new_protected_nopar:Npn \__unravel_set_action_text:
4420 {
4421   \__unravel_set_action_text:x
4422   {
4423     \tl_to_str:N \l__unravel_head_tl
4424     \tl_if_single_token:VT \l__unravel_head_tl
4425     { = ~ \exp_after:wN \token_to_meaning:N \l__unravel_head_tl }
4426   }
4427 }
```

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

`\__unravel_print_state:`

```
4428 \cs_new_protected:Npn \__unravel_print_state:
4429 {
4430   \group_begin:
4431     \int_set:Nn \tex_escapechar:D { 92 }
4432     \int_compare:nNnT \g__unravel_noise_int > \c_zero
4433     {
4434       \exp_args:Nx \__unravel_print_state_output:n
4435       { \gtl_to_str:N \g__unravel_output_gtl }
4436       \seq_set_map:Nn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
4437       { \__unravel_to_str:n {##1} }
4438       \seq_remove_all:Nn \l__unravel_tmpa_seq { }
4439       \exp_args:Nx \__unravel_print_state_prev:n
4440       { \seq_use:Nn \l__unravel_tmpa_seq { \ } }
4441       \exp_args:Nx \__unravel_print_state_input:n
4442       { \__unravel_input_to_str: }
4443     }
4444   \group_end:
4445   \__unravel_prompt:
4446 }
```

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

`\__unravel_print_state_output:n`

```
4447 \cs_new_protected:Npn \__unravel_print_state_output:n #1
4448 {
4449   \tl_if_empty:nF {#1}
4450   {
4451     \int_set:Nn \l__unravel_print_int { \str_count:n {#1} }
4452     \__unravel_print_message:nn { <| ~ }
4453     {
4454       \int_compare:nNnTF
4455       \l__unravel_print_int > \g__unravel_max_output_int
4456       {
4457         (
4458           \int_eval:n
4459           {
4460             \l__unravel_print_int
```

```

4461         - \g__unravel_max_output_int + 14
4462     } ~
4463     chars
4464 )~
4465 ...
4466 \str_substr:nnn {#1}
4467 { \l__unravel_print_int - \g__unravel_max_output_int + 15 }
4468 { \l__unravel_print_int }
4469 }
4470 {#1}
4471 }
4472 }
4473 }

```

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

\\_\_unravel\_print\_state\_prev:n

```

4474 \cs_new_protected:Npn \__unravel_print_state_prev:n #1
4475 {
4476   % \int_set:Nn \l__unravel_print_int { \str_count:n {#1} }
4477   \__unravel_print_message:nn { || ~ } {#1}
4478   % {
4479   %   \int_compare:nNnTF \l__unravel_print_int > \g__unravel_max_prev_int
4480   %   {
4481   %     (
4482   %       \int_eval:n
4483   %       { \l__unravel_print_int - \g__unravel_max_prev_int + 14 } ~
4484   %       chars
4485   %     )~
4486   %     ...
4487   %     \str_substr:nnn {#1}
4488   %     { \l__unravel_print_int - \g__unravel_max_prev_int + 15 }
4489   %     { \l__unravel_print_int }
4490   %   }
4491   %   {#1}
4492   % }
4493 }

```

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

\\_\_unravel\_print\_state\_input:n

```

4494 \cs_new_protected:Npn \__unravel_print_state_input:n #1
4495 {
4496   \int_set:Nn \l__unravel_print_int { \str_count:n {#1} }
4497   \__unravel_print_message:nn { |> ~ }
4498   {
4499     \int_compare:nNnTF \l__unravel_print_int > \g__unravel_max_input_int
4500     {
4501       \str_substr:nnn {#1} { 1 } { \g__unravel_max_input_int - 14 }
4502       ...~
4503       (

```

```

4504         \int_eval:n
4505         { \l__unravel_print_int - \g__unravel_max_input_int + 14 } ~
4506         chars
4507     )
4508 }
4509 {#1}
4510 }
4511 }

```

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

\\_\_unravel\_print\_meaning:

```

4512 \cs_new_protected:Npn \__unravel_print_meaning:
4513 {
4514     \__unravel_input_if_empty:TF
4515     { \__unravel_print_message:nn { } { Empty~input! } }
4516     {
4517         \__unravel_input_get:N \l__unravel_tmpb_gtl
4518         \__unravel_print_message:nn { }
4519         {
4520             \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
4521             = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
4522         }
4523     }
4524 }

```

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

\\_\_unravel\_print\_action:

\\_\_unravel\_print\_action:x

```

4525 \cs_new_protected:Npn \__unravel_print_action:
4526 {
4527     \int_gincr:N \g__unravel_step_int
4528     \__unravel_print_message:nn { }
4529     {
4530         % \
4531         [====~ Step~ \int_use:N \g__unravel_step_int \ =====]~
4532         \int_compare:nNnTF
4533         { \str_count:N \g__unravel_action_text_str }
4534         > { \g__unravel_max_action_int }
4535         {
4536             \str_substr:Nnn \g__unravel_action_text_str
4537             { 1 } { \g__unravel_max_action_int - 3 } ...
4538         }
4539         { \g__unravel_action_text_str }
4540         % \
4541         % \ < \int_use:N \g__unravel_input_int > % ^^A todo: remove
4542         % \ < \seq_count:N \g__unravel_prev_input_seq > % ^^A todo: remove
4543     }
4544     \__unravel_print_state:
4545 }
4546 \cs_new_protected:Npn \__unravel_print_action:x #1

```



```

4547 {
4548   \__unravel_set_action_text:x {#1}
4549   \__unravel_print_action:
4550 }
(End definition for \__unravel_print_action: and \__unravel_print_action:x.)

```

\\_unravel\_print\_gtl\_action:N

```

4551 \cs_new_protected:Npn \__unravel_print_gtl_action:N #1
4552 {
4553   \__unravel_print_action:x { \gtl_to_str:N #1 }
4554 }
(End definition for \__unravel_print_gtl_action:N.)

```

\\_\_unravel\_print\_done:x

```

4555 \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:

```

4556 \cs_new_protected_nopar:Npn \__unravel_print_assigned_token:
4557 {
4558   \__unravel_after_assignment: % ^^A todo: simplify
4559   \__unravel_print_action:x
4560   {
4561     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4562     = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
4563   }
4564   \__unravel_omit_after_assignment:w
4565 }
4566 \cs_new_protected_nopar:Npn \__unravel_print_assigned_register:
4567 {
4568   \__unravel_after_assignment: % ^^A todo: simplify
4569   \__unravel_print_action:x
4570   {
4571     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
4572     \tl_if_single:NT \l__unravel_defined_tl
4573     { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
4574     = \exp_after:wN \tex_the:D \l__unravel_defined_tl
4575   }
4576   \__unravel_omit_after_assignment:w
4577 }
(End definition for \__unravel_print_assigned_token: and \__unravel_print_assigned_register:.)

```

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

```

4578 \cs_new_protected_nopar:Npn \__unravel_print_welcome:
4579 {
4580   \__unravel_print_message:nn { }
4581   {
4582     \\
4583     =====~ Welcome~ to~ the~ unravel~ package~ =====\\

```

```

4584     \iow_indent:n
4585     {
4586         "<"~ denotes~ the~ output~ to~ TeX's~ stomach. \\
4587         "|"~ denotes~ tokens~ waiting~ to~ be~ used. \\
4588         ">"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
4589         Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
4590     }
4591 }
4592 \__unravel_print_state:
4593 }
(End definition for \__unravel_print_welcome:.)

```

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

```

4594 \cs_new_protected_nopar:Npn \__unravel_print_outcome:
4595 {
4596     % \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
4597     % \int_gset_eq:NN \g__unravel_max_prev_int \c_max_int
4598     % \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
4599     % \__unravel_print_state:
4600     \__unravel_print_message:nn { } { [====~The~end!~====] \\ }
4601 }
(End definition for \__unravel_print_outcome:.)

```

## 2.15.2 Prompt

\\_\_unravel\_prompt:

```

4602 \cs_new_protected_nopar:Npn \__unravel_prompt:
4603 {
4604     \int_gdecr:N \g__unravel_nonstop_int
4605     \int_compare:nNnF \g__unravel_nonstop_int > \c_zero
4606     {
4607         \group_begin:
4608             \int_set_eq:NN \tex_escapechar:D \c_minus_one
4609             \int_set_eq:NN \tex_endlinechar:D \c_minus_one
4610             \tl_use:N \g__unravel_prompt_before_tl
4611             \tl_gclear:N \g__unravel_prompt_before_tl
4612             \__unravel_prompt_aux:
4613         \group_end:
4614     }
4615 }
4616 \cs_new_protected_nopar:Npn \__unravel_prompt_aux:
4617 {
4618     \ior_get_str:Nc \g__unravel_prompt_ior { Your~input }
4619     \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
4620 }
4621 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
4622 {
4623     \tl_if_empty:nF {#1}
4624     {

```

```

4625 \exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
4626 {
4627   { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
4628   { q }
4629   {
4630     \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4631     \int_gzero:N \g__unravel_nonstop_int
4632   }
4633   { x }
4634   {
4635     \group_end:
4636     \exp_after:wN \__unravel_exit:w \__unravel_exit:w
4637   }
4638   { X } { \tex_batchmode:D \tex_end:D }
4639   { s } { \__unravel_prompt_scan_int:nn {#1}
4640     \__unravel_prompt_silent_steps:n }
4641   { o } { \__unravel_prompt_scan_int:nn {#1}
4642     { \int_gset:Nn \g__unravel_noise_int } }
4643   { C }
4644   {
4645     \tl_gset_rescan:Nnx \g__unravel_tmpc_tl
4646     { \ExplSyntaxOn } { \tl_tail:n {#1} }
4647     \tl_gput_left:Nn \g__unravel_tmpc_tl
4648     { \tl_gclear:N \g__unravel_tmpc_tl }
4649     \group_insert_after:N \g__unravel_tmpc_tl
4650   }
4651 }
4652 { \__unravel_prompt_help: }
4653 }
4654 }
4655 \cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
4656 {
4657   \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
4658   \l__unravel_prompt_tmpa_int = 0 \use_none:n #1 \scan_stop:
4659 }
4660 \cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
4661 {
4662   #2 \l__unravel_prompt_tmpa_int
4663   \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
4664 }
4665 \cs_new_protected:Npn \__unravel_prompt_help:
4666 {
4667   \__unravel_print:n { "m":~meaning-of~first-token }
4668   \__unravel_print:n { "q":~semi-quiet }
4669   \__unravel_print:n { "x":~exit~this~instance~of~unravel }
4670   \__unravel_print:n { "X":~try~harder~to~exit }
4671   \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
4672   \__unravel_print:n
4673   { "o<num>":~0~=>~only~log~not~online,~1~=>~both,~-1~=>~neither.}
4674   \__unravel_print:n { "C<code>":~run~some~code~immediately }

```

```

4675   \__unravel_prompt_aux:
4676   }
4677 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
4678   {
4679   \int_gset_eq:NN \g__unravel_noise_int \c_minus_one
4680   \tl_gset:Nn \g__unravel_prompt_before_tl
4681     { \int_gset_eq:NN \g__unravel_noise_int \c_one }
4682   \int_gset:Nn \g__unravel_nonstop_int {#1}
4683   }
(End definition for \__unravel_prompt:.)

```

## 2.16 Main command

`\unravel` Simply call an underlying internal command.

```

4684 \cs_new_protected:Npn \unravel #1 { \__unravel_unravel:n {#1} }
(End definition for \unravel. This function is documented on page ??.)

```

`\UnravelDebug` Turn on debugging mode.

```

4685 \cs_new_protected_nopar:Npn \UnravelDebug
4686   {
4687   \bool_set_true:N \l__unravel_debug_bool
4688   }
(End definition for \UnravelDebug. This function is documented on page ??.)

```

`\__unravel_unravel:n` Welcome the user, then initialize the input, output and step. Until the input is exhausted, print the current status and do one step.

```

4689 \cs_new_protected:Npn \__unravel_unravel:n #1
4690   {
4691   \int_gzero:N \g__unravel_step_int
4692   \__unravel_input_gset:n {#1}
4693   \seq_gclear:N \g__unravel_prev_input_seq
4694   \gtl_gclear:N \g__unravel_output_gtl
4695   \tl_gclear:N \g__unravel_if_limit_tl
4696   \int_gzero:N \g__unravel_if_limit_int
4697   \int_gzero:N \g__unravel_if_depth_int
4698   \gtl_gclear:N \g__unravel_after_assignment_gtl
4699   \bool_gset_true:N \g__unravel_set_box_allowed_bool
4700   \bool_gset_false:N \g__unravel_name_in_progress_bool
4701   \cs_gset_eq:NN \g__unravel_prompt_ior \c_minus_one % ^^A todo:?
4702   \__unravel_print_welcome:
4703   \__unravel_main_loop:
4704   \__unravel_exit_point:
4705   \__unravel_print_outcome:
4706   \bool_if:nTF
4707     {
4708     \tl_if_empty_p:N \g__unravel_if_limit_tl
4709     && \int_compare_p:nNn \g__unravel_if_limit_int = \c_zero
4710     && \int_compare_p:nNn \g__unravel_if_depth_int = \c_zero

```

```

4711     && \seq_if_empty_p:N \g__unravel_prev_input_seq
4712     }
4713     { \__unravel_input_if_empty:TF { } { \__unravel_bad_finish: } }
4714     { \__unravel_bad_finish: }
4715     \__unravel_exit_point:
4716     }
4717 \cs_new_protected_nopar:Npn \__unravel_bad_finish:
4718 {
4719     \msg_error:nxx { unravel } { internal }
4720     { the-last-unravel-finished-badly }
4721     }

```

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

`\__unravel_main_loop:` Loop forever, getting a token and performing the corresponding command.

```

4722 \cs_new_protected_nopar:Npn \__unravel_main_loop:
4723 {
4724     \__unravel_get_x_next:
4725     \__unravel_set_cmd:
4726     \__unravel_do_step:
4727     \__unravel_main_loop:
4728     }

```

*(End definition for \\_\_unravel\_main\_loop:.)*

## 2.17 Messages

```

4729 \msg_new:nnn { unravel } { unknown-primitive }
4730 { Internal~error:-the-primitive-’#1’~is-not-known. }
4731 \msg_new:nnn { unravel } { extra-fi-or-else }
4732 { Extra-fi,~or,~or~else. }
4733 \msg_new:nnn { unravel } { missing-lbrace }
4734 { Missing~left~brace~inserted. }
4735 \msg_new:nnn { unravel } { missing-dollar }
4736 { Missing~dollar~inserted. }
4737 \msg_new:nnn { unravel } { unknown-expandable }
4738 { Internal~error:-the-expandable-command-’#1’~is-not-known. }
4739 \msg_new:nnn { unravel } { missing-font-id }
4740 { Missing~font~identifier.-\iow_char:N\~nullfont~inserted. }
4741 \msg_new:nnn { unravel } { missing-rparen }
4742 { Missing~right~parenthesis~inserted-for-expression. }
4743 \msg_new:nnn { unravel } { incompatible-units }
4744 { Mu~glue/dimen-used-as-a-normal~glue/dimen-or~vice-versa. }
4745 \msg_new:nnn { unravel } { missing-mudim }
4746 { Missing~mu~unit. }
4747 \msg_new:nnn { unravel } { missing-cs }
4748 { Missing~control-sequence.-\iow_char:N\~inaccessible~inserted. }
4749 \msg_new:nnn { unravel } { missing-box }
4750 { Missing~box~inserted. }
4751 \msg_new:nnn { unravel } { missing-to }
4752 { Missing~keyword-’to’~inserted. }

```

```

4753 \msg_new:nnn { unravel } { improper-leaders }
4754   { Leaders-not-followed-by-proper-glue. }
4755 \msg_new:nnn { unravel } { extra-close }
4756   { Extra-right-brace-or-\iow_char:N\endgroup. }
4757 \msg_new:nnn { unravel } { off-save }
4758   { Something-is-wrong-with-groups. }
4759 \msg_new:nnn { unravel } { hrule-bad-mode }
4760   { \iow_char\hrule-used-in-wrong-mode. }
4761 \msg_new:nnn { unravel } { invalid-mode }
4762   { Invalid-mode-for-this-command. }
4763 \msg_new:nnn { unravel } { color-stack-action-missing }
4764   { Missing-color-stack-action. }
4765 \msg_new:nnn { unravel } { action-type-missing }
4766   { Missing-action-type. }
4767 \msg_new:nnn { unravel } { identifier-type-missing }
4768   { Missing-identifier-type. }
4769 \msg_new:nnn { unravel } { destination-type-missing }
4770   { Missing-destination-type. }
4771 \msg_new:nnn { unravel } { erroneous-prefixes }
4772   { Prefixes-applied-to-non-assignment-command. }
4773 \msg_new:nnn { unravel } { improper-setbox }
4774   { \iow_char:N\setbox-while-fetching-base-of-an-accent. }
4775 \msg_new:nnn { unravel } { after-advance }
4776   {
4777     Missing-register-after-\iow_char:N\advance,~
4778     \iow_char:N\multiply,~or~\iow_char:N\divide.
4779   }
4780 \msg_new:nnn { unravel } { bad-unless }
4781   { \iow_char:N\unless-not-followed-by-conditional. }
4782 \msg_new:nnn { unravel } { missing-endcsname }
4783   { Missing-\iow_char:N\endcsname-inserted. }
4784 \msg_new:nnn { unravel } { runaway-if }
4785   { Runaway-\iow_char:N\if... }
4786 \msg_new:nnn { unravel } { extra-or }
4787   { Extra-\iow_char:N\or. }
4788 \msg_new:nnn { unravel } { missing-equals }
4789   { Missing-equals-for-\iow_char:N\ifnum-or-\iow_char:N\ifdim. }
4790 \msg_new:nnn { unravel } { internal }
4791   { Internal-error:~'#1'.~\ Please report. }
4792 \msg_new:nnn { unravel } { not-implemented }
4793   { The-following-feature-is-not-implemented:~'#1'. }
4794 </package>

```