

The `unravel` package: watching TeX digest tokens*

Bruno Le Floch

2021/05/11

Contents

| | | |
|----------|---|----------|
| 1 | unravel documentation | 2 |
| 1.1 | Commands | 2 |
| 1.2 | Examples | 3 |
| 1.3 | Options | 4 |
| 1.4 | Differences between <code>unravel</code> and TeX's processing | 5 |
| 1.5 | Future perhaps | 6 |
| 2 | unravel implementation | 6 |
| 2.1 | Primitives, variants, and helpers | 10 |
| 2.1.1 | Adjustments to <code>expl3</code> | 10 |
| 2.1.2 | Renamed primitives | 10 |
| 2.1.3 | Variants | 11 |
| 2.1.4 | Miscellaneous helpers | 12 |
| 2.1.5 | String helpers | 13 |
| 2.1.6 | Helpers for control flow | 15 |
| 2.1.7 | Helpers concerning tokens | 15 |
| 2.1.8 | Helpers for previous input | 18 |
| 2.2 | Variables | 20 |
| 2.2.1 | User interaction | 20 |
| 2.2.2 | Working with tokens | 22 |
| 2.2.3 | Numbers and conditionals | 24 |
| 2.2.4 | Boxes and groups | 24 |
| 2.2.5 | Constants | 25 |
| 2.2.6 | TeX parameters | 25 |
| 2.3 | Numeric codes | 26 |
| 2.4 | Get next token | 40 |
| 2.5 | Manipulating the input | 46 |
| 2.5.1 | Elementary operations | 46 |
| 2.5.2 | Insert token for error recovery | 51 |
| 2.5.3 | Macro calls | 52 |
| 2.6 | Expand next token | 53 |
| 2.7 | Basic scanning subroutines | 55 |

*This file has version number 0.3a, last revised 2021/05/11.

| | | |
|--------|--------------------------|-----|
| 2.8 | Working with boxes | 74 |
| 2.9 | Paragraphs | 78 |
| 2.10 | Groups | 80 |
| 2.11 | Modes | 83 |
| 2.12 | Commands | 85 |
| 2.12.1 | Characters: from 0 to 15 | 85 |
| 2.12.2 | Boxes: from 16 to 31 | 90 |
| 2.12.3 | From 32 to 47 | 95 |
| 2.12.4 | Maths: from 48 to 56 | 99 |
| 2.12.5 | From 57 to 70 | 101 |
| 2.12.6 | Extensions | 104 |
| 2.12.7 | Assignments | 111 |
| 2.13 | Expandable primitives | 121 |
| 2.13.1 | Conditionals | 129 |
| 2.14 | User interaction | 137 |
| 2.14.1 | Print | 137 |
| 2.14.2 | Prompt | 143 |
| 2.14.3 | Errors | 146 |
| 2.15 | Keys | 148 |
| 2.16 | Main command | 149 |
| 2.17 | Messages | 152 |

1 unravel documentation

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

1.1 Commands

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

This command shows in the terminal the steps performed by T_EX when running the *code*. By default, it pauses to let the user read the description of every step: simply press `<return>` to proceed. Typing `s<integer>` instead will go forward *integer* steps somewhat silently. In the future it will be possible to use a negative *integer* to go back a few steps. Typing `h` gives a list of various other possibilities. The available *key-value* options are described in Section 1.3.

`\unravelsetup` {*options*}

Sets *options* that apply to all subsequent `\unravel`. See options in Section 1.3.

`\unravel:nn` {*options*} {*code*}

See `\unravel`.

| | |
|-------------------------------|---|
| <code>\unravel_get:nnN</code> | <code>\unravel_get:nnN {<options>} {<code>} <tl var></code> |
| | Performs <code>\unravel:nn</code> with the <code><options></code> and <code><code></code> then saves the output into the <code><tl var></code> . The option <code>mute</code> is useful in this case. |
| <code>\unravel_setup:n</code> | <code>\unravel_setup:n {<options>}</code> |
| | See <code>\unravelsetup</code> . |

1.2 Examples

The `unravel` package is currently based on the behaviour of `pdfTeX`, but it should work in all engines supported by `expl3` (`pdfTeX`, `XYTeX`, `LuaTeX`, `epTeX`, `eupTeX`) as long as none of the primitives specific to those engines is used. Any difference between how `unravel` and `(pdf)TeX` process a given piece of code, unless described in the section 1.4, should be reported on the issue tracker (<https://github.com/blefloch/latex-unravel/issues>).

As a simple example, one can run `LATeX` on the following file.

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

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

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

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

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

Given all the work that `unravel` has to do to emulate \TeX , it is not fast on very large pieces of code. For instance, running it on `\documentclass{article}` takes about thirty seconds on my machine, and finishes after somewhat less than 21000 steps.

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

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

1.3 Options

`explicit-prompt`

Boolean option (default `false`) determining whether to give an explicit prompt. If `true`, the text “Your input=” will appear at the beginning of lines where user input is expected.

`internal-debug`

Boolean option (default `false`) used to debug `unravel` itself.

`machine`

Option which takes no value and makes `unravel` produce an output that is somewhat more suitable for automatic processing. In particular, it sets `max-action`, `max-output`, `max-input` to very large values, and `number-steps` to `false`.

`max-action`
`max-output`
`max-input`

Integer options (defaults 50, 300, 300) determining the maximum number of characters displayed for the action, the output part, and the input part.

`mute`

Make none of the steps produce any output, by setting `trace-assigns`, `trace-expansion`, `trace-other`, `welcome-message` to `false`. This is only useful with `\unravel_get:nnN` or when other options change some of these settings.

`number-steps`

Boolean option (default `true`) determining whether to number steps.

`online`

Integer option determining where to write the output: terminal and log if the option is positive, log only if the option is zero, neither if the option is negative.

`prompt-input`

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

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

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

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

`welcome-message`

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

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

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

Differences.

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

- Explicit begin-group and end-group characters other than the usual left and right braces may make `unravel` choke, or may be silently replaced by the usual left and right braces.
- `\endinput` is ignored with a warning, as it is very difficult to implement it in a way similar to `TEX`'s, and as it is most often used at the very end of files, in a redundant way.
- `\outer` is not supported.
- `\unravel` cannot be nested.
- Control sequences of the form `\notexpanded: . . .` are reserved for use by `unravel`.

1.5 Future perhaps

- Use the `file-error` fatal error message: first implement `\@@_file_if_exist:nTF` and use it to determine whether `\input` will throw a fatal error in `\batchmode` and `\nonstopmode`.
- Use the `interwoven-preambles` fatal error message once alignments are implemented.
- Find out why so many input levels are used (see the log of the `unravel003` testfile for instance)

2 unravel implementation

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

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

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

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

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

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

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

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

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

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

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

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

(End definition for `\l__unravel_setup_restore_tl`.)

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

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

(End definition for `__unravel_setup_restore:`.)

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

`__unravel_setup_save_aux:n`

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

(End definition for `_ unravel_setup_save:` and `_ unravel_setup_save_aux:n.`)

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

(End definition for `_ unravel_setup_catcodes:nmn.`)

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

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

(End definition for `_ unravel_setup_latex:.`)

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

```

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

```

(End definition for __unravel_setup_unravel:.)

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

```

127 \expandafter \endgroup \__unravel_setup_latex:

```

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

```

128 \RequirePackage{expl3,xparse}[2021/01/01]
129 \RequirePackage{gtl}[2018/12/28]
130 \csname cs_if_exist:cF\endcsname{str_range:nmn}{\RequirePackage{l3str}}

```

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

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

```

131 \csname use:n\endcsname
132 {%
133   \csname __unravel_setup_restore:\endcsname
134   \ProvidesExplPackage
135     {unravel} {2021/05/11} {0.3a} {Watching TeX digest tokens}%
136   \csname __unravel_setup_unravel:\endcsname
137 }%
```

2.1 Primitives, variants, and helpers

2.1.1 Adjustments to `expl3`

In upcoming versions of `expl3`, the `\group_align_safe_begin:` and `\group_align_safe_end:` commands may involve an explicit end-group character token with non-standard character code, which would wrongly be normalized by `gtl` (used by `unravel`), hence break. To avoid this we change here the definitions slightly.

```

138 \cs_gset:Npn \group_align_safe_begin:
139   { \exp:w \if_false: { \fi: -' } \exp_stop_f: }
140 \cs_gset:Npn \group_align_safe_end:
141   { \if_int_compare:w '{ = \c_zero_int } \fi: }
```

2.1.2 Renamed primitives

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

```

142 \cs_new_eq:NN \__unravel_currentgrouptype: \tex_currentgrouptype:D
143 \cs_new_protected:Npn \__unravel_set_escapechar:n
144   { \int_set:Nn \tex_escapechar:D }
145 \cs_new_eq:NN \__unravel_everyeof:w \tex_everyeof:D
146 \cs_new_eq:NN \__unravel_everypar:w \tex_everypar:D
147 \cs_new_eq:NN \__unravel_hbox:w \tex_hbox:D
148 \cs_new_eq:NN \__unravel_mag: \tex_mag:D
149 \cs_new_eq:NN \__unravel_nullfont: \tex_nullfont:D
150 \cs_new_eq:NN \__unravel_the:w \tex_the:D
151 \cs_new_eq:NN \__unravel_number:w \tex_number:D
```

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

`__unravel_special_relax:` A special marker slightly different from `\relax` (its `\meaning` is `\relax` but it differs from `\relax` according to `\ifx`). In the right-hand side of our assignment, `__unravel_special_relax:` could be replaced by any other expandable command.

```

152 \exp_after:wN \cs_new_eq:NN
153   \exp_after:wN \__unravel_special_relax:
154   \exp_not:N \__unravel_special_relax:
```

(End definition for `__unravel_special_relax:.`)

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

```
155 \int_const:Nn \c__unravel_prompt_ior { 16 }
156 \int_const:Nn \c__unravel_noprompt_ior { -1 }
```

(End definition for `\c__unravel_prompt_ior` and `\c__unravel_noprompt_ior`.)

2.1.3 Variants

Variants that we need.

```
157 \cs_generate_variant:Nn \seq_push:Nn { Nf }
158 \cs_generate_variant:Nn \str_head:n { f }
159 \cs_generate_variant:Nn \tl_to_str:n { o }
160 \cs_generate_variant:Nn \tl_if_eq:nnTF { o }
161 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
162 \cs_generate_variant:Nn \tl_if_head_eq_meaning:nNTF { V }
163 \cs_generate_variant:Nn \tl_if_single_token:nT { V }
164 \cs_generate_variant:Nn \gtl_gput_right:Nn { NV }
165 \cs_generate_variant:Nn \gtl_if_empty:NTF { c }
166 \cs_generate_variant:Nn \gtl_if_tl:NT { c }
167 \cs_generate_variant:Nn \gtl_to_str:N { c }
168 \cs_generate_variant:Nn \gtl_gpop_left:NN { c }
169 \cs_generate_variant:Nn \gtl_get_left:NN { c }
170 \cs_generate_variant:Nn \gtl_gset:Nn { c }
171 \cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
172 \cs_generate_variant:Nn \gtl_gclear:N { c }
173 \cs_generate_variant:Nn \gtl_gclear_new:N { c }
174 \cs_generate_variant:Nn \gtl_left_tl:N { c }
```

`__unravel_tl_if_in:ooTF` Analogue of `\tl_if_in:ooTF` but with an extra group because that function redefines an auxiliary that may appear in the code being debugged (see Github issue #27).

```
175 \cs_new_protected:Npn \__unravel_tl_if_in:ooTF #1#2#3#4
176 {
177   \group_begin:
178   \exp_args:Noo \tl_if_in:nnTF {#1} {#2}
179   { \group_end: #3 } { \group_end: #4 }
180 }
```

(End definition for `__unravel_tl_if_in:ooTF`.)

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

```
181 \tl_new:N \l__unravel_exp_tl
182 \cs_new_protected:Npn \__unravel_exp_args:Nx #1#2
183 {
184   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 {#2} }
185   \l__unravel_exp_tl
186 }
187 \cs_new_protected:Npn \__unravel_exp_args:NNx #1#2#3
188 {
189   \cs_set_nopar:Npx \l__unravel_exp_tl { \exp_not:N #1 \exp_not:N #2 {#3} }
190   \l__unravel_exp_tl
191 }
```

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

2.1.4 Miscellaneous helpers

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

```
192 \cs_new_protected:Npn \__unravel_tmp:w { }
```

(End definition for `__unravel_tmp:w`.)

`__unravel_file_get:nN`
`__unravel_file_get_aux:wN`

```
193 \cs_set_protected:Npn \__unravel_tmp:w #1
194 {
195   \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
196   {
197     \group_begin:
198     \__unravel_everyeof:w { #1 ##2 }
199     \exp_after:wN \__unravel_file_get_aux:wN
200     \exp_after:wN \prg_do_nothing:
201     \tex_input:D ##1 \scan_stop:
202   }
203   \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
204   {
205     \group_end:
206     \tl_set:Nx ##2
207     { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
208   }
209 }
210 \exp_args:No \__unravel_tmp:w { \token_to_str:N : : }
```

(End definition for `__unravel_file_get:nN` and `__unravel_file_get_aux:wN`.)

`__unravel_tl_first_int:N`
`__unravel_tl_first_int_aux:Nn`

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

```
211 \cs_new:Npn \__unravel_tl_first_int:N #1
212 {
213   \int_eval:n
214   {
215     \exp_after:wN \__unravel_tl_first_int_aux:Nn
216     \exp_after:wN \__unravel_tl_first_int_aux:Nn
217     #1 ? 0 ? \q_stop
218   }
219 }
220 \cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
221 {
222   \tl_if_single:nT {#2}
223   {
224     \token_if_eq_catcode:NNT + #2
225     {
226       \if_int_compare:w 1 < 1 #2 \exp_stop_f:
227       #2
```



```

258 {
259   \tex_romannumeral:D
260   \if_charcode:w \token_to_str:N \ \_unravel_strip_escape_aux:w \fi:
261   \_unravel_strip_escape_aux:N
262 }
263 \cs_new:Npn \_unravel_strip_escape_aux:N #1 { \c_zero_int }
264 \cs_new:Npn \_unravel_strip_escape_aux:w #1#2
265 { - \_unravel_number:w #1 \c_zero_int }

```

(End definition for `_unravel_strip_escape:w`, `_unravel_strip_escape_aux:N`, and `_unravel_strip_escape_aux:w`.)

`_unravel_to_str:Nn` Use the type-appropriate conversion to string.

```

266 \cs_new:Npn \_unravel_to_str:Nn #1
267 {
268   \if_meaning:w T #1
269   \exp_after:wN \tl_to_str:n
270   \else:
271   \exp_after:wN \gtl_to_str:n
272   \fi:
273 }

```

(End definition for `_unravel_to_str:Nn`.)

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

```

274 \cs_new:Npn \_unravel_str_truncate_left:nn #1#2
275 {
276   \exp_args:Nf \_unravel_str_truncate_left_aux:nnn
277   { \str_count:n {#1} } {#1} {#2}
278 }
279 \cs_new:Npn \_unravel_str_truncate_left_aux:nnn #1#2#3
280 {
281   \int_compare:nNnTF {#1} > {#3}
282   {
283     ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
284     \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
285   }
286   { \tl_to_str:n {#2} }
287 }

```

(End definition for `_unravel_str_truncate_left:nn` and `_unravel_str_truncate_left_aux:nnn`.)

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

```

288 \cs_new:Npn \_unravel_str_truncate_right:nn #1#2
289 {
290   \exp_args:Nf \_unravel_str_truncate_right_aux:nnn
291   { \str_count:n {#1} } {#1} {#2}
292 }
293 \cs_new:Npn \_unravel_str_truncate_right_aux:nnn #1#2#3
294 {

```

```

295 \int_compare:nNnTF {#1} > {#3}
296 {
297   \str_range:nnn {#2} { 1 } { #3 - 25 } ~
298   ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
299 }
300 { \tl_to_str:n {#2} }
301 }

```

(End definition for `__unravel_str_truncate_right:nn` and `__unravel_str_truncate_right_aux:nnn`.)

2.1.6 Helpers for control flow

```

\__unravel_exit:w Jump to the very end of this instance of \unravel.
\__unravel_exit_hard:w
\__unravel_exit_point:
302 \cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
303 \cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }
304 \cs_new:Npn \__unravel_exit_hard:w #1 \__unravel_exit_point: #2 \__unravel_exit_point: { }

```

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

```

\__unravel_break:w Useful to jump out of complicated conditionals.
\__unravel_break_point:
305 \cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
306 \cs_new:Npn \__unravel_break:w #1 \__unravel_break_point: { }

```

(End definition for `__unravel_break:w` and `__unravel_break_point:.`)

```

\__unravel_cmd_if_internal:TF Test whether the \l__unravel_head_cmd_int denotes an “internal” command, between
min_internal and max_internal (see Section 2.3).

```

```

307 \prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
308 {
309   \int_compare:nNnTF
310     \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
311     { \prg_return_false: }
312     {
313       \int_compare:nNnTF
314         \l__unravel_head_cmd_int
315         > { \__unravel_tex_use:n { max_internal } }
316         { \prg_return_false: }
317         { \prg_return_true: }
318     }
319 }

```

(End definition for `__unravel_cmd_if_internal:TF`.)

2.1.7 Helpers concerning tokens

```

\__unravel_active_do:nn Apply some code to an active character token constructed from its character code.

```

```

320 \cs_new_protected:Npn \__unravel_active_do:nn #1#2
321 {
322   \group_begin:
323   \char_set_active_eq:nN {#1} \scan_stop:
324   \use:x
325   {
326     \group_end:
327     \exp_not:n {#2} { \char_generate:nn {#1} { 13 } }
328   }
329 }

```

(End definition for `_unravel_active_do:nn`.)

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

```
330 \cs_new:Npn \_unravel_meaning_to_char:n #1
331   { \_unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
332 \cs_new:Npn \_unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
333   { \_unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
334 \cs_new:Npn \_unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
335   { \tl_if_empty:nTF {#2} { ~ } {#2} }
336 \cs_generate_variant:Nn \_unravel_meaning_to_char:n { o }
337 \cs_new:Npn \_unravel_token_to_char:N #1
338   { \_unravel_meaning_to_char:o { \token_to_meaning:N #1 } }
```

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

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

```
339 \prg_new_conditional:Npnn \_unravel_token_if_expandable:N #1
340   { p , T , F , TF }
341   {
342     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
343     \prg_return_false:
344   \else:
345     \prg_return_true:
346   \fi:
347   }
```

(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.

```
348 \prg_new_conditional:Npnn \_unravel_token_if_protected:N #1
349   { p , T , F , TF }
350   {
351     \_unravel_token_if_expandable:N #1
352     {
353       \token_if_protected_macro:N #1
354       { \prg_return_true: }
355       {
356         \token_if_protected_long_macro:N #1
357         { \prg_return_true: }
358         { \prg_return_false: }
359       }
360     }
361     { \prg_return_true: }
362   }
```

(End definition for `_unravel_token_if_protected:N`.)

`_unravel_token_if_active_char:N` Lowercase the token after setting its `\lccode` (more precisely the `\lccode` of the first
character in its string representation) to a known value, then compare the result with
that active character.

```
363 \group_begin:
```



```

364 \char_set_catcode_active:n { 'Z }
365 \prg_new_protected_conditional:Npnn \__unravel_token_if_active_char:N #1
366 { TF }
367 {
368   \group_begin:
369     \__unravel_exp_args:Nx \char_set_lccode:nn
370     { ' \exp_args:No \str_head:n { \token_to_str:N #1 } }
371     { ' Z }
372     \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
373     { \group_end: \prg_return_true: }
374     { \group_end: \prg_return_false: }
375   }
376 \group_end:

```

(End definition for `__unravel_token_if_active_char:NTF`.)

`__unravel_token_if_definable:NTF` Within a group, set the escape character to a non-space value (backslash). Convert the token to a string with `\token_to_str:N`. The result is multiple characters if the token is a control sequence, and a single character otherwise (even for explicit catcode 6 character tokens which would be doubled if we used `\tl_to_str:n` instead of `\token_to_str:N`). Thus `\str_tail:n` gives a non-empty result exactly for control sequences. Those are definable (technically, not always: `\expandafter\font\csname\endcsname=cmr10\expandafter\def\the\csname\endcsname{}`). For characters just check for active characters. In both cases remember to end the group.

```

377 \group_begin:
378 \char_set_catcode_active:n { 'Z }
379 \prg_new_protected_conditional:Npnn \__unravel_token_if_definable:N #1
380 { TF }
381 {
382   \group_begin:
383     \__unravel_set_escapechar:n { 92 }
384     \tl_set:Nx \l__unravel_tmpa_tl
385     { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
386     \tl_if_empty:NTF \l__unravel_tmpa_tl
387     {
388       \__unravel_token_if_active_char:NTF #1
389       { \group_end: \prg_return_true: }
390       { \group_end: \prg_return_false: }
391     }
392     { \group_end: \prg_return_true: }
393   }
394 \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.

```

395 \prg_new_protected_conditional:Npnn \__unravel_gtl_if_head_is_definable:N #1
396 { TF , F }
397 {
398   \gtl_if_single_token:NTF #1
399   {

```

```

400     \gtl_if_head_is_N_type:NTF #1
401     {
402         \gtl_head_do:NN #1 \__unravel_token_if_definable:NTF
403         { \prg_return_true: }
404         { \prg_return_false: }
405     }
406     { \prg_return_false: }
407 }
408 { \prg_return_false: }
409 }

```

(End definition for __unravel_gtl_if_head_is_definable:NTF.)

2.1.8 Helpers for previous input

__unravel_prev_input_count: Count prev input levels, skipping empty ones (of either tl or gtl type).

```

\__unravel_prev_input_count_aux:n 410 \cs_new:Npn \__unravel_prev_input_count:
\__unravel_prev_input_count_aux:Nn 411 {
412     \int_eval:n
413     {
414         0
415         \seq_map_function:NN \g__unravel_prev_input_seq
416         \__unravel_prev_input_count_aux:n
417     }
418 }
419 \cs_new:Npn \__unravel_prev_input_count_aux:n #1
420 { \__unravel_prev_input_count_aux:Nn #1 }
421 \cs_new:Npn \__unravel_prev_input_count_aux:Nn #1#2
422 {
423     \if_meaning:w T #1
424     \exp_after:wN \tl_if_empty:nF
425     \else:
426     \exp_after:wN \str_if_eq:onF \exp_after:wN \c_empty_gtl
427     \fi:
428     {#2} { + 1 }
429 }

```

(End definition for __unravel_prev_input_count:, __unravel_prev_input_count_aux:n, and __unravel_prev_input_count_aux:Nn.)

__unravel_prev_input_gpush:

```

\__unravel_prev_input_gpush:N 430 \cs_new_protected:Npn \__unravel_prev_input_gpush:
\__unravel_prev_input_gpush_gtl: 431 { \seq_gput_right:Nn \g__unravel_prev_input_seq { T { } } }
\__unravel_prev_input_gpush_gtl:N 432 \cs_new_protected:Npn \__unravel_prev_input_gpush:N
\__unravel_prev_input_gpush_aux:NN 433 { \__unravel_prev_input_gpush_aux:NN T }
434 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
435 { \__unravel_prev_input_gpush_gtl:N \c_empty_gtl }
436 \cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
437 { \__unravel_prev_input_gpush_aux:NN G }
438 \cs_new_protected:Npn \__unravel_prev_input_gpush_aux:NN #1#2
439 { \seq_gput_right:Nx \g__unravel_prev_input_seq { #1 { \exp_not:o {#2} } } }

```

(End definition for __unravel_prev_input_gpush: and others.)

```

\l__unravel_prev_aux_tl
\__unravel_prev_input_get:N 440 \tl_new:N \l__unravel_prev_aux_tl
\__unravel_prev_input_gpop:N 441 \cs_new_protected:Npn \__unravel_prev_input_get:N
  \__unravel_prev_input_gpop_gtl:N 442 { \__unravel_prev_input_aux:NNN \seq_get_right:NN T }
  \__unravel_prev_input_aux:NNN 443 \cs_new_protected:Npn \__unravel_prev_input_gpop:N
  \__unravel_prev_input_aux:NNNn 444 { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN T }
  \__unravel_prev_input_aux:NNNn 445 \cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
  \__unravel_prev_input_aux:NNN 446 { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN G }
  \__unravel_prev_input_aux:NNNn 447 \cs_new_protected:Npn \__unravel_prev_input_aux:NNN #1#2#3
  448 {
  449   #1 \g__unravel_prev_input_seq \l__unravel_prev_aux_tl
  450   \exp_after:wN \__unravel_prev_input_aux:NNNn
  451   \exp_after:wN #2 \exp_after:wN #3 \l__unravel_prev_aux_tl
  452 }
  453 \cs_new_protected:Npn \__unravel_prev_input_aux:NNNn #1#2#3
  454 {
  455   \token_if_eq_meaning:NNTF #1#3
  456   { \tl_set:Nn }
  457   { \msg_error:nnnnnn { unravel } { prev-input } {#1} {#3} }
  458   #2
  459 }

```

(End definition for \l__unravel_prev_aux_tl and others.)

```

\__unravel_prev_input_silent:n
\__unravel_prev_input_silent:V 460 \cs_new_protected:Npn \__unravel_prev_input_silent:n #1
\__unravel_prev_input_silent:x 461 {
  \__unravel_prev_input:n 462   \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
  \__unravel_prev_input:V 463   \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
  \__unravel_prev_input:x 464   \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
  465 }
  466 \cs_generate_variant:Nn \__unravel_prev_input_silent:n { V }
  467 \cs_new_protected:Npn \__unravel_prev_input_silent:x
  468 { \__unravel_exp_args:Nx \__unravel_prev_input_silent:n }
  469 \cs_new_protected:Npn \__unravel_prev_input:n #1
  470 {
  471   \__unravel_prev_input_silent:n {#1}
  472   \__unravel_print_action:x { \tl_to_str:n {#1} }
  473 }
  474 \cs_generate_variant:Nn \__unravel_prev_input:n { V }
  475 \cs_new_protected:Npn \__unravel_prev_input:x
  476 { \__unravel_exp_args:Nx \__unravel_prev_input:n }

```

(End definition for __unravel_prev_input_silent:n and __unravel_prev_input:n.)

```

\__unravel_prev_input_gtl:N
  477 \cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
  478 {
  479   \__unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
  480   \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
  481   \__unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
  482 }

```

(End definition for __unravel_prev_input_gtl:N.)

```

\__unravel_prev_input_join_get:nnN
\__unravel_join_get_aux:NNnN
483 \cs_new_protected:Npn \__unravel_prev_input_join_get:nnN #1
484 {
485   \int_case:nnF {#1}
486   {
487     { 2 } { \__unravel_join_get_aux:NNnN \skip_eval:n \tex_glueexpr:D }
488     { 3 } { \__unravel_join_get_aux:NNnN \muskip_eval:n \tex_muexpr:D }
489   }
490   {
491     \__unravel_error:nnnnn { internal } { join-factor } { } { } { }
492     \__unravel_join_get_aux:NNnN \use:n \prg_do_nothing:
493   }
494 }
495 \cs_new_protected:Npn \__unravel_join_get_aux:NNnN #1#2#3#4
496 {
497   \__unravel_prev_input_gpop:N \l__unravel_head_tl
498   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
499   \tl_set:Nx #4 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl #3 } }
500 }

```

(End definition for __unravel_prev_input_join_get:nnN and __unravel_join_get_aux:NNnN.)

2.2 Variables

2.2.1 User interaction

```

\g__unravel_before_print_state_tl
\g__unravel_before_prompt_tl
501 \tl_new:N \g__unravel_before_print_state_tl
502 \tl_new:N \g__unravel_before_prompt_tl

```

(End definition for \g__unravel_before_print_state_tl and \g__unravel_before_prompt_tl.)

```

\l__unravel_prompt_tmpa_int
503 \int_new:N \l__unravel_prompt_tmpa_int

```

(End definition for \l__unravel_prompt_tmpa_int.)

```

\g__unravel_nonstop_int
504 \int_new:N \g__unravel_nonstop_int

```

(End definition for \g__unravel_nonstop_int.)

```

\g__unravel_default_explicit_prompt_bool
\g__unravel_explicit_prompt_bool
\g__unravel_default_internal_debug_bool
\g__unravel_internal_debug_bool
\g__unravel_default_number_steps_bool
\g__unravel_number_steps_bool
\g__unravel_default_online_int
\g__unravel_online_int
\g__unravel_default_prompt_input_clist
\g__unravel_prompt_input_clist
\g__unravel_default_trace_assigns_bool
\g__unravel_trace_assigns_bool
\g__unravel_default_trace_expansion_bool
\g__unravel_trace_expansion_bool
\g__unravel_default_trace_other_bool
\g__unravel_trace_other_bool
\g__unravel_default_welcome_message_bool
\g__unravel_welcome_message_bool

```

Variables for the options `explicit-prompt`, `internal-debug`, `number-steps`, and so on. The `default_` booleans/integers store the default value of these options, and are affected by `\unravelsetup` or `\unravel_setup:n`.

```

505 \bool_new:N \g__unravel_default_explicit_prompt_bool
506 \bool_new:N \g__unravel_default_internal_debug_bool
507 \bool_new:N \g__unravel_default_number_steps_bool
508 \int_new:N \g__unravel_default_online_int
509 \clist_new:N \g__unravel_default_prompt_input_clist
510 \bool_new:N \g__unravel_default_trace_assigns_bool
511 \bool_new:N \g__unravel_default_trace_expansion_bool

```

```

512 \bool_new:N \g__unravel_default_trace_other_bool
513 \bool_new:N \g__unravel_default_welcome_message_bool
514 \bool_gset_true:N \g__unravel_default_number_steps_bool
515 \int_gset:Nn \g__unravel_default_online_int { 1 }
516 \bool_gset_true:N \g__unravel_default_trace_assigns_bool
517 \bool_gset_true:N \g__unravel_default_trace_expansion_bool
518 \bool_gset_true:N \g__unravel_default_trace_other_bool
519 \bool_gset_true:N \g__unravel_default_welcome_message_bool
520 \bool_new:N \g__unravel_explicit_prompt_bool
521 \bool_new:N \g__unravel_internal_debug_bool
522 \bool_new:N \g__unravel_number_steps_bool
523 \int_new:N \g__unravel_online_int
524 \clist_new:N \g__unravel_prompt_input_clist
525 \bool_new:N \g__unravel_trace_assigns_bool
526 \bool_new:N \g__unravel_trace_expansion_bool
527 \bool_new:N \g__unravel_trace_other_bool
528 \bool_new:N \g__unravel_welcome_message_bool

```

(End definition for \g__unravel_default_explicit_prompt_bool and others.)

`\g__unravel_step_int` Current expansion step.

```
529 \int_new:N \g__unravel_step_int
```

(End definition for \g__unravel_step_int.)

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

```
530 \str_new:N \g__unravel_action_text_str
```

(End definition for \g__unravel_action_text_str.)

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

```

\g__unravel_default_max_output_int 531 \int_new:N \g__unravel_default_max_action_int
\g__unravel_default_max_input_int 532 \int_new:N \g__unravel_default_max_output_int
\g__unravel_max_action_int 533 \int_new:N \g__unravel_default_max_input_int
\g__unravel_max_output_int 534 \int_gset:Nn \g__unravel_default_max_action_int { 50 }
\g__unravel_max_input_int 535 \int_gset:Nn \g__unravel_default_max_output_int { 300 }
536 \int_gset:Nn \g__unravel_default_max_input_int { 300 }
537 \int_new:N \g__unravel_max_action_int
538 \int_new:N \g__unravel_max_output_int
539 \int_new:N \g__unravel_max_input_int

```

(End definition for \g__unravel_default_max_action_int and others.)

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

```
540 \bool_new:N \g__unravel_speedup_macros_bool
541 \bool_gset_true:N \g__unravel_speedup_macros_bool
```

(End definition for \g__unravel_speedup_macros_bool.)

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

```
542 \int_new:N \l__unravel_print_int
```

(End definition for \l__unravel_print_int.)

2.2.2 Working with tokens

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

```
543 \int_new:N \g__unravel_input_int
```

(End definition for `\g__unravel_input_int`.)

`\g__unravel_input_tmpa_int`
`\l__unravel_input_tmpa_tl`

```
544 \int_new:N \g__unravel_input_tmpa_int
```

```
545 \tl_new:N \l__unravel_input_tmpa_tl
```

(End definition for `\g__unravel_input_tmpa_int` and `\l__unravel_input_tmpa_tl`.)

`\g__unravel_prev_input_seq`
`\l__unravel_prev_input_tl`
`\l__unravel_prev_input_gtl`

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

```
546 \seq_new:N \g__unravel_prev_input_seq
```

```
547 \tl_new:N \l__unravel_prev_input_tl
```

```
548 \gtl_new:N \l__unravel_prev_input_gtl
```

(End definition for `\g__unravel_prev_input_seq`, `\l__unravel_prev_input_tl`, and `\l__unravel_prev_input_gtl`.)

`\g__unravel_output_gtl`

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

```
549 \gtl_new:N \g__unravel_output_gtl
```

(End definition for `\g__unravel_output_gtl`.)

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

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

```
550 \gtl_new:N \l__unravel_head_gtl
```

```
551 \tl_new:N \l__unravel_head_tl
```

```
552 \cs_new_eq:NN \l__unravel_head_token ?
```

```
553 \int_new:N \l__unravel_head_cmd_int
```

```
554 \int_new:N \l__unravel_head_char_int
```

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

`\l__unravel_head_meaning_tl`

```
555 \tl_new:N \l__unravel_head_meaning_tl
```

(End definition for `\l__unravel_head_meaning_tl`.)

`\l__unravel_argi_tl`
`\l__unravel_argii_tl`

Token list variables used to store first/second arguments.

```
556 \tl_new:N \l__unravel_argi_tl
```

```
557 \tl_new:N \l__unravel_argii_tl
```

(End definition for `\l__unravel_argi_tl` and `\l__unravel_argii_tl`.)

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

`\l__unravel_tmpb_gtl`

`\g__unravel_tmpc_tl` 558 `\tl_new:N \l__unravel_tmpa_tl`

`\l__unravel_tmpa_seq` 559 `\gtl_new:N \l__unravel_unused_gtl`

`\l__unravel_unused_gtl` 560 `\gtl_new:N \l__unravel_tmpb_gtl`

`\l__unravel_tmpb_token` 561 `\tl_new:N \g__unravel_tmpc_tl`

562 `\seq_new:N \l__unravel_tmpa_seq`

563 `\cs_new_eq:NN \l__unravel_tmpb_token ?`

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

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

564 `\tl_new:N \l__unravel_defined_tl`

565 `\tl_new:N \l__unravel_defining_tl`

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

`__unravel_inaccessible:w`

566 `\cs_new_eq:NN __unravel_inaccessible:w ?`

(End definition for `__unravel_inaccessible:w`.)

`\g__unravel_lastnamedcs_tl`

Used for LuaTeX's `\lastnamedcs` primitive.

567 `\tl_new:N \g__unravel_lastnamedcs_tl`

(End definition for `\g__unravel_lastnamedcs_tl`.)

`\g__unravel_after_assignment_gtl`

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

`\g__unravel_set_box_allowed_bool`

`\g__unravel_name_in_progress_bool`

568 `\gtl_new:N \g__unravel_after_assignment_gtl`

569 `\bool_new:N \g__unravel_set_box_allowed_bool`

570 `\bool_new:N \g__unravel_name_in_progress_bool`

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

`\l__unravel_after_group_gtl`

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

571 `\gtl_new:N \l__unravel_after_group_gtl`

(End definition for `\l__unravel_after_group_gtl`.)

`\c__unravel_parameters_tl`

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

572 `\group_begin:`

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

574 `\tl_const:Nx \c__unravel_parameters_tl`

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

576 `\group_end:`

(End definition for `\c__unravel_parameters_tl`.)

2.2.3 Numbers and conditionals

`\g__unravel_val_level_int` See TeX's `cur_val_level` variable. This is set by `__unravel_rescan_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.

```
577 \int_new:N \g__unravel_val_level_int
```

(End definition for `\g__unravel_val_level_int`.)

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

```
\g__unravel_if_limit_int 578 \tl_new:N \g__unravel_if_limit_tl
\g__unravel_if_depth_int 579 \int_new:N \g__unravel_if_limit_int
580 \int_new:N \g__unravel_if_depth_int
```

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

`\l__unravel_if_nesting_int`

```
581 \int_new:N \l__unravel_if_nesting_int
```

(End definition for `\l__unravel_if_nesting_int`.)

2.2.4 Boxes and groups

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

```
582 \seq_new:N \l__unravel_leaders_box_seq
```

(End definition for `\l__unravel_leaders_box_seq`.)

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

```
583 \int_new:N \g__unravel_ends_int
584 \int_gset:Nn \g__unravel_ends_int { 3 }
```

(End definition for `\g__unravel_ends_int`.)

2.2.5 Constants

```

\c__unravel_plus_tl
\c__unravel_minus_tl 585 \tl_const:Nn \c__unravel_plus_tl { + }
\c__unravel_times_tl 586 \tl_const:Nn \c__unravel_minus_tl { - }
\c__unravel_over_tl 587 \tl_const:Nn \c__unravel_times_tl { * }
\c__unravel_lq_tl 588 \tl_const:Nn \c__unravel_over_tl { / }
\c__unravel_rq_tl 589 \tl_const:Nn \c__unravel_lq_tl { ' }
\c__unravel_dq_tl 590 \tl_const:Nn \c__unravel_rq_tl { ' }
\c__unravel_lp_tl 591 \tl_const:Nn \c__unravel_dq_tl { " }
\c__unravel_rp_tl 592 \tl_const:Nn \c__unravel_lp_tl { ( }
\c__unravel_eq_tl 593 \tl_const:Nn \c__unravel_rp_tl { ) }
\c__unravel_comma_tl 594 \tl_const:Nn \c__unravel_eq_tl { = }
\c__unravel_point_tl 595 \tl_const:Nn \c__unravel_comma_tl { , }
596 \tl_const:Nn \c__unravel_point_tl { . }

```

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

```

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

```

(End definition for `\c__unravel_frozen_relax_gtl`.)

2.2.6 TeX parameters

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

```
598 \int_new:N \g__unravel_mag_set_int
```

(End definition for `\g__unravel_mag_set_int`.)

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

```

599 \cs_new_protected:Npn \__unravel_prepare_mag:
600 {
601   \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
602   {
603     \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
604     {
605       \__unravel_tex_error:nn { incompatible-mag } { }
606       \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
607     }
608   }
609   \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
610   {
611     \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
612     \int_gset:Nn \__unravel_mag: { 1000 }
613   }
614   \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
615 }

```

(End definition for `__unravel_prepare_mag:`.)

2.3 Numeric codes

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

```

\__unravel_tex_const:nn
  \__unravel_tex_use:n
616 \cs_new_protected:Npn \__unravel_tex_const:nn #1#2
617   { \int_const:cn { c__unravel_tex_#1_int } {#2} }
618 \cs_new:Npn \__unravel_tex_use:n #1 { \int_use:c { c__unravel_tex_#1_int } }

```

(End definition for __unravel_tex_const:nn and __unravel_tex_use:n.)

```

\__unravel_tex_primitive:nnn
  \__unravel_tex_primitive_pdf:nnn
619 \cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
620   {
621     \tl_const:cx { c__unravel_tex_#1_tl }
622     { { \__unravel_tex_use:n {#2} } {#3} }
623   }
624 \cs_new_protected:Npn \__unravel_tex_primitive_pdf:nnn #1#2#3
625   {
626     \sys_if_engine_pdftex:F
627     { \__unravel_tex_primitive:nnn {#1} {#2} {#3} }
628     \__unravel_tex_primitive:nnn { pdf #1 } {#2} {#3}
629   }

```

(End definition for __unravel_tex_primitive:nnn and __unravel_tex_primitive_pdf:nnn.)

```

\__unravel_new_tex_cmd:nn
\__unravel_new_eq_tex_cmd:nn
630 \cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
631   {
632     \cs_new_protected:cpn
633     { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
634   }
635 \cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
636   {
637     \cs_new_eq:cc
638     { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
639     { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
640   }

```

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

```

\__unravel_new_tex_expandable:nn
641 \cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
642   {
643     \cs_new_protected:cpn
644     { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
645   }

```

(End definition for __unravel_new_tex_expandable:nn.)

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

```

646 \__unravel_tex_const:nn { relax } { 0 }
647 \__unravel_tex_const:nn { begin-group_char } { 1 }
648 \__unravel_tex_const:nn { end-group_char } { 2 }

```

```

649 \_unravel_tex_const:nn { math_char } { 3 }
650 \_unravel_tex_const:nn { tab_mark } { 4 }
651 \_unravel_tex_const:nn { alignment_char } { 4 }
652 \_unravel_tex_const:nn { car_ret } { 5 }
653 \_unravel_tex_const:nn { macro_char } { 6 }
654 \_unravel_tex_const:nn { superscript_char } { 7 }
655 \_unravel_tex_const:nn { subscript_char } { 8 }
656 \_unravel_tex_const:nn { endv } { 9 }
657 \_unravel_tex_const:nn { blank_char } { 10 }
658 \_unravel_tex_const:nn { the_char } { 11 }
659 \_unravel_tex_const:nn { other_char } { 12 }
660 \_unravel_tex_const:nn { par_end } { 13 }
661 \_unravel_tex_const:nn { stop } { 14 }
662 \_unravel_tex_const:nn { delim_num } { 15 }
663 \_unravel_tex_const:nn { max_char_code } { 15 }
664 \_unravel_tex_const:nn { char_num } { 16 }
665 \_unravel_tex_const:nn { math_char_num } { 17 }
666 \_unravel_tex_const:nn { mark } { 18 }
667 \_unravel_tex_const:nn { xray } { 19 }
668 \_unravel_tex_const:nn { make_box } { 20 }
669 \_unravel_tex_const:nn { hmove } { 21 }
670 \_unravel_tex_const:nn { vmove } { 22 }
671 \_unravel_tex_const:nn { un_hbox } { 23 }
672 \_unravel_tex_const:nn { un_vbox } { 24 }
673 \_unravel_tex_const:nn { remove_item } { 25 }
674 \_unravel_tex_const:nn { hskip } { 26 }
675 \_unravel_tex_const:nn { vskip } { 27 }
676 \_unravel_tex_const:nn { mskip } { 28 }
677 \_unravel_tex_const:nn { kern } { 29 }
678 \_unravel_tex_const:nn { mkern } { 30 }
679 \_unravel_tex_const:nn { leader_ship } { 31 }
680 \_unravel_tex_const:nn { halign } { 32 }
681 \_unravel_tex_const:nn { valign } { 33 }
682 \_unravel_tex_const:nn { no_align } { 34 }
683 \_unravel_tex_const:nn { vrule } { 35 }
684 \_unravel_tex_const:nn { hrule } { 36 }
685 \_unravel_tex_const:nn { insert } { 37 }
686 \_unravel_tex_const:nn { vadjust } { 38 }
687 \_unravel_tex_const:nn { ignore_spaces } { 39 }
688 \_unravel_tex_const:nn { after_assignment } { 40 }
689 \_unravel_tex_const:nn { after_group } { 41 }
690 \_unravel_tex_const:nn { break_penalty } { 42 }
691 \_unravel_tex_const:nn { start_par } { 43 }
692 \_unravel_tex_const:nn { ital_corr } { 44 }
693 \_unravel_tex_const:nn { accent } { 45 }
694 \_unravel_tex_const:nn { math_accent } { 46 }
695 \_unravel_tex_const:nn { discretionary } { 47 }
696 \_unravel_tex_const:nn { eq_no } { 48 }
697 \_unravel_tex_const:nn { left_right } { 49 }
698 \_unravel_tex_const:nn { math_comp } { 50 }
699 \_unravel_tex_const:nn { limit_switch } { 51 }
700 \_unravel_tex_const:nn { above } { 52 }
701 \_unravel_tex_const:nn { math_style } { 53 }
702 \_unravel_tex_const:nn { math_choice } { 54 }

```

```

703 \_unravel_tex_const:nn { non_script } { 55 }
704 \_unravel_tex_const:nn { vcenter } { 56 }
705 \_unravel_tex_const:nn { case_shift } { 57 }
706 \_unravel_tex_const:nn { message } { 58 }
707 \_unravel_tex_const:nn { extension } { 59 }
708 \_unravel_tex_const:nn { in_stream } { 60 }
709 \_unravel_tex_const:nn { begin_group } { 61 }
710 \_unravel_tex_const:nn { end_group } { 62 }
711 \_unravel_tex_const:nn { omit } { 63 }
712 \_unravel_tex_const:nn { ex_space } { 64 }
713 \_unravel_tex_const:nn { no_boundary } { 65 }
714 \_unravel_tex_const:nn { radical } { 66 }
715 \_unravel_tex_const:nn { end_cs_name } { 67 }
716 \_unravel_tex_const:nn { min_internal } { 68 }
717 \_unravel_tex_const:nn { char_given } { 68 }
718 \_unravel_tex_const:nn { math_given } { 69 }
719 \_unravel_tex_const:nn { last_item } { 70 }
720 \_unravel_tex_const:nn { max_non_prefixed_command } { 70 }
721 \_unravel_tex_const:nn { toks_register } { 71 }
722 \_unravel_tex_const:nn { assign_toks } { 72 }
723 \_unravel_tex_const:nn { assign_int } { 73 }
724 \_unravel_tex_const:nn { assign_dimen } { 74 }
725 \_unravel_tex_const:nn { assign_glue } { 75 }
726 \_unravel_tex_const:nn { assign_mu_glue } { 76 }
727 \_unravel_tex_const:nn { assign_font_dimen } { 77 }
728 \_unravel_tex_const:nn { assign_font_int } { 78 }
729 \_unravel_tex_const:nn { set_aux } { 79 }
730 \_unravel_tex_const:nn { set_prev_graf } { 80 }
731 \_unravel_tex_const:nn { set_page_dimen } { 81 }
732 \_unravel_tex_const:nn { set_page_int } { 82 }
733 \_unravel_tex_const:nn { set_box_dimen } { 83 }
734 \_unravel_tex_const:nn { set_shape } { 84 }
735 \_unravel_tex_const:nn { def_code } { 85 }
736 \_unravel_tex_const:nn { def_family } { 86 }
737 \_unravel_tex_const:nn { set_font } { 87 }
738 \_unravel_tex_const:nn { def_font } { 88 }
739 \_unravel_tex_const:nn { register } { 89 }
740 \_unravel_tex_const:nn { max_internal } { 89 }
741 \_unravel_tex_const:nn { advance } { 90 }
742 \_unravel_tex_const:nn { multiply } { 91 }
743 \_unravel_tex_const:nn { divide } { 92 }
744 \_unravel_tex_const:nn { prefix } { 93 }
745 \_unravel_tex_const:nn { let } { 94 }
746 \_unravel_tex_const:nn { shorthand_def } { 95 }
747 \_unravel_tex_const:nn { read_to_cs } { 96 }
748 \_unravel_tex_const:nn { def } { 97 }
749 \_unravel_tex_const:nn { set_box } { 98 }
750 \_unravel_tex_const:nn { hyph_data } { 99 }
751 \_unravel_tex_const:nn { set_interaction } { 100 }
752 \_unravel_tex_const:nn { letterspace_font } { 101 }
753 \_unravel_tex_const:nn { pdf_copy_font } { 102 }
754 \_unravel_tex_const:nn { max_command } { 102 }
755 \_unravel_tex_const:nn { undefined_cs } { 103 }
756 \_unravel_tex_const:nn { expand_after } { 104 }

```

```

757 \_unravel_tex_const:nn { no_expand } { 105 }
758 \_unravel_tex_const:nn { input } { 106 }
759 \_unravel_tex_const:nn { if_test } { 107 }
760 \_unravel_tex_const:nn { fi_or_else } { 108 }
761 \_unravel_tex_const:nn { cs_name } { 109 }
762 \_unravel_tex_const:nn { convert } { 110 }
763 \_unravel_tex_const:nn { the } { 111 }
764 \_unravel_tex_const:nn { top_bot_mark } { 112 }
765 \_unravel_tex_const:nn { call } { 113 }
766 \_unravel_tex_const:nn { end_template } { 117 }

```

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

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

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

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

```

767 \sys_if_engine luatex:T
768 {
769   \_unravel_tex_primitive:nnn
770     { \exp_after:wN \use_none:n \token_to_meaning:N \_unravel_special_relax: }
771     { relax } { 1 }
772 }
773 \_unravel_tex_primitive:nnn { relax } { relax } { 256 }
774 \_unravel_tex_primitive:nnn { span } { tab_mark } { 256 }
775 \_unravel_tex_primitive:nnn { cr } { car_ret } { 257 }
776 \_unravel_tex_primitive:nnn { crcr } { car_ret } { 258 }
777 \_unravel_tex_primitive:nnn { par } { par_end } { 256 }
778 \_unravel_tex_primitive:nnn { end } { stop } { 0 }
779 \_unravel_tex_primitive:nnn { dump } { stop } { 1 }
780 \_unravel_tex_primitive:nnn { delimiter } { delim_num } { 0 }
781 \_unravel_tex_primitive:nnn { char } { char_num } { 0 }
782 \_unravel_tex_primitive:nnn { mathchar } { math_char_num } { 0 }
783 \_unravel_tex_primitive:nnn { mark } { mark } { 0 }
784 \_unravel_tex_primitive:nnn { marks } { mark } { 5 }
785 \_unravel_tex_primitive:nnn { show } { xray } { 0 }
786 \_unravel_tex_primitive:nnn { showbox } { xray } { 1 }
787 \_unravel_tex_primitive:nnn { showthe } { xray } { 2 }
788 \_unravel_tex_primitive:nnn { showlists } { xray } { 3 }
789 \_unravel_tex_primitive:nnn { showgroups } { xray } { 4 }

```

```

790 \_unravel_tex_primitive:nnn { showtokens } { xray } { 5 }
791 \_unravel_tex_primitive:nnn { showifs } { xray } { 6 }
792 \_unravel_tex_primitive:nnn { box } { make_box } { 0 }
793 \_unravel_tex_primitive:nnn { copy } { make_box } { 1 }
794 \_unravel_tex_primitive:nnn { lastbox } { make_box } { 2 }
795 \_unravel_tex_primitive:nnn { vsplit } { make_box } { 3 }
796 \_unravel_tex_primitive:nnn { vtop } { make_box } { 4 }
797 \_unravel_tex_primitive:nnn { vbox } { make_box } { 5 }
798 \_unravel_tex_primitive:nnn { hbox } { make_box } { 106 }
799 \_unravel_tex_primitive:nnn { moveright } { hmove } { 0 }
800 \_unravel_tex_primitive:nnn { moveleft } { hmove } { 1 }
801 \_unravel_tex_primitive:nnn { lower } { vmove } { 0 }
802 \_unravel_tex_primitive:nnn { raise } { vmove } { 1 }
803 \_unravel_tex_primitive:nnn { unhbox } { un_hbox } { 0 }
804 \_unravel_tex_primitive:nnn { unhcopy } { un_hbox } { 1 }
805 \_unravel_tex_primitive:nnn { unvbox } { un_vbox } { 0 }
806 \_unravel_tex_primitive:nnn { unvcopy } { un_vbox } { 1 }
807 \_unravel_tex_primitive:nnn { pagediscards } { un_vbox } { 2 }
808 \_unravel_tex_primitive:nnn { splitdiscards } { un_vbox } { 3 }
809 \_unravel_tex_primitive:nnn { unpenalty } { remove_item } { 12 }
810 \_unravel_tex_primitive:nnn { unkern } { remove_item } { 11 }
811 \_unravel_tex_primitive:nnn { unskip } { remove_item } { 10 }
812 \_unravel_tex_primitive:nnn { hfil } { hskip } { 0 }
813 \_unravel_tex_primitive:nnn { hfill } { hskip } { 1 }
814 \_unravel_tex_primitive:nnn { hss } { hskip } { 2 }
815 \_unravel_tex_primitive:nnn { hfilneg } { hskip } { 3 }
816 \_unravel_tex_primitive:nnn { hskip } { hskip } { 4 }
817 \_unravel_tex_primitive:nnn { vfil } { vskip } { 0 }
818 \_unravel_tex_primitive:nnn { vfill } { vskip } { 1 }
819 \_unravel_tex_primitive:nnn { vss } { vskip } { 2 }
820 \_unravel_tex_primitive:nnn { vfilneg } { vskip } { 3 }
821 \_unravel_tex_primitive:nnn { vskip } { vskip } { 4 }
822 \_unravel_tex_primitive:nnn { mskip } { mskip } { 5 }
823 \_unravel_tex_primitive:nnn { kern } { kern } { 1 }
824 \_unravel_tex_primitive:nnn { mkern } { mkern } { 99 }
825 \_unravel_tex_primitive:nnn { shipout } { leader_ship } { 99 }
826 \_unravel_tex_primitive:nnn { leaders } { leader_ship } { 100 }
827 \_unravel_tex_primitive:nnn { cleaders } { leader_ship } { 101 }
828 \_unravel_tex_primitive:nnn { xleaders } { leader_ship } { 102 }
829 \_unravel_tex_primitive:nnn { halign } { halign } { 0 }
830 \_unravel_tex_primitive:nnn { valign } { valign } { 0 }
831 \_unravel_tex_primitive:nnn { beginL } { valign } { 4 }
832 \_unravel_tex_primitive:nnn { endL } { valign } { 5 }
833 \_unravel_tex_primitive:nnn { beginR } { valign } { 8 }
834 \_unravel_tex_primitive:nnn { endR } { valign } { 9 }
835 \_unravel_tex_primitive:nnn { noalign } { no_align } { 0 }
836 \_unravel_tex_primitive:nnn { vrule } { vrule } { 0 }
837 \_unravel_tex_primitive:nnn { hrule } { hrule } { 0 }
838 \_unravel_tex_primitive:nnn { insert } { insert } { 0 }
839 \_unravel_tex_primitive:nnn { vadjust } { vadjust } { 0 }
840 \_unravel_tex_primitive:nnn { ignorespaces } { ignore_spaces } { 0 }
841 \_unravel_tex_primitive:nnn { afterassignment } { after_assignment } { 0 }
842 \_unravel_tex_primitive:nnn { aftergroup } { after_group } { 0 }
843 \_unravel_tex_primitive:nnn { penalty } { break_penalty } { 0 }

```

```

844 \_unravel_tex_primitive:nnn { indent } { start_par } { 1 }
845 \_unravel_tex_primitive:nnn { noindent } { start_par } { 0 }
846 \_unravel_tex_primitive:nnn { quitvmode } { start_par } { 2 }
847 \_unravel_tex_primitive:nnn { / } { ital_corr } { 0 }
848 \_unravel_tex_primitive:nnn { accent } { accent } { 0 }
849 \_unravel_tex_primitive:nnn { mathaccent } { math_accent } { 0 }
850 \sys_if_engine_luatex:T
851 {
852 \_unravel_tex_primitive:nnn
853 { explicitdiscretionary } { discretionary } { 1 }
854 }
855 \_unravel_tex_primitive:nnn { - } { discretionary } { 1 }
856 \_unravel_tex_primitive:nnn { discretionary } { discretionary } { 0 }
857 \_unravel_tex_primitive:nnn { eqno } { eq_no } { 0 }
858 \_unravel_tex_primitive:nnn { leqno } { eq_no } { 1 }
859 \_unravel_tex_primitive:nnn { left } { left_right } { 30 }
860 \_unravel_tex_primitive:nnn { right } { left_right } { 31 }
861 \_unravel_tex_primitive:nnn { middle } { left_right } { 17 }
862 \_unravel_tex_primitive:nnn { mathord } { math_comp } { 16 }
863 \_unravel_tex_primitive:nnn { mathop } { math_comp } { 17 }
864 \_unravel_tex_primitive:nnn { mathbin } { math_comp } { 18 }
865 \_unravel_tex_primitive:nnn { mathrel } { math_comp } { 19 }
866 \_unravel_tex_primitive:nnn { mathopen } { math_comp } { 20 }
867 \_unravel_tex_primitive:nnn { mathclose } { math_comp } { 21 }
868 \_unravel_tex_primitive:nnn { mathpunct } { math_comp } { 22 }
869 \_unravel_tex_primitive:nnn { mathinner } { math_comp } { 23 }
870 \_unravel_tex_primitive:nnn { underline } { math_comp } { 26 }
871 \_unravel_tex_primitive:nnn { overline } { math_comp } { 27 }
872 \_unravel_tex_primitive:nnn { displaylimits } { limit_switch } { 0 }
873 \_unravel_tex_primitive:nnn { limits } { limit_switch } { 1 }
874 \_unravel_tex_primitive:nnn { nolimits } { limit_switch } { 2 }
875 \_unravel_tex_primitive:nnn { above } { above } { 0 }
876 \_unravel_tex_primitive:nnn { over } { above } { 1 }
877 \_unravel_tex_primitive:nnn { atop } { above } { 2 }
878 \_unravel_tex_primitive:nnn { abovewithdelims } { above } { 3 }
879 \_unravel_tex_primitive:nnn { overwithdelims } { above } { 4 }
880 \_unravel_tex_primitive:nnn { atopwithdelims } { above } { 5 }
881 \_unravel_tex_primitive:nnn { displaystyle } { math_style } { 0 }
882 \_unravel_tex_primitive:nnn { textstyle } { math_style } { 2 }
883 \_unravel_tex_primitive:nnn { scriptstyle } { math_style } { 4 }
884 \_unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
885 \_unravel_tex_primitive:nnn { mathchoice } { math_choice } { 0 }
886 \_unravel_tex_primitive:nnn { nonscript } { non_script } { 0 }
887 \_unravel_tex_primitive:nnn { vcenter } { vcenter } { 0 }
888 \_unravel_tex_primitive:nnn { lowercase } { case_shift } { 256 }
889 \_unravel_tex_primitive:nnn { uppercase } { case_shift } { 512 }
890 \_unravel_tex_primitive:nnn { message } { message } { 0 }
891 \_unravel_tex_primitive:nnn { errmessage } { message } { 1 }
892 \_unravel_tex_primitive:nnn { openout } { extension } { 0 }
893 \_unravel_tex_primitive:nnn { write } { extension } { 1 }
894 \_unravel_tex_primitive:nnn { closeout } { extension } { 2 }
895 \_unravel_tex_primitive:nnn { special } { extension } { 3 }
896 \_unravel_tex_primitive:nnn { immediate } { extension } { 4 }
897 \_unravel_tex_primitive:nnn { setlanguage } { extension } { 5 }

```

```

898 \_unravel_tex_primitive_pdf:nnn { literal      } { extension } { 6 }
899 \_unravel_tex_primitive_pdf:nnn { obj         } { extension } { 7 }
900 \_unravel_tex_primitive_pdf:nnn { refobj    } { extension } { 8 }
901 \_unravel_tex_primitive_pdf:nnn { xform     } { extension } { 9 }
902 \_unravel_tex_primitive_pdf:nnn { refxform  } { extension } { 10 }
903 \_unravel_tex_primitive_pdf:nnn { ximage    } { extension } { 11 }
904 \_unravel_tex_primitive_pdf:nnn { refximage } { extension } { 12 }
905 \_unravel_tex_primitive_pdf:nnn { annot     } { extension } { 13 }
906 \_unravel_tex_primitive_pdf:nnn { startlink } { extension } { 14 }
907 \_unravel_tex_primitive_pdf:nnn { endlink   } { extension } { 15 }
908 \_unravel_tex_primitive_pdf:nnn { outline   } { extension } { 16 }
909 \_unravel_tex_primitive_pdf:nnn { dest      } { extension } { 17 }
910 \_unravel_tex_primitive_pdf:nnn { thread    } { extension } { 18 }
911 \_unravel_tex_primitive_pdf:nnn { startthread } { extension } { 19 }
912 \_unravel_tex_primitive_pdf:nnn { endthread  } { extension } { 20 }
913 \_unravel_tex_primitive_pdf:nnn { savepos   } { extension } { 21 }
914 \_unravel_tex_primitive_pdf:nnn { info      } { extension } { 22 }
915 \_unravel_tex_primitive_pdf:nnn { catalog   } { extension } { 23 }
916 \_unravel_tex_primitive_pdf:nnn { names     } { extension } { 24 }
917 \_unravel_tex_primitive_pdf:nnn { fontattr  } { extension } { 25 }
918 \_unravel_tex_primitive_pdf:nnn { includechars } { extension } { 26 }
919 \_unravel_tex_primitive_pdf:nnn { mapfile   } { extension } { 27 }
920 \_unravel_tex_primitive_pdf:nnn { mapline   } { extension } { 28 }
921 \_unravel_tex_primitive_pdf:nnn { trailer   } { extension } { 29 }
922 \_unravel_tex_primitive_pdf:nnn { resettimer } { extension } { 30 }
923 \_unravel_tex_primitive_pdf:nnn { fontexpand } { extension } { 31 }
924 \_unravel_tex_primitive_pdf:nnn { setrandomseed } { extension } { 32 }
925 \_unravel_tex_primitive_pdf:nnn { snaprefpoint } { extension } { 33 }
926 \_unravel_tex_primitive_pdf:nnn { snapy     } { extension } { 34 }
927 \_unravel_tex_primitive_pdf:nnn { snapycomp } { extension } { 35 }
928 \_unravel_tex_primitive_pdf:nnn { glyphtounicode } { extension } { 36 }
929 \_unravel_tex_primitive_pdf:nnn { colorstack } { extension } { 37 }
930 \_unravel_tex_primitive_pdf:nnn { setmatrix  } { extension } { 38 }
931 \_unravel_tex_primitive_pdf:nnn { save       } { extension } { 39 }
932 \_unravel_tex_primitive_pdf:nnn { restore    } { extension } { 40 }
933 \_unravel_tex_primitive_pdf:nnn { nobuiltintounicode } { extension } { 41 }
934 \_unravel_tex_primitive:nnn { openin      } { in_stream } { 1 }
935 \_unravel_tex_primitive:nnn { closein     } { in_stream } { 0 }
936 \_unravel_tex_primitive:nnn { begingroup  } { begin_group } { 0 }
937 \_unravel_tex_primitive:nnn { endgroup    } { end_group } { 0 }
938 \_unravel_tex_primitive:nnn { omit       } { omit } { 0 }
939 \_unravel_tex_primitive:nnn { ~         } { ex_space } { 0 }
940 \_unravel_tex_primitive:nnn { noboundary  } { no_boundary } { 0 }
941 \_unravel_tex_primitive:nnn { radical    } { radical } { 0 }
942 \_unravel_tex_primitive:nnn { endcsname  } { end_cs_name } { 0 }
943 \_unravel_tex_primitive:nnn { lastpenalty } { last_item } { 0 }
944 \_unravel_tex_primitive:nnn { lastkern   } { last_item } { 1 }
945 \_unravel_tex_primitive:nnn { lastskip   } { last_item } { 2 }
946 \_unravel_tex_primitive:nnn { lastnodetype } { last_item } { 3 }
947 \_unravel_tex_primitive:nnn { inputlineno } { last_item } { 4 }
948 \_unravel_tex_primitive:nnn { badness    } { last_item } { 5 }
949 \_unravel_tex_primitive_pdf:nnn { texversion } { last_item } { 6 }
950 \_unravel_tex_primitive_pdf:nnn { lastobj    } { last_item } { 7 }
951 \_unravel_tex_primitive_pdf:nnn { lastxform  } { last_item } { 8 }

```



```

952 \_unravel_tex_primitive_pdf:nnn { lastximage      } { last_item } { 9 }
953 \_unravel_tex_primitive_pdf:nnn { lastximagepages } { last_item } { 10 }
954 \_unravel_tex_primitive_pdf:nnn { lastannot      } { last_item } { 11 }
955 \_unravel_tex_primitive_pdf:nnn { lastxpos      } { last_item } { 12 }
956 \_unravel_tex_primitive_pdf:nnn { lasttypos   } { last_item } { 13 }
957 \_unravel_tex_primitive_pdf:nnn { retval       } { last_item } { 14 }
958 \_unravel_tex_primitive_pdf:nnn { lastximagecolordepth } { last_item } { 15 }
959 \_unravel_tex_primitive_pdf:nnn { elapsedtime  } { last_item } { 16 }
960 \_unravel_tex_primitive_pdf:nnn { shellescape } { last_item } { 17 }
961 \_unravel_tex_primitive_pdf:nnn { randomseed  } { last_item } { 18 }
962 \_unravel_tex_primitive_pdf:nnn { lastlink    } { last_item } { 19 }
963 \_unravel_tex_primitive:nnn { eTeXversion } { last_item } { 20 }
964 \_unravel_tex_primitive:nnn { currentgrouplevel } { last_item } { 21 }
965 \_unravel_tex_primitive:nnn { currentgroupstype } { last_item } { 22 }
966 \_unravel_tex_primitive:nnn { currentiflevel   } { last_item } { 23 }
967 \_unravel_tex_primitive:nnn { currentiftypetype } { last_item } { 24 }
968 \_unravel_tex_primitive:nnn { currentifbranch  } { last_item } { 25 }
969 \_unravel_tex_primitive:nnn { gluestretchorder } { last_item } { 26 }
970 \_unravel_tex_primitive:nnn { glueshrinkorder  } { last_item } { 27 }
971 \_unravel_tex_primitive:nnn { fontcharwd      } { last_item } { 28 }
972 \_unravel_tex_primitive:nnn { fontcharht     } { last_item } { 29 }
973 \_unravel_tex_primitive:nnn { fontchardp     } { last_item } { 30 }
974 \_unravel_tex_primitive:nnn { fontcharic     } { last_item } { 31 }
975 \_unravel_tex_primitive:nnn { parshapelength  } { last_item } { 32 }
976 \_unravel_tex_primitive:nnn { parshapeindent  } { last_item } { 33 }
977 \_unravel_tex_primitive:nnn { parshapedimen   } { last_item } { 34 }
978 \_unravel_tex_primitive:nnn { gluestretch     } { last_item } { 35 }
979 \_unravel_tex_primitive:nnn { glueshrink     } { last_item } { 36 }
980 \_unravel_tex_primitive:nnn { mutogluex      } { last_item } { 37 }
981 \_unravel_tex_primitive:nnn { gluetoemu     } { last_item } { 38 }
982 \_unravel_tex_primitive:nnn { numexpr       } { last_item } { 39 }
983 \_unravel_tex_primitive:nnn { dimexpr       } { last_item } { 40 }
984 \_unravel_tex_primitive:nnn { glueexpr      } { last_item } { 41 }
985 \_unravel_tex_primitive:nnn { muexpr       } { last_item } { 42 }
986 \_unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
987 \_unravel_tex_primitive:nnn { output      } { assign_toks } { 1 }
988 \_unravel_tex_primitive:nnn { everypar    } { assign_toks } { 2 }
989 \_unravel_tex_primitive:nnn { everymath   } { assign_toks } { 3 }
990 \_unravel_tex_primitive:nnn { everydisplay } { assign_toks } { 4 }
991 \_unravel_tex_primitive:nnn { everyhbox   } { assign_toks } { 5 }
992 \_unravel_tex_primitive:nnn { everyvbox   } { assign_toks } { 6 }
993 \_unravel_tex_primitive:nnn { everyjob    } { assign_toks } { 7 }
994 \_unravel_tex_primitive:nnn { everycr    } { assign_toks } { 8 }
995 \_unravel_tex_primitive:nnn { errhelp    } { assign_toks } { 9 }
996 \_unravel_tex_primitive_pdf:nnn { pagesattr   } { assign_toks } { 10 }
997 \_unravel_tex_primitive_pdf:nnn { pageattr    } { assign_toks } { 11 }
998 \_unravel_tex_primitive_pdf:nnn { pageresources } { assign_toks } { 12 }
999 \_unravel_tex_primitive_pdf:nnn { pkmode      } { assign_toks } { 13 }
1000 \_unravel_tex_primitive:nnn { everyeof    } { assign_toks } { 14 }
1001 \_unravel_tex_primitive:nnn { pretolerance } { assign_int } { 0 }
1002 \_unravel_tex_primitive:nnn { tolerance   } { assign_int } { 1 }
1003 \_unravel_tex_primitive:nnn { linepenalty } { assign_int } { 2 }
1004 \_unravel_tex_primitive:nnn { hyphenpenalty } { assign_int } { 3 }
1005 \_unravel_tex_primitive:nnn { exhyphenpenalty } { assign_int } { 4 }

```

```

1006 \_unravel_tex_primitive:nnn { clubpenalty } { assign_int } { 5 }
1007 \_unravel_tex_primitive:nnn { widowpenalty } { assign_int } { 6 }
1008 \_unravel_tex_primitive:nnn { displaywidowpenalty } { assign_int } { 7 }
1009 \_unravel_tex_primitive:nnn { brokenpenalty } { assign_int } { 8 }
1010 \_unravel_tex_primitive:nnn { binoppenalty } { assign_int } { 9 }
1011 \_unravel_tex_primitive:nnn { relpenalty } { assign_int } { 10 }
1012 \_unravel_tex_primitive:nnn { predisplaypenalty } { assign_int } { 11 }
1013 \_unravel_tex_primitive:nnn { postdisplaypenalty } { assign_int } { 12 }
1014 \_unravel_tex_primitive:nnn { interlinepenalty } { assign_int } { 13 }
1015 \_unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int } { 14 }
1016 \_unravel_tex_primitive:nnn { finalhyphendemerits } { assign_int } { 15 }
1017 \_unravel_tex_primitive:nnn { adjdemerits } { assign_int } { 16 }
1018 \_unravel_tex_primitive:nnn { mag } { assign_int } { 17 }
1019 \_unravel_tex_primitive:nnn { delimitfactor } { assign_int } { 18 }
1020 \_unravel_tex_primitive:nnn { looseness } { assign_int } { 19 }
1021 \_unravel_tex_primitive:nnn { time } { assign_int } { 20 }
1022 \_unravel_tex_primitive:nnn { day } { assign_int } { 21 }
1023 \_unravel_tex_primitive:nnn { month } { assign_int } { 22 }
1024 \_unravel_tex_primitive:nnn { year } { assign_int } { 23 }
1025 \_unravel_tex_primitive:nnn { showboxbreadth } { assign_int } { 24 }
1026 \_unravel_tex_primitive:nnn { showboxdepth } { assign_int } { 25 }
1027 \_unravel_tex_primitive:nnn { hbadness } { assign_int } { 26 }
1028 \_unravel_tex_primitive:nnn { vbadness } { assign_int } { 27 }
1029 \_unravel_tex_primitive:nnn { pausing } { assign_int } { 28 }
1030 \_unravel_tex_primitive:nnn { tracingonline } { assign_int } { 29 }
1031 \_unravel_tex_primitive:nnn { tracingmacros } { assign_int } { 30 }
1032 \_unravel_tex_primitive:nnn { tracingstats } { assign_int } { 31 }
1033 \_unravel_tex_primitive:nnn { tracingparagraphs } { assign_int } { 32 }
1034 \_unravel_tex_primitive:nnn { tracingpages } { assign_int } { 33 }
1035 \_unravel_tex_primitive:nnn { tracingoutput } { assign_int } { 34 }
1036 \_unravel_tex_primitive:nnn { tracinglostchars } { assign_int } { 35 }
1037 \_unravel_tex_primitive:nnn { tracingcommands } { assign_int } { 36 }
1038 \_unravel_tex_primitive:nnn { tracingrestores } { assign_int } { 37 }
1039 \_unravel_tex_primitive:nnn { uchyp } { assign_int } { 38 }
1040 \_unravel_tex_primitive:nnn { outputpenalty } { assign_int } { 39 }
1041 \_unravel_tex_primitive:nnn { maxdeadcycles } { assign_int } { 40 }
1042 \_unravel_tex_primitive:nnn { hangafter } { assign_int } { 41 }
1043 \_unravel_tex_primitive:nnn { floatingpenalty } { assign_int } { 42 }
1044 \_unravel_tex_primitive:nnn { globaldefs } { assign_int } { 43 }
1045 \_unravel_tex_primitive:nnn { fam } { assign_int } { 44 }
1046 \_unravel_tex_primitive:nnn { escapechar } { assign_int } { 45 }
1047 \_unravel_tex_primitive:nnn { defaultthyphenchar } { assign_int } { 46 }
1048 \_unravel_tex_primitive:nnn { defaultskewchar } { assign_int } { 47 }
1049 \_unravel_tex_primitive:nnn { endliness } { assign_int } { 48 }
1050 \_unravel_tex_primitive:nnn { newlinechar } { assign_int } { 49 }
1051 \_unravel_tex_primitive:nnn { language } { assign_int } { 50 }
1052 \_unravel_tex_primitive:nnn { lefthyphenmin } { assign_int } { 51 }
1053 \_unravel_tex_primitive:nnn { righthyphenmin } { assign_int } { 52 }
1054 \_unravel_tex_primitive:nnn { holdinginserts } { assign_int } { 53 }
1055 \_unravel_tex_primitive:nnn { errorcontextlines } { assign_int } { 54 }
1056 \_unravel_tex_primitive:nnn { pdfoutput } { assign_int } { 55 }
1057 \_unravel_tex_primitive_pdf:nnn { compresslevel } { assign_int } { 56 }
1058 \_unravel_tex_primitive_pdf:nnn { decimaldigits } { assign_int } { 57 }
1059 \_unravel_tex_primitive_pdf:nnn { movechars } { assign_int } { 58 }

```

```

1060 \_unravel_tex_primitive_pdf:nnn { imageresolution } { assign_int } { 59 }
1061 \_unravel_tex_primitive_pdf:nnn { pkresolution } { assign_int } { 60 }
1062 \_unravel_tex_primitive_pdf:nnn { uniqueresname } { assign_int } { 61 }
1063 \_unravel_tex_primitive_pdf:nnn
1064 { optionalwaysusepdfpagebox } { assign_int } { 62 }
1065 \_unravel_tex_primitive_pdf:nnn
1066 { optionpdfinclusionerrorlevel } { assign_int } { 63 }
1067 \_unravel_tex_primitive_pdf:nnn
1068 { optionpdfminorversion } { assign_int } { 64 }
1069 \_unravel_tex_primitive_pdf:nnn { minorversion } { assign_int } { 64 }
1070 \_unravel_tex_primitive_pdf:nnn { forcepagebox } { assign_int } { 65 }
1071 \_unravel_tex_primitive_pdf:nnn { pagebox } { assign_int } { 66 }
1072 \_unravel_tex_primitive_pdf:nnn
1073 { inclusionerrorlevel } { assign_int } { 67 }
1074 \_unravel_tex_primitive_pdf:nnn { gamma } { assign_int } { 68 }
1075 \_unravel_tex_primitive_pdf:nnn { imagegamma } { assign_int } { 69 }
1076 \_unravel_tex_primitive_pdf:nnn { imagehicolor } { assign_int } { 70 }
1077 \_unravel_tex_primitive_pdf:nnn { imageapplygamma } { assign_int } { 71 }
1078 \_unravel_tex_primitive_pdf:nnn { adjustspacing } { assign_int } { 72 }
1079 \_unravel_tex_primitive_pdf:nnn { protrudechars } { assign_int } { 73 }
1080 \_unravel_tex_primitive_pdf:nnn { tracingfonts } { assign_int } { 74 }
1081 \_unravel_tex_primitive_pdf:nnn { objcompresslevel } { assign_int } { 75 }
1082 \_unravel_tex_primitive_pdf:nnn
1083 { adjustinterwordglue } { assign_int } { 76 }
1084 \_unravel_tex_primitive_pdf:nnn { prependkern } { assign_int } { 77 }
1085 \_unravel_tex_primitive_pdf:nnn { appendkern } { assign_int } { 78 }
1086 \_unravel_tex_primitive_pdf:nnn { gentounicode } { assign_int } { 79 }
1087 \_unravel_tex_primitive_pdf:nnn { draftmode } { assign_int } { 80 }
1088 \_unravel_tex_primitive_pdf:nnn { inclusioncopyfonts } { assign_int } { 81 }
1089 \_unravel_tex_primitive:nnn { tracingassigns } { assign_int } { 82 }
1090 \_unravel_tex_primitive:nnn { tracinggroups } { assign_int } { 83 }
1091 \_unravel_tex_primitive:nnn { tracingifs } { assign_int } { 84 }
1092 \_unravel_tex_primitive:nnn { tracingscantokens } { assign_int } { 85 }
1093 \_unravel_tex_primitive:nnn { tracingnesting } { assign_int } { 86 }
1094 \_unravel_tex_primitive:nnn { predisplaydirection } { assign_int } { 87 }
1095 \_unravel_tex_primitive:nnn { lastlinefit } { assign_int } { 88 }
1096 \_unravel_tex_primitive:nnn { savingvdiscards } { assign_int } { 89 }
1097 \_unravel_tex_primitive:nnn { savinghyphcodes } { assign_int } { 90 }
1098 \_unravel_tex_primitive:nnn { TeXeTstate } { assign_int } { 91 }
1099 \_unravel_tex_primitive:nnn { parindent } { assign_dimen } { 0 }
1100 \_unravel_tex_primitive:nnn { mathsurround } { assign_dimen } { 1 }
1101 \_unravel_tex_primitive:nnn { lineskiplimit } { assign_dimen } { 2 }
1102 \_unravel_tex_primitive:nnn { hsize } { assign_dimen } { 3 }
1103 \_unravel_tex_primitive:nnn { vsize } { assign_dimen } { 4 }
1104 \_unravel_tex_primitive:nnn { maxdepth } { assign_dimen } { 5 }
1105 \_unravel_tex_primitive:nnn { splitmaxdepth } { assign_dimen } { 6 }
1106 \_unravel_tex_primitive:nnn { boxmaxdepth } { assign_dimen } { 7 }
1107 \_unravel_tex_primitive:nnn { hfuzz } { assign_dimen } { 8 }
1108 \_unravel_tex_primitive:nnn { vfuzz } { assign_dimen } { 9 }
1109 \_unravel_tex_primitive:nnn { delimitershortfall } { assign_dimen } { 10 }
1110 \_unravel_tex_primitive:nnn { nulldelimiterspace } { assign_dimen } { 11 }
1111 \_unravel_tex_primitive:nnn { scriptspace } { assign_dimen } { 12 }
1112 \_unravel_tex_primitive:nnn { predisplaysize } { assign_dimen } { 13 }
1113 \_unravel_tex_primitive:nnn { displaywidth } { assign_dimen } { 14 }

```

```

1114 \_unravel_tex_primitive:nnn { displayindent      } { assign_dimen } { 15 }
1115 \_unravel_tex_primitive:nnn { overfullrule     } { assign_dimen } { 16 }
1116 \_unravel_tex_primitive:nnn { hangindent     } { assign_dimen } { 17 }
1117 \_unravel_tex_primitive:nnn { hoffset      } { assign_dimen } { 18 }
1118 \_unravel_tex_primitive:nnn { voffset      } { assign_dimen } { 19 }
1119 \_unravel_tex_primitive:nnn { emergencystretch } { assign_dimen } { 20 }
1120 \_unravel_tex_primitive_pdf:nnn { horigin      } { assign_dimen } { 21 }
1121 \_unravel_tex_primitive_pdf:nnn { vorigin      } { assign_dimen } { 22 }
1122 \_unravel_tex_primitive_pdf:nnn { pagewidth    } { assign_dimen } { 23 }
1123 \_unravel_tex_primitive_pdf:nnn { pageheight   } { assign_dimen } { 24 }
1124 \_unravel_tex_primitive_pdf:nnn { linkmargin   } { assign_dimen } { 25 }
1125 \_unravel_tex_primitive_pdf:nnn { destmargin   } { assign_dimen } { 26 }
1126 \_unravel_tex_primitive_pdf:nnn { threadmargin } { assign_dimen } { 27 }
1127 \_unravel_tex_primitive_pdf:nnn { firstlineheight } { assign_dimen } { 28 }
1128 \_unravel_tex_primitive_pdf:nnn { lastlinedepth } { assign_dimen } { 29 }
1129 \_unravel_tex_primitive_pdf:nnn { eachlineheight } { assign_dimen } { 30 }
1130 \_unravel_tex_primitive_pdf:nnn { eachlinedepth } { assign_dimen } { 31 }
1131 \_unravel_tex_primitive_pdf:nnn { ignoreddimen } { assign_dimen } { 32 }
1132 \_unravel_tex_primitive_pdf:nnn { pxdimen      } { assign_dimen } { 33 }
1133 \_unravel_tex_primitive:nnn { lineskip     } { assign_glue } { 0 }
1134 \_unravel_tex_primitive:nnn { baselineskip } { assign_glue } { 1 }
1135 \_unravel_tex_primitive:nnn { parskip      } { assign_glue } { 2 }
1136 \_unravel_tex_primitive:nnn { abovedisplayskip } { assign_glue } { 3 }
1137 \_unravel_tex_primitive:nnn { belowdisplayskip } { assign_glue } { 4 }
1138 \_unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
1139 \_unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
1140 \_unravel_tex_primitive:nnn { leftskip     } { assign_glue } { 7 }
1141 \_unravel_tex_primitive:nnn { rightskip    } { assign_glue } { 8 }
1142 \_unravel_tex_primitive:nnn { topskip      } { assign_glue } { 9 }
1143 \_unravel_tex_primitive:nnn { splittopskip } { assign_glue } { 10 }
1144 \_unravel_tex_primitive:nnn { tabskip      } { assign_glue } { 11 }
1145 \_unravel_tex_primitive:nnn { spaceskip    } { assign_glue } { 12 }
1146 \_unravel_tex_primitive:nnn { xspaceskip   } { assign_glue } { 13 }
1147 \_unravel_tex_primitive:nnn { parfillskip  } { assign_glue } { 14 }
1148 \_unravel_tex_primitive:nnn { thinmuskip   } { assign_mu_glue } { 15 }
1149 \_unravel_tex_primitive:nnn { medmuskip    } { assign_mu_glue } { 16 }
1150 \_unravel_tex_primitive:nnn { thickmuskip  } { assign_mu_glue } { 17 }
1151 \_unravel_tex_primitive:nnn { fontdimen    } { assign_font_dimen } { 0 }
1152 \_unravel_tex_primitive:nnn { hyphenchar   } { assign_font_int } { 0 }
1153 \_unravel_tex_primitive:nnn { skewchar     } { assign_font_int } { 1 }
1154 \_unravel_tex_primitive:nnn { lpcode       } { assign_font_int } { 2 }
1155 \_unravel_tex_primitive:nnn { rpcode       } { assign_font_int } { 3 }
1156 \_unravel_tex_primitive:nnn { efcodes     } { assign_font_int } { 4 }
1157 \_unravel_tex_primitive:nnn { tagcode      } { assign_font_int } { 5 }
1158 \_unravel_tex_primitive_pdf:nnn { noligatures  } { assign_font_int } { 6 }
1159 \_unravel_tex_primitive:nnn { knbscode     } { assign_font_int } { 7 }
1160 \_unravel_tex_primitive:nnn { stbscode     } { assign_font_int } { 8 }
1161 \_unravel_tex_primitive:nnn { shbscode     } { assign_font_int } { 9 }
1162 \_unravel_tex_primitive:nnn { knbccode     } { assign_font_int } { 10 }
1163 \_unravel_tex_primitive:nnn { knaccodes    } { assign_font_int } { 11 }
1164 \_unravel_tex_primitive:nnn { spacefactor  } { set_aux } { 102 }
1165 \_unravel_tex_primitive:nnn { prevdepth    } { set_aux } { 1 }
1166 \_unravel_tex_primitive:nnn { prevgraf     } { set_prev_graf } { 0 }
1167 \_unravel_tex_primitive:nnn { pagegoal     } { set_page_dimen } { 0 }

```

```

1168 \_unravel_tex_primitive:nnn { pagetotal      } { set_page_dimen } { 1 }
1169 \_unravel_tex_primitive:nnn { pagestretch  } { set_page_dimen } { 2 }
1170 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 3 }
1171 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 4 }
1172 \_unravel_tex_primitive:nnn { pagefillstretch } { set_page_dimen } { 5 }
1173 \_unravel_tex_primitive:nnn { pageshrink     } { set_page_dimen } { 6 }
1174 \_unravel_tex_primitive:nnn { pagedepth     } { set_page_dimen } { 7 }
1175 \_unravel_tex_primitive:nnn { deadcycles    } { set_page_int   } { 0 }
1176 \_unravel_tex_primitive:nnn { insertpenalties } { set_page_int   } { 1 }
1177 \_unravel_tex_primitive:nnn { interactionmode } { set_page_int   } { 2 }
1178 \_unravel_tex_primitive:nnn { wd            } { set_box_dimen  } { 1 }
1179 \_unravel_tex_primitive:nnn { dp            } { set_box_dimen  } { 2 }
1180 \_unravel_tex_primitive:nnn { ht            } { set_box_dimen  } { 3 }
1181 \_unravel_tex_primitive:nnn { parshape     } { set_shape      } { 0 }
1182 \_unravel_tex_primitive:nnn { interlinepenalties } { set_shape      } { 1 }
1183 \_unravel_tex_primitive:nnn { clubpenalties } { set_shape      } { 2 }
1184 \_unravel_tex_primitive:nnn { widowpenalties } { set_shape      } { 3 }
1185 \_unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape      } { 4 }
1186 \_unravel_tex_primitive:nnn { catcode      } { def_code       } { 0 }
1187 \_unravel_tex_primitive:nnn { lccode       } { def_code       } { 256 }
1188 \_unravel_tex_primitive:nnn { uccode       } { def_code       } { 512 }
1189 \_unravel_tex_primitive:nnn { sfcode       } { def_code       } { 768 }
1190 \_unravel_tex_primitive:nnn { mathcode     } { def_code       } { 1024 }
1191 \_unravel_tex_primitive:nnn { delcode      } { def_code       } { 1591 }
1192 \_unravel_tex_primitive:nnn { textfont     } { def_family     } { -48 }
1193 \_unravel_tex_primitive:nnn { scriptfont   } { def_family     } { -32 }
1194 \_unravel_tex_primitive:nnn { scriptscriptfont } { def_family     } { -16 }
1195 \_unravel_tex_primitive:nnn { nullfont     } { set_font       } { 0 }
1196 \_unravel_tex_primitive:nnn { font         } { def_font       } { 0 }
1197 \_unravel_tex_primitive:nnn { count       } { register       } { 1 000 000 }
1198 \_unravel_tex_primitive:nnn { dimen       } { register       } { 2 000 000 }
1199 \_unravel_tex_primitive:nnn { skip        } { register       } { 3 000 000 }
1200 \_unravel_tex_primitive:nnn { muskip      } { register       } { 4 000 000 }
1201 \_unravel_tex_primitive:nnn { advance     } { advance        } { 0 }
1202 \_unravel_tex_primitive:nnn { multiply    } { multiply       } { 0 }
1203 \_unravel_tex_primitive:nnn { divide      } { divide         } { 0 }
1204 \_unravel_tex_primitive:nnn { long        } { prefix         } { 1 }
1205 \_unravel_tex_primitive:nnn { outer       } { prefix         } { 2 }
1206 \_unravel_tex_primitive:nnn { global      } { prefix         } { 4 }
1207 \_unravel_tex_primitive:nnn { protected   } { prefix         } { 8 }
1208 \_unravel_tex_primitive:nnn { let         } { let            } { 0 }
1209 \_unravel_tex_primitive:nnn { futurelet   } { let            } { 1 }
1210 \_unravel_tex_primitive:nnn { chardef     } { shorthand_def  } { 0 }
1211 \_unravel_tex_primitive:nnn { mathchardef } { shorthand_def  } { 1 }
1212 \_unravel_tex_primitive:nnn { countdef    } { shorthand_def  } { 2 }
1213 \_unravel_tex_primitive:nnn { dimendef    } { shorthand_def  } { 3 }
1214 \_unravel_tex_primitive:nnn { skipdef     } { shorthand_def  } { 4 }
1215 \_unravel_tex_primitive:nnn { muskipdef   } { shorthand_def  } { 5 }
1216 \_unravel_tex_primitive:nnn { toksdef     } { shorthand_def  } { 6 }
1217 \_unravel_tex_primitive:nnn { read        } { read_to_cs     } { 0 }
1218 \_unravel_tex_primitive:nnn { readline    } { read_to_cs     } { 1 }
1219 \_unravel_tex_primitive:nnn { def         } { def            } { 0 }
1220 \_unravel_tex_primitive:nnn { gdef       } { def            } { 1 }
1221 \_unravel_tex_primitive:nnn { edef       } { def            } { 2 }

```

```

1222 \_unravel_tex_primitive:nnn { xdef } { def } { 3 }
1223 \_unravel_tex_primitive:nnn { setbox } { set_box } { 0 }
1224 \_unravel_tex_primitive:nnn { hyphenation } { hyph_data } { 0 }
1225 \_unravel_tex_primitive:nnn { patterns } { hyph_data } { 1 }
1226 \_unravel_tex_primitive:nnn { batchmode } { set_interaction } { 0 }
1227 \_unravel_tex_primitive:nnn { nonstopmode } { set_interaction } { 1 }
1228 \_unravel_tex_primitive:nnn { scrollmode } { set_interaction } { 2 }
1229 \_unravel_tex_primitive:nnn { errorstopmode } { set_interaction } { 3 }
1230 \_unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
1231 \_unravel_tex_primitive_pdf:nnn { copyfont } { pdf_copy_font } { 0 }
1232 \_unravel_tex_primitive:nnn { undefined } { undefined_cs } { 0 }
1233 \_unravel_tex_primitive:nnn { ndefined } { undefined_cs } { 0 }
1234 \_unravel_tex_primitive:nnn { expandafter } { expand_after } { 0 }
1235 \_unravel_tex_primitive:nnn { unless } { expand_after } { 1 }
1236 \_unravel_tex_primitive_pdf:nnn { primitive } { no_expand } { 1 }
1237 \_unravel_tex_primitive:nnn { noexpand } { no_expand } { 0 }
1238 \_unravel_tex_primitive:nnn { input } { input } { 0 }
1239 \_unravel_tex_primitive:nnn { endinput } { input } { 1 }
1240 \_unravel_tex_primitive:nnn { scantokens } { input } { 2 }
1241 \_unravel_tex_primitive:nnn { if } { if_test } { 0 }
1242 \_unravel_tex_primitive:nnn { ifcat } { if_test } { 1 }
1243 \_unravel_tex_primitive:nnn { ifnum } { if_test } { 2 }
1244 \_unravel_tex_primitive:nnn { ifdim } { if_test } { 3 }
1245 \_unravel_tex_primitive:nnn { ifodd } { if_test } { 4 }
1246 \_unravel_tex_primitive:nnn { ifvmode } { if_test } { 5 }
1247 \_unravel_tex_primitive:nnn { ifhmode } { if_test } { 5 }
1248 \_unravel_tex_primitive:nnn { ifmmode } { if_test } { 5 }
1249 \_unravel_tex_primitive:nnn { ifinner } { if_test } { 5 }
1250 \_unravel_tex_primitive:nnn { ifvoid } { if_test } { 9 }
1251 \_unravel_tex_primitive:nnn { ifhbox } { if_test } { 9 }
1252 \_unravel_tex_primitive:nnn { ifvbox } { if_test } { 9 }
1253 \_unravel_tex_primitive:nnn { ifx } { if_test } { 12 }
1254 \_unravel_tex_primitive:nnn { ifeof } { if_test } { 13 }
1255 \_unravel_tex_primitive:nnn { iftrue } { if_test } { 14 }
1256 \_unravel_tex_primitive:nnn { iffalse } { if_test } { 15 }
1257 \_unravel_tex_primitive:nnn { ifcase } { if_test } { 16 }
1258 \_unravel_tex_primitive:nnn { ifdefined } { if_test } { 17 }
1259 \_unravel_tex_primitive:nnn { ifcsname } { if_test } { 18 }
1260 \_unravel_tex_primitive:nnn { iffontchar } { if_test } { 19 }
1261 \_unravel_tex_primitive:nnn { ifincsname } { if_test } { 20 }
1262 \_unravel_tex_primitive:nnn { ifprimitive } { if_test } { 21 }
1263 \_unravel_tex_primitive:nnn { ifpdfprimitive } { if_test } { 21 }
1264 \_unravel_tex_primitive:nnn { ifabsnum } { if_test } { 22 }
1265 \_unravel_tex_primitive:nnn { ifpdfabsnum } { if_test } { 22 }
1266 \_unravel_tex_primitive:nnn { ifabsdim } { if_test } { 23 }
1267 \_unravel_tex_primitive:nnn { ifpdfabsdim } { if_test } { 23 }
1268 \bool_if:nT { \sys_if_engine_ptex_p: || \sys_if_engine_uptex_p: }
1269 {
1270 \_unravel_tex_primitive:nnn { iftdir } { if_test } { 5 }
1271 \_unravel_tex_primitive:nnn { ifydir } { if_test } { 5 }
1272 \_unravel_tex_primitive:nnn { ifddir } { if_test } { 5 }
1273 \_unravel_tex_primitive:nnn { ifmdir } { if_test } { 5 }
1274 \_unravel_tex_primitive:nnn { iftbox } { if_test } { 9 }
1275 \_unravel_tex_primitive:nnn { ifybox } { if_test } { 9 }

```

```

1276     \_unravel_tex_primitive:nnn { ifdbox           } { if_test } { 9 }
1277     \_unravel_tex_primitive:nnn { ifmbox           } { if_test } { 9 }
1278     \_unravel_tex_primitive:nnn { ifjfont          } { if_test } { 24 }
1279     \_unravel_tex_primitive:nnn { iftfont          } { if_test } { 24 }
1280   }
1281 \_unravel_tex_primitive:nnn { fi                   } { fi_or_else } { 2 }
1282 \_unravel_tex_primitive:nnn { else                 } { fi_or_else } { 3 }
1283 \_unravel_tex_primitive:nnn { or                   } { fi_or_else } { 4 }
1284 \_unravel_tex_primitive:nnn { csname               } { cs_name } { 0 }
1285 \_unravel_tex_primitive:nnn { lastnamedcs          } { cs_name } { 1 }
1286 \_unravel_tex_primitive:nnn { number               } { convert } { 0 }
1287 \_unravel_tex_primitive:nnn { romannumeral         } { convert } { 1 }
1288 \_unravel_tex_primitive:nnn { string               } { convert } { 2 }
1289 \_unravel_tex_primitive:nnn { meaning              } { convert } { 3 }
1290 \_unravel_tex_primitive:nnn { fontname             } { convert } { 4 }
1291 \_unravel_tex_primitive:nnn { eTeXrevision         } { convert } { 5 }
1292 \_unravel_tex_primitive_pdf:nnn { texrevision      } { convert } { 6 }
1293 \_unravel_tex_primitive_pdf:nnn { texbanner        } { convert } { 7 }
1294 \_unravel_tex_primitive_pdf:nnn { pdffontname     } { convert } { 8 }
1295 \_unravel_tex_primitive_pdf:nnn { fontobjnum      } { convert } { 9 }
1296 \_unravel_tex_primitive_pdf:nnn { fontsize        } { convert } { 10 }
1297 \_unravel_tex_primitive_pdf:nnn { pageref         } { convert } { 11 }
1298 \_unravel_tex_primitive_pdf:nnn { xformname       } { convert } { 12 }
1299 \_unravel_tex_primitive_pdf:nnn { escapestring    } { convert } { 13 }
1300 \_unravel_tex_primitive_pdf:nnn { escapename      } { convert } { 14 }
1301 \_unravel_tex_primitive:nnn { leftmarginkern     } { convert } { 15 }
1302 \_unravel_tex_primitive:nnn { rightmarginkern    } { convert } { 16 }
1303 \_unravel_tex_primitive_pdf:nnn { strcmp         } { convert } { 17 }
1304 \_unravel_tex_primitive_pdf:nnn { colorstackinit } { convert } { 18 }
1305 \_unravel_tex_primitive_pdf:nnn { escapehex      } { convert } { 19 }
1306 \_unravel_tex_primitive_pdf:nnn { unescapehex    } { convert } { 20 }
1307 \_unravel_tex_primitive_pdf:nnn { creationdate   } { convert } { 21 }
1308 \_unravel_tex_primitive_pdf:nnn { filemoddate    } { convert } { 22 }
1309 \_unravel_tex_primitive_pdf:nnn { filesize       } { convert } { 23 }
1310 \_unravel_tex_primitive_pdf:nnn { mdfivesum     } { convert } { 24 }
1311 \_unravel_tex_primitive_pdf:nnn { filedump       } { convert } { 25 }
1312 \_unravel_tex_primitive_pdf:nnn { match         } { convert } { 26 }
1313 \_unravel_tex_primitive_pdf:nnn { lastmatch     } { convert } { 27 }
1314 \_unravel_tex_primitive_pdf:nnn { uniformdeviate } { convert } { 28 }
1315 \_unravel_tex_primitive_pdf:nnn { normaldeviate } { convert } { 29 }
1316 \_unravel_tex_primitive_pdf:nnn { insertht     } { convert } { 30 }
1317 \_unravel_tex_primitive_pdf:nnn { ximagebbox    } { convert } { 31 }
1318 \_unravel_tex_primitive:nnn { jobname           } { convert } { 32 }
1319 \sys_if_engine_luatex:T
1320   { \_unravel_tex_primitive:nnn { directlua      } { convert } { 33 } }
1321   \_unravel_tex_primitive:nnn { expanded        } { convert } { 34 }
1322 \sys_if_engine_luatex:T
1323   { \_unravel_tex_primitive:nnn { luaescapestring } { convert } { 35 } }
1324 \sys_if_engine_xetex:T
1325   {
1326     \_unravel_tex_primitive:nnn { Ucharcat       } { convert } { 40 }
1327   }
1328 \_unravel_tex_primitive:nnn { the                 } { the } { 0 }
1329 \_unravel_tex_primitive:nnn { unexpanded         } { the } { 1 }

```

```

1330 \__unravel_tex_primitive:nnn { detokenize          } { the } { 5 }
1331 \__unravel_tex_primitive:nnn { topmark            } { top_bot_mark } { 0 }
1332 \__unravel_tex_primitive:nnn { firstmark         } { top_bot_mark } { 1 }
1333 \__unravel_tex_primitive:nnn { botmark           } { top_bot_mark } { 2 }
1334 \__unravel_tex_primitive:nnn { splitfirstmark    } { top_bot_mark } { 3 }
1335 \__unravel_tex_primitive:nnn { splitbotmark      } { top_bot_mark } { 4 }
1336 \__unravel_tex_primitive:nnn { topmarks          } { top_bot_mark } { 5 }
1337 \__unravel_tex_primitive:nnn { firstmarks        } { top_bot_mark } { 6 }
1338 \__unravel_tex_primitive:nnn { botmarks          } { top_bot_mark } { 7 }
1339 \__unravel_tex_primitive:nnn { splitfirstmarks    } { top_bot_mark } { 8 }
1340 \__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:
\__unravel_get_next_aux:w

```

If the input is empty, insert a frozen `\relax` (the alternative would be either to grab a token in the input stream after `\unravel`, which is tough, or simply produce an error and exit; perhaps this should be configurable). Then remove the first token in the input, and store it in `\l__unravel_head_gtl`. Set `\l__unravel_head_token` equal in meaning to that first token. Then set `\l__unravel_head_tl` to contain the token, unless it is a begin-group or end-group character, in which case this token list is emptied.

```

1341 \cs_new_protected:Npn \__unravel_get_next:
1342   {
1343     \__unravel_input_if_empty:TF
1344     {
1345       \__unravel_error:nnnnn { runaway-unravel } { } { } { } { }
1346       \__unravel_back_input_gtl:N \c__unravel_frozen_relax_gtl
1347     }
1348     { }
1349     \__unravel_input_gpop:N \l__unravel_head_gtl
1350     \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
1351     \gtl_if_tl:NNTF \l__unravel_head_gtl
1352     {
1353       \tl_set:Nx \l__unravel_head_tl
1354       { \gtl_head:N \l__unravel_head_gtl }
1355       \token_if_eq_meaning:NNT
1356       \l__unravel_head_token \__unravel_special_relax:
1357       \__unravel_get_next_notexpanded:
1358     }
1359     { \tl_clear:N \l__unravel_head_tl }
1360   }
1361 \cs_new_protected:Npn \__unravel_get_next_aux:w
1362   { \cs_set_eq:NN \l__unravel_head_token }

```


(End definition for `_unravel_get_next:` and `_unravel_get_next_aux:w.`)

`_unravel_get_next_notexpanded:` At this point we have likely encountered a special `\relax` marker that we use to mark cases where `\noexpand` acts on a control sequence or an active character. To make sure
`_unravel_notexpanded_test:w` of that check the control sequence has the form `\notexpanded:...`. Since we don't
`_unravel_notexpanded_expand:nN` know the escape character we must use `\cs_to_str:N`, but that function is not meant
`_unravel_notexpanded_expand:NN` for active characters and has a runaway argument if its argument is a space (active since we know its meaning is the special `\relax`). To avoid the runaway we include an arbitrary delimiter Z. If the token in `\l_unravel_head_tl` is not `\notexpanded:...` we do nothing. Otherwise `_unravel_notexpanded_expand:n` reconstructs the token that was hit with `\noexpand` (an active character if the argument is a single character) and do the job of `_unravel_get_next:`, setting `\l_unravel_head_token` to the special `\relax` marker for expandable commands, as `\noexpand` would.

```

1363 \cs_set_protected:Npn \_unravel_tmp:w #1
1364   {
1365     \cs_new_protected:Npn \_unravel_get_next_notexpanded:
1366       {
1367         \tl_if_eq:onTF { \l_unravel_head_tl } { \_unravel_unravel_marker: }
1368         { \_unravel_get_next_marker: }
1369         {
1370           \_unravel_exp_args:NNx \use:nn \_unravel_notexpanded_test:w
1371           { \scan_stop: \exp_after:wN \cs_to_str:N \l_unravel_head_tl Z }
1372           \q_mark \_unravel_notexpanded_expand:n
1373           #1 Z \q_mark \use_none:n
1374           \q_stop
1375         }
1376       }
1377     \cs_new_protected:Npn \_unravel_notexpanded_test:w
1378     ##1 #1 ##2 Z \q_mark ##3##4 \q_stop
1379     { ##3 {##2} }
1380   }
1381 \exp_args:Nx \_unravel_tmp:w { \scan_stop: \tl_to_str:n { notexpanded: } }
1382 \group_begin:
1383 \char_set_catcode_active:n { 0 }
1384 \cs_new_protected:Npn \_unravel_notexpanded_expand:n #1
1385   {
1386     \_unravel_exp_args:Nx \tl_if_empty:nTF { \str_tail:n {#1} }
1387     {
1388       \group_begin:
1389       \char_set_lccode:nm { 0 } { '#1 }
1390       \tex_lowercase:D
1391       {
1392         \group_end:
1393         \_unravel_notexpanded_expand:N ^^@
1394       }
1395     }
1396     {
1397       \group_begin: \exp_args:NNc \group_end:
1398       \_unravel_notexpanded_expand:N { \use_none:n #1 }
1399     }
1400   }
1401 \group_end:
1402 \cs_new_protected:Npn \_unravel_notexpanded_expand:N #1

```

```

1403 {
1404   \gtl_set:Nn \l__unravel_head_gtl {#1}
1405   \tl_set:Nn \l__unravel_head_tl {#1}
1406   \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
1407 }

```

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

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

```

1408 \cs_new_protected:Npn \__unravel_get_next_marker:
1409 {
1410   \__unravel_get_next:
1411   \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
1412     { \__unravel_error:nxxxx { nested-unravel } { } { } { } { } }
1413     { \__unravel_error:nxxxx { internal } { marker~unknown } { } { } { } }
1414   \__unravel_input_gpop_item:NF \l__unravel_argi_tl
1415     { \__unravel_error:nxxxx { internal } { marker~1 } { } { } { } }
1416   \__unravel_input_gpop_item:NF \l__unravel_argii_tl
1417     { \__unravel_error:nxxxx { internal } { marker~2 } { } { } { } }
1418   \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
1419   \__unravel_exp_args:Nx \__unravel_back_input:n
1420     { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
1421   \__unravel_get_next:
1422 }

```

(End definition for `__unravel_get_next_marker:.`)

`__unravel_get_token:` Call `__unravel_get_next:` to set `\l__unravel_head_gtl`, `\l__unravel_head_tl` and `\l__unravel_head_token`, then call `__unravel_set_cmd:` to set `\l__unravel_head_cmd_int` and `\l__unravel_head_char_int`.

```

1423 \cs_new_protected:Npn \__unravel_get_token:
1424 {
1425   \__unravel_get_next:
1426   \__unravel_set_cmd:
1427 }

```

(End definition for `__unravel_get_token:.`)

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

```

1428 \cs_new_protected:Npn \__unravel_set_cmd:
1429 {
1430   \__unravel_set_cmd_aux_meaning:
1431   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1432     { }
1433     {
1434       \__unravel_token_if_expandable:NTF \l__unravel_head_token
1435         {
1436           \token_if_macro:NTF \l__unravel_head_token

```

```

1437         { \_unravel_set_cmd_aux_macro: }
1438         { \_unravel_set_cmd_aux_unknown: }
1439     }
1440     {
1441     \token_if_cs:NTF \l__unravel_head_token
1442     { \_unravel_set_cmd_aux_cs: }
1443     { \_unravel_set_cmd_aux_char: }
1444     }
1445 }
1446 }

```

(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.

```

1447 \cs_new_protected:Npn \_unravel_set_cmd_aux_meaning:
1448 {
1449     \tl_set:Nx \l__unravel_head_meaning_tl
1450     {
1451         \exp_after:wN \_unravel_strip_escape:w
1452         \token_to_meaning:N \l__unravel_head_token
1453         \tl_to_str:n { : }
1454     }
1455     \tl_set:Nx \l__unravel_head_meaning_tl
1456     {
1457         \exp_after:wN \_unravel_set_cmd_aux_meaning:w
1458         \l__unravel_head_meaning_tl \q_stop
1459     }
1460 }
1461 \use:x
1462 {
1463     \cs_new:Npn \exp_not:N \_unravel_set_cmd_aux_meaning:w
1464     ##1 \token_to_str:N : ##2 \exp_not:N \q_stop {##1}
1465 }

```

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

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

```

1466 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nTF #1#2
1467 {
1468     \cs_if_exist:cTF { c__unravel_tex_#1_tl }
1469     {
1470         \exp_last_unbraced:Nv \_unravel_set_cmd_aux_primitive:nn
1471         { c__unravel_tex_#1_tl }
1472         #2
1473     }
1474 }
1475 \cs_generate_variant:Nn \_unravel_set_cmd_aux_primitive:nTF { o }
1476 \cs_new_protected:Npn \_unravel_set_cmd_aux_primitive:nn #1#2
1477 {
1478     \int_set:Nn \l__unravel_head_cmd_int {#1}

```

```

1479     \int_set:Nn \l__unravel_head_char_int {#2}
1480   }

```

(End definition for `__unravel_set_cmd_aux_primitive:nTF` and `__unravel_set_cmd_aux_primitive:nn`.)

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

```

1481 \cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
1482 {
1483   \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
1484   \int_zero:N \l__unravel_head_char_int
1485 }

```

(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`.

```

1486 \sys_if_engine_luatex:TF
1487 {
1488   \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1489   {
1490     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1491     \c__unravel_tex_relax_tl
1492     \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1493     { \tl_to_str:n { xpandable~luacall } }
1494     { }
1495     {
1496       \__unravel_error:nxxxx { unknown-primitive }
1497       { \l__unravel_head_meaning_tl } { } { } { }
1498     }
1499   }
1500 }
1501 {
1502   \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
1503   {
1504     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1505     \c__unravel_tex_relax_tl
1506     \__unravel_error:nxxxx { unknown-primitive }
1507     { \l__unravel_head_meaning_tl } { } { } { }
1508   }
1509 }

```

(End definition for `__unravel_set_cmd_aux_unknown:.`)

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

```

1510 \cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
1511 {
1512   \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
1513   { \tl_to_str:n { elect-font } }
1514   {
1515     \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
1516     \c__unravel_tex_nullfont_tl

```

```

1517     }
1518     { \_unravel_set_cmd_aux_numeric: }
1519 }

```

(End definition for _unravel_set_cmd_aux_cs:.)

_unravel_set_cmd_aux_numeric: Insert \q_mark before the first non-letter (in fact, anything less than A) in the \meaning by looping one character at a time (skipping spaces, but there should be none). We expect the first part to be char or mathchar (or kchar or omathchar in (u)pTeX), or one of count, dimen, skip, muskip, or toks. In the first two (three) cases, the command is char_given or math_given. It is otherwise identical to the corresponding primitive (\count etc.). We then keep track of the associated number (part after \q_mark) in \l__unravel_head_char_int. For unknown non-expandable primitives, assuming that their meaning consists solely of letters, the \q_mark is inserted at their end, and is followed by +0, so nothing breaks.

```

1520 \cs_new_protected:Npn \_unravel_set_cmd_aux_numeric:
1521 {
1522   \tl_set:Nx \l__unravel_tmpa_tl
1523     {
1524       \exp_after:wN \_unravel_set_cmd_aux_numeric:N
1525       \l__unravel_head_meaning_tl + 0
1526     }
1527   \exp_after:wN \_unravel_set_cmd_aux_numeric:w
1528   \l__unravel_tmpa_tl \q_stop
1529 }
1530 \cs_new:Npn \_unravel_set_cmd_aux_numeric:N #1
1531 {
1532   \if_int_compare:w '#1 < 'A \exp_stop_f:
1533     \exp_not:N \q_mark
1534     \exp_after:wN \use_i:nn
1535   \fi:
1536   #1 \_unravel_set_cmd_aux_numeric:N
1537 }
1538 \cs_new_protected:Npn \_unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
1539 {
1540   \str_case:nnF {#1}
1541     {
1542       { char }      { \_unravel_set_cmd_aux_given:n { char_given } }
1543       { kchar }    { \_unravel_set_cmd_aux_given:n { char_given } }
1544       { mathchar } { \_unravel_set_cmd_aux_given:n { math_given } }
1545       { omathchar } { \_unravel_set_cmd_aux_given:n { math_given } }
1546     }
1547     {
1548       \_unravel_set_cmd_aux_primitive:nTF {#1}
1549       { }
1550       { \_unravel_set_cmd_aux_unknown: }
1551       \int_add:Nn \l__unravel_head_char_int { 100 000 }
1552     }
1553   \int_add:Nn \l__unravel_head_char_int {#2}
1554 }
1555 \cs_new_protected:Npn \_unravel_set_cmd_aux_given:n #1
1556 {
1557   \int_set:Nn \l__unravel_head_cmd_int { \_unravel_tex_use:n {#1} }
1558   \int_zero:N \l__unravel_head_char_int

```

```
1559 }
(End definition for \_unravel_set_cmd_aux_numeric: and others.)
```

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

```
1560 \cs_new_protected:Npn \_unravel_set_cmd_aux_char:
1561 {
1562   \tl_set:Nx \l__unravel_head_meaning_tl
1563     { \token_to_meaning:N \l__unravel_head_token }
1564   \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
1565     { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
1566   \exp_after:wN \_unravel_set_cmd_aux_char:w
1567     \l__unravel_head_meaning_tl \q_stop
1568   \_unravel_exp_args:NNx \int_set:Nn \l__unravel_head_char_int
1569     { ‘ \_unravel_token_to_char:N \l__unravel_head_token }
1570 }
1571 \cs_new_protected:Npn \_unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
1572 {
1573   \int_set:Nn \l__unravel_head_cmd_int
1574     { \_unravel_tex_use:n { #1_char } }
1575 }
```

(End definition for `_unravel_set_cmd_aux_char:` and `_unravel_set_cmd_aux_char:w`.)

2.5 Manipulating the input

2.5.1 Elementary operations

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

```
1576 \cs_new:Npn \_unravel_input_to_str:
1577 {
1578   \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
1579     \_unravel_input_to_str_aux:n
1580 }
1581 \cs_new:Npn \_unravel_input_to_str_aux:n #1
1582 { \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.

```
1583 \cs_new_protected:Npn \_unravel_input_if_empty:TF
1584 {
1585   \int_compare:nNnTF \g__unravel_input_int = 0
1586     { \use_i:nn }
1587     {
1588       \gtl_if_empty:cTF
1589         { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1590         {
```

```

1591         \int_gdecr:N \g__unravel_input_int
1592         \__unravel_input_if_empty:TF
1593     }
1594     {
1595         \__unravel_input_split:
1596         \use_ii:nn
1597     }
1598 }
1599 }

```

(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

```

1600 \cs_new_protected:Npn \__unravel_input_split:
1601 {
1602     \int_compare:nNnT \g__unravel_input_int = 1
1603     {
1604         \exp_args:Nc \__unravel_input_split_aux:N
1605         { g__unravel_input_1_gtl }
1606     }
1607 }
1608 \cs_new_protected:Npn \__unravel_input_split_aux:N #1
1609 {
1610     \gtl_if_tl:NT #1
1611     {
1612         \gtl_if_head_is_N_type:NT #1
1613         {
1614             \tl_set:Nx \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
1615             \__unravel_exp_args:NNx \use:nn
1616             \__unravel_input_split_auxii:N
1617             { \tl_head:N \l__unravel_input_tmpa_tl }
1618         }
1619     }
1620 }
1621 \cs_new_protected:Npn \__unravel_input_split_auxii:N #1
1622 {
1623     \token_if_parameter:NF #1
1624     {
1625         \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
1626         { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
1627         \group_begin:
1628         \cs_set:Npn \__unravel_input_split_auxiii:w
1629             ##1 \__unravel_input_split_end: { + 1 }
1630         \int_gset:Nn \g__unravel_input_int
1631             { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
1632         \group_end:
1633         \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
1634         \l__unravel_input_tmpa_tl \__unravel_input_split_end:
1635     }
1636 }
1637 \cs_new:Npn \__unravel_input_split_end: { }
1638 \cs_new_protected:Npn \__unravel_input_split_auxiii:w
1639     #1 \__unravel_input_split_end:

```

```

1640 {
1641   \gtl_gclear_new:c
1642   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
1643   \gtl_gset:cn
1644   { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
1645   \int_gdecr:N \g__unravel_input_tmpa_int
1646 }

```

(End definition for __unravel_input_split:.)

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

```

1647 \cs_new_protected:Npn \__unravel_input_gset:n
1648 {
1649   \int_gzero:N \g__unravel_input_int
1650   \__unravel_back_input:n
1651 }

```

(End definition for __unravel_input_gset:n.)

__unravel_input_get:N

```

1652 \cs_new_protected:Npn \__unravel_input_get:N #1
1653 {
1654   \__unravel_input_if_empty:TF
1655   { \gtl_set:Nn #1 { \q_no_value } }
1656   {
1657     \gtl_get_left:cN
1658     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1659   }
1660 }

```

(End definition for __unravel_input_get:N.)

__unravel_input_get_left:N

```

\__unravel_input_get_left_aux:nN
\l__unravel_input_get_left_tl
1661 \tl_new:N \l__unravel_input_get_left_tl
1662 \cs_new_protected:Npn \__unravel_input_get_left:N #1
1663 {
1664   \tl_clear:N #1
1665   \exp_args:NV \__unravel_input_get_left_aux:nN \g__unravel_input_int #1
1666 }
1667 \cs_new_protected:Npn \__unravel_input_get_left_aux:nN #1#2
1668 {
1669   \int_compare:nNnF {#1} = 0
1670   {
1671     \tl_set:Nx \l__unravel_input_get_left_tl
1672     { \gtl_left_tl:c { g__unravel_input_#1_gtl } }
1673     \tl_concat:NNN #2 #2 \l__unravel_input_get_left_tl
1674     \gtl_if_tl:cT { g__unravel_input_#1_gtl }
1675     {
1676       \exp_args:Nf \__unravel_input_get_left_aux:nN
1677       { \int_eval:n { #1 - 1 } } #2
1678     }
1679   }
1680 }

```

(End definition for __unravel_input_get_left:N, __unravel_input_get_left_aux:nN, and \l__unravel_input_get_left_tl.)

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

```

1681 \cs_new_protected:Npn \__unravel_input_gpop:N #1
1682 {
1683   \__unravel_input_if_empty:TF
1684     { \gtl_set:Nn #1 { \q_no_value } }
1685     {
1686       \gtl_gpop_left:cN
1687       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1688     }
1689 }

```

(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.

```

1690 \cs_new_protected:Npn \__unravel_input_merge:
1691 {
1692   \int_gdecr:N \g__unravel_input_int
1693   \gtl_gconcat:ccc
1694     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1695     { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1696     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1697   \gtl_gclear:c
1698   { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
1699 }

```

(End definition for `__unravel_input_merge:`.)

`__unravel_input_gpop_item:NNTF` 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`

```

1700 \prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
1701 {
1702   \int_compare:nNnTF \g__unravel_input_int = 0
1703     { \prg_return_false: }
1704     {
1705       \exp_args:Nc \__unravel_input_gpop_item_aux:NN
1706       { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
1707     }
1708 }
1709 \cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
1710 {
1711   \gtl_gpop_left_item:NNTF #1#2
1712     { \prg_return_true: }
1713     {
1714       \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
1715         { \prg_return_false: }
1716         {
1717           \int_compare:nNnTF \g__unravel_input_int = 1
1718             { \prg_return_false: }
1719             {

```

```

1720         \_unravel_input_merge:
1721         \exp_args:Nc \_unravel_input_gpop_item_aux:NN
1722         {
1723             g\_unravel_input_
1724             \int_use:N \g\_unravel_input_int _gtl
1725         }
1726         #2
1727     }
1728 }
1729 }
1730 }

```

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

_unravel_input_gpop_tl:N

```

1731 \cs_new_protected:Npn \_unravel_input_gpop_tl:N #1
1732 { \tl_clear:N #1 \_unravel_input_gpop_tl_aux:N #1 }
1733 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:N #1
1734 {
1735     \int_compare:nNnF \g\_unravel_input_int = 0
1736     {
1737         \exp_args:Nc \_unravel_input_gpop_tl_aux:NN
1738         { g\_unravel_input_ \int_use:N \g\_unravel_input_int _gtl } #1
1739     }
1740 }
1741 \cs_new_protected:Npn \_unravel_input_gpop_tl_aux:NN #1#2
1742 {
1743     \gtl_if_tl:NTF #1
1744     {
1745         \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1746         \gtl_gclear:N #1
1747         \int_gdecr:N \g\_unravel_input_int
1748         \_unravel_input_gpop_tl_aux:N #2
1749     }
1750     {
1751         \int_compare:nNnTF \g\_unravel_input_int > 1
1752         { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
1753         { \use_i:nn }
1754         {
1755             \tl_put_right:Nx #2 { \gtl_left_tl:N #1 }
1756             \gtl_gpop_left_tl:N #1
1757         }
1758         {
1759             \_unravel_input_merge:
1760             \_unravel_input_gpop_tl_aux:N #2
1761         }
1762     }
1763 }

```

(End definition for _unravel_input_gpop_tl:N.)

_unravel_back_input:n
_unravel_back_input:x

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

```

1764 \cs_new_protected:Npn \__unravel_back_input:n
1765 {
1766   \int_gincr:N \g__unravel_input_int
1767   \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1768   \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1769 }
1770 \cs_generate_variant:Nn \__unravel_back_input:n { V , o }
1771 \cs_new_protected:Npn \__unravel_back_input:x
1772 { \__unravel_exp_args:Nx \__unravel_back_input:n }

```

(End definition for __unravel_back_input:n.)

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

```

1773 \cs_new_protected:Npn \__unravel_back_input_gtl:N #1
1774 {
1775   \gtl_if_tl:NTF #1
1776   { \__unravel_back_input:x { \gtl_left_tl:N #1 } }
1777   {
1778     \gtl_gconcat:cNc
1779     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1780     #1
1781     { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
1782   }
1783 }

```

(End definition for __unravel_back_input_gtl:N.)

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

```

1784 \cs_new_protected:Npn \__unravel_back_input:
1785 { \__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.

```

1786 \cs_new_protected:Npn \__unravel_back_input_tl_o:
1787 {
1788   \tl_set:Nx \l__unravel_tmpa_tl
1789   { \exp_args:NV \exp_not:o \l__unravel_head_tl }
1790   \__unravel_back_input:V \l__unravel_tmpa_tl
1791   \__unravel_print_expansion:x
1792   { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
1793 }

```

(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.

```

1794 \cs_new_protected:Npn \__unravel_insert_relax:
1795 {
1796   \__unravel_back_input:

```

```

1797 \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
1798 \__unravel_back_input:
1799 \__unravel_print_action:
1800 }

```

(End definition for __unravel_insert_relax:.)

__unravel_insert_group_begin_error:

```

1801 \cs_new_protected:Npn \__unravel_insert_group_begin_error:
1802 {
1803   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
1804   \__unravel_back_input:
1805   \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
1806   \__unravel_back_input:
1807   \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
1808   \__unravel_print_action:
1809 }

```

(End definition for __unravel_insert_group_begin_error:.)

__unravel_insert_dollar_error:

```

1810 \cs_new_protected:Npn \__unravel_insert_dollar_error:
1811 {
1812   \__unravel_back_input:
1813   \__unravel_back_input:n { $ } % $
1814   \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
1815   \__unravel_print_action:
1816 }

```

(End definition for __unravel_insert_dollar_error:.)

2.5.3 Macro calls

__unravel_macro_prefix:N

__unravel_macro_parameter:N

__unravel_macro_replacement:N

```

1817 \use:x
1818 {
1819   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
1820   {
1821     \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
1822     \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
1823     \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
1824     \exp_not:N \q_stop
1825   }
1826   \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
1827     \exp_not:n { #1 } \tl_to_str:n { : } \exp_not:n { #2 -> }
1828     \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
1829     { \exp_not:n { #4 #6 { #1 } { #2 } { #3 } } }
1830 }
1831 \cs_new:Npn \__unravel_macro_prefix:N #1
1832 { \__unravel_macro_split_do:NN #1 \use_i:nnn }
1833 \cs_new:Npn \__unravel_macro_parameter:N #1
1834 { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
1835 \cs_new:Npn \__unravel_macro_replacement:N #1
1836 { \__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`.)

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

```

1837 \cs_new_protected:Npn \__unravel_macro_call:
1838 {
1839   \bool_if:NTF \g__unravel_speedup_macros_bool
1840   {
1841     \tl_set:Nx \l__unravel_tmpa_tl
1842     {^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
1843     \__unravel_tl_if_in:ooTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
1844     { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
1845   }
1846   { \__unravel_macro_call_safe: }
1847   \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1848   \__unravel_print_expansion:
1849 }
1850 \cs_new_protected:Npn \__unravel_macro_call_safe:
1851 {
1852   \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1853   \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1854 }
1855 \cs_new_protected:Npn \__unravel_macro_call_quick:
1856 {
1857   \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
1858   { ? \use_none_delimit_by_q_stop:w } \q_stop
1859 }
1860 \cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
1861 {
1862   \use_none:n #2
1863   \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
1864   { \__unravel_macro_call_quick_runaway:Nw #3 }
1865   \tl_put_right:Nx \l__unravel_head_tl
1866   { { \exp_not:V \l__unravel_tmpa_tl } }
1867   \__unravel_macro_call_quick_loop:NNN
1868   #3
1869 }
1870 \cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
1871 {
1872   \__unravel_error:nxxxx { runaway-macro-parameter }
1873   { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
1874 }

```

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

2.6 Expand next token

`__unravel_expand_do:N` The argument is a command that will almost always be run to continue a loop whose aim is to find the next non-expandable token, for various purposes. The only case where we will end up grabbing the argument is to suppress the loop by `__unravel_noexpand:N`.

- `__unravel_get_x_next`: when T_EX is looking for the first non-expandable token in the main loop or when looking for numbers, optional spaces etc.
- `__unravel_get_x_or_protected`: at the start of an alignment cell.
- `__unravel_get_token_xdef`: in the replacement text of `\edef` and `\xdef`.
- `__unravel_get_token_x`: in the argument of `\message` and the like.
- `\prg_do_nothing`: in `__unravel_expandafter`: namely after `\expandafter`.

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

```

1875 \cs_new_protected:Npn \__unravel_expand_do:N
1876 {
1877   \__unravel_set_action_text:
1878   \bool_if:NT \g__unravel_internal_debug_bool
1879     {
1880       \__unravel_set_cmd:
1881       \__unravel_exp_args:Nx \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_in
1882     }
1883   \token_if_macro:NTF \l__unravel_head_token
1884     { \__unravel_macro_call: }
1885     { \__unravel_expand_nonmacro: }
1886 }

```

(End definition for `__unravel_expand_do:N`.)

`__unravel_expand_nonmacro:` The token is a primitive. We find its (cleaned-up) `\meaning`, and call the function implementing that expansion. If we do not recognize the meaning then it is probably an unknown primitive. Then do something similar to what we do for macros: get all tokens that are not too unlikely to appear in the arguments of the primitive and expand the resulting token list once before putting it back into the input stream.

```

1887 \cs_new_protected:Npn \__unravel_expand_nonmacro:
1888 {
1889   \__unravel_set_cmd_aux_meaning:
1890   \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
1891   {
1892     \cs_if_exist_use:cF
1893     { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
1894     { \__unravel_error:nxxxx { internal } { expandable } { } { } { } }
1895   }
1896   {
1897     \__unravel_set_cmd_aux_unknown:
1898     \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
1899     \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
1900     \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
1901     \__unravel_print_expansion:
1902   }
1903 }

```

(End definition for `__unravel_expand_nonmacro:.`)

`__unravel_get_x_next:` Get a token. If it is expandable, then expand it, and repeat. This function does not set the `cmd` and `char` integers. It is the basis of all routines that look for keywords, numbers, equal signs, filenames, optional spaces etc (in the language of L^AT_EX3 these are situations where T_EX “f-expands”). It is also the basis of the `__unravel_main_loop:`.

```

1904 \cs_new_protected:Npn \__unravel_get_x_next:
1905 {
1906   \__unravel_get_next:
1907   \__unravel_token_if_expandable:NT \l__unravel_head_token
1908   { \__unravel_expand_do:N \__unravel_get_x_next: }
1909 }

```

(End definition for __unravel_get_x_next:.)

`_unravel_get_x_or_protected:` Get a token. If it is expandable, but not protected, then expand it, and repeat. This function does not set the `cmd` and `char` integers. This function is not used at present: it will be used at the start of alignment cells.

```

1910 \cs_new_protected:Npn \_unravel_get_x_or_protected:
1911 {
1912   \__unravel_get_next:
1913   \__unravel_token_if_protected:NF \l__unravel_head_token
1914   { \__unravel_expand_do:N \_unravel_get_x_or_protected: }
1915 }

```

(End definition for _unravel_get_x_or_protected:.)

`__unravel_get_token_xdef:` These are similar to `__unravel_get_x_next:`, for use when reading the replacement text of `\edef`/`\xdef` or the argument of a primitive like `\message` that should be expanded as we read tokens. Loop until finding a non-expandable token (or protected macro).

```

1916 \cs_new_protected:Npn \__unravel_get_token_xdef:
1917 {
1918   \__unravel_get_next:
1919   \__unravel_token_if_protected:NF \l__unravel_head_token
1920   { \__unravel_expand_do:N \__unravel_get_token_xdef: }
1921 }
1922 \cs_new_protected:Npn \__unravel_get_token_x:
1923 {
1924   \__unravel_get_next:
1925   \__unravel_token_if_protected:NF \l__unravel_head_token
1926   { \__unravel_expand_do:N \__unravel_get_token_x: }
1927 }

```

(End definition for __unravel_get_token_xdef: and __unravel_get_token_x:.)

2.7 Basic scanning subroutines

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

```

1928 \cs_new_protected:Npn \__unravel_get_x_non_blank:
1929 {
1930   \__unravel_get_x_next:
1931   \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1932   { \__unravel_get_x_non_blank: }
1933 }

```

(End definition for `_unravel_get_x_non_blank:`.)

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

```
1934 \cs_new_protected:Npn \_unravel_get_x_non_relax:
1935 {
1936   \_unravel_get_x_next:
1937   \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
1938     { \_unravel_get_x_non_relax: }
1939     {
1940       \token_if_eq_meaning:NNTF \l__unravel_head_token \_unravel_special_relax:
1941         { \_unravel_get_x_non_relax: }
1942         {
1943           \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
1944             { \_unravel_get_x_non_relax: }
1945         }
1946     }
1947 }
```

(End definition for `_unravel_get_x_non_relax:`.)

`_unravel_skip_optional_space:`

```
1948 \cs_new_protected:Npn \_unravel_skip_optional_space:
1949 {
1950   \_unravel_get_x_next:
1951   \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1952     { \_unravel_back_input: }
1953 }
```

(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_rescan_something_internal:n` leaves raw numbers in the previous-input sequence.

```
1954 \cs_new_protected:Npn \_unravel_scan_optional_equals:
1955 {
1956   \_unravel_get_x_non_blank:
1957   \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
1958     { \_unravel_prev_input:n { = } }
1959     {
1960       \_unravel_prev_input_silent:n { = }
1961       \_unravel_back_input:
1962     }
1963 }
```

(End definition for `_unravel_scan_optional_equals:`.)

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

```
1964 \cs_new_protected:Npn \_unravel_scan_left_brace:
1965 {
1966   \_unravel_get_x_non_relax:
1967   \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
1968     {
```



```

1969     \__unravel_insert_group_begin_error:
1970     \__unravel_get_next:
1971     }
1972 }

```

(End definition for __unravel_scan_left_brace:.)

```

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

```

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

```

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

```

```

1973 \cs_new_protected:Npn \__unravel_scan_keyword:n #1
1974 { \__unravel_scan_keyword:nTF {#1} { } { } }
1975 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
1976 { T , F , TF }
1977 {
1978   \__unravel_prev_input_gpush_gtl:
1979   \__unravel_scan_keyword_loop:NNN \c_true_bool
1980   #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
1981 }
1982 \cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
1983 {
1984   \quark_if_recursion_tail_stop_do:nn {#2}
1985   { \__unravel_scan_keyword_true: }
1986   \quark_if_recursion_tail_stop_do:nn {#3}
1987   { \__unravel_error:nxxxx { internal } { odd-keyword-length } { } { } { } }
1988   \__unravel_get_x_next:
1989   \__unravel_scan_keyword_test:NNTF #2#3
1990   {
1991     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
1992     \__unravel_scan_keyword_loop:NNN \c_false_bool
1993   }

```

```

1994     {
1995     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
1996     { \__unravel_scan_keyword_false:w }
1997     \bool_if:NF #1
1998     { \__unravel_scan_keyword_false:w }
1999     \__unravel_scan_keyword_loop:NNN #1#2#3
2000     }
2001   }
2002 \prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
2003 { TF }
2004 {
2005   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2006   { \prg_return_false: }
2007   {
2008     \str_if_eq:eeTF
2009     { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
2010     { \prg_return_true: }
2011     {
2012       \str_if_eq:eeTF
2013       { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
2014       { \prg_return_true: }
2015       { \prg_return_false: }
2016     }
2017   }
2018 }
2019 \cs_new_protected:Npn \__unravel_scan_keyword_true:
2020 {
2021   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2022   \__unravel_prev_input:x { \gtl_to_str:N \l__unravel_tmpb_gtl }
2023   \prg_return_true:
2024 }
2025 \cs_new_protected:Npn \__unravel_scan_keyword_false:w
2026 #1 \q_recursion_stop
2027 {
2028   \__unravel_back_input:
2029   \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2030   \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
2031   \prg_return_false:
2032 }

```

(End definition for __unravel_scan_keyword:n and others.)

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

```

2033 \cs_new_protected:Npn \__unravel_scan_to:
2034 {
2035   \__unravel_scan_keyword:nF { tTo0 }
2036   {
2037     \__unravel_error:nnnnn { missing-to } { } { } { } { }
2038     \__unravel_prev_input:n { to }
2039   }
2040 }

```

(End definition for __unravel_scan_to:.)

`__unravel_scan_font_ident:` Find a font identifier.

```
2041 \cs_new_protected:Npn \__unravel_scan_font_ident:
2042 {
2043   \__unravel_get_x_non_blank:
2044   \__unravel_set_cmd:
2045   \int_case:nnF \l__unravel_head_cmd_int
2046   {
2047     { \__unravel_tex_use:n { def_font } }
2048     { \__unravel_prev_input:V \l__unravel_head_tl }
2049     { \__unravel_tex_use:n { letterspace_font } }
2050     { \__unravel_prev_input:V \l__unravel_head_tl }
2051     { \__unravel_tex_use:n { pdf_copy_font } }
2052     { \__unravel_prev_input:V \l__unravel_head_tl }
2053     { \__unravel_tex_use:n { set_font } }
2054     { \__unravel_prev_input:V \l__unravel_head_tl }
2055     { \__unravel_tex_use:n { def_family } }
2056     {
2057       \__unravel_prev_input:V \l__unravel_head_tl
2058       \__unravel_scan_int:
2059     }
2060   }
2061   {
2062     \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
2063     \__unravel_back_input:
2064     \__unravel_prev_input:n { \__unravel_nullfont: }
2065   }
2066 }
```

(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`).

```
2067 \cs_new_protected:Npn \__unravel_scan_font_int:
2068 {
2069   \int_case:nnF \l__unravel_head_char_int
2070   {
2071     { 0 } { \__unravel_scan_font_ident: }
2072     { 1 } { \__unravel_scan_font_ident: }
2073     { 6 } { \__unravel_scan_font_ident: }
2074   }
2075   { \__unravel_scan_font_ident: \__unravel_scan_int: }
2076 }
```

(End definition for __unravel_scan_font_int:.)

`__unravel_scan_font_dimen:` Find operands for `\fontdimen`.

```
2077 \cs_new_protected:Npn \__unravel_scan_font_dimen:
2078 {
2079   \__unravel_scan_int:
2080   \__unravel_scan_font_ident:
2081 }
```

(End definition for __unravel_scan_font_dimen:.)

`__unravel_rescan_something_internal:n` Receives an (explicit) “level” argument:

`__unravel_scan_something_aux:nwn`

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

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

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

If the argument (`#3` in the auxiliary) is < 4 but the level that will be produced (`#1` in the auxiliary) is ≥ 4 (that is, 4, 5, or 8) complain about a missing number and insert a zero dimension, to get exactly T_EX's error recovery. If the level produced is 8, complain that `\the` cannot do this.

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

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

```

2082 \cs_new_protected:Npn \__unravel_rescan_something_internal:n #1
2083 {
2084   \__unravel_set_cmd:
2085   \__unravel_set_action_text:
2086   \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
2087   \exp_after:wN \__unravel_scan_something_aux:nwn
2088   \l__unravel_tmpa_tl \q_stop {#1}
2089 }
2090 \cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
2091 {
2092   \int_compare:nT { #3 < 4 <= #1 }
2093   {
2094     \__unravel_back_input:
2095     \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2096     \__unravel_thing_use_get:nNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
2097     \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
2098     \__unravel_break:w
2099   }
2100   \int_compare:nNnT {#1} = { 8 }

```

```

2101     {
2102     \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
2103     \__unravel_rescan_something_internal_auxii:nn 0 { 0 }
2104     \__unravel_break:w
2105     }
2106 \tl_if_empty:nF {#2}
2107     {
2108     \__unravel_prev_input_gpush:N \l__unravel_head_tl
2109     \__unravel_print_action:
2110     #2
2111     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2112     }
2113 \int_compare:nNnTF {#3} < { 4 }
2114     { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
2115     { \tl_set:Nx \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
2116 \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
2117 \__unravel_break_point:
2118 \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
2119 }
2120 \cs_new_protected:Npn \__unravel_rescan_something_internal_auxii:nn #1#2
2121 {
2122 \__unravel_prev_input_silent:n {#1}
2123 \__unravel_set_action_text:
2124 \__unravel_set_action_text:x
2125     { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
2126 \int_gset:Nn \g__unravel_val_level_int {#2}
2127 }
2128 \cs_generate_variant:Nn \__unravel_rescan_something_internal_auxii:nn { V }

```

(End definition for __unravel_rescan_something_internal:n and __unravel_scan_something_aux:nwn.)

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

```

2129 \cs_new:Npn \__unravel_thing_case:
2130 {
2131 \int_case:nnF \l__unravel_head_cmd_int
2132 {
2133     { 68 } { 0 } % char_given
2134     { 69 } { 0 } % math_given
2135     { 70 } { \__unravel_thing_last_item: } % last_item
2136     { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
2137     { 72 } { 5 } % assign_toks
2138     { 73 } { 0 } % assign_int
2139     { 74 } { 1 } % assign_dimen
2140     { 75 } { 2 } % assign_glue
2141     { 76 } { 3 } % assign_mu_glue
2142     { 77 } { 1 \__unravel_scan_font_dimen: } % assign_font_dimen
2143     { 78 } { 0 \__unravel_scan_font_int: } % assign_font_int
2144     { 79 } { \__unravel_thing_set_aux: } % set_aux
2145     { 80 } { 0 } % set_prev_graf

```

```

2146     { 81 } { 1 } % set_page_dimen
2147     { 82 } { 0 } % set_page_int
2148     { 83 } { 1 \_unravel_scan_int: } % set_box_dimen
2149     { 84 } { 0 \_unravel_scan_int: } % set_shape
2150     { 85 } { 0 \_unravel_scan_int: } % def_code
2151     { 86 } { 4 \_unravel_scan_int: } % def_family
2152     { 87 } { 4 } % set_font
2153     { 88 } { 4 } % def_font
2154     { 89 } { \_unravel_thing_register: } % register
2155     {101 } { 4 } % letterspace_font
2156     {102 } { 4 } % pdf_copy_font
2157   }
2158   { 8 }
2159 }
2160 \cs_new:Npn \_unravel_thing_set_aux:
2161 { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
2162 \cs_new:Npn \_unravel_thing_last_item:
2163 {
2164   \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
2165   {
2166     \int_case:nnF \l__unravel_head_char_int
2167     {
2168       { 1 } { 1 } % lastkern
2169       { 2 } { 2 } % lastskip
2170     }
2171     { 0 } % other integer parameters
2172   }
2173   {
2174     \int_case:nnF \l__unravel_head_char_int
2175     {
2176       { 26 } { 0 \_unravel_scan_normal_glue: } % gluestretchorder
2177       { 27 } { 0 \_unravel_scan_normal_glue: } % glueshrinkorder
2178       { 28 } % fontcharwd
2179       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2180       { 29 } % fontcharht
2181       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2182       { 30 } % fontchardp
2183       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2184       { 31 } % fontcharic
2185       { 1 \_unravel_scan_font_ident: \_unravel_scan_int: }
2186       { 32 } { 1 \_unravel_scan_int: } % parshapelength
2187       { 33 } { 1 \_unravel_scan_int: } % parshapeindent
2188       { 34 } { 1 \_unravel_scan_int: } % parshapedimen
2189       { 35 } { 1 \_unravel_scan_normal_glue: } % gluestretch
2190       { 36 } { 1 \_unravel_scan_normal_glue: } % glueshrink
2191       { 37 } { 2 \_unravel_scan_mu_glue: } % mutoglu
2192       { 38 } { 3 \_unravel_scan_normal_glue: } % gluetomu
2193       { 39 } % numepr
2194       { 0 \_unravel_scan_expr:N \_unravel_scan_int: }
2195       { 40 } % dimexpr
2196       { 1 \_unravel_scan_expr:N \_unravel_scan_normal_dimen: }
2197       { 41 } % glueexpr
2198       { 2 \_unravel_scan_expr:N \_unravel_scan_normal_glue: }
2199       { 42 } % muexpr

```

```

2200         { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
2201     }
2202     { }
2203 }
2204 }
2205 \cs_new:Npn \__unravel_thing_register:
2206 {
2207     \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
2208     \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
2209     { \__unravel_scan_int: }
2210 }

```

(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.

```

2211 \cs_new_protected:Npn \__unravel_scan_toks_register:
2212 {
2213     \int_compare:nNnT \l__unravel_head_char_int = 0
2214     { \__unravel_scan_int: }
2215 }

```

(End definition for __unravel_scan_toks_register:.)

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

```

2216 \cs_new_protected:Npn \__unravel_thing_use_get:mnNN #1#2#3#4
2217 {
2218     \int_compare:nNnTF {#2} < { 3 }
2219     {
2220         \int_compare:nNnT {#1} = { 3 }
2221         { \__unravel_tex_error:nV { incompatible-units } #3 }
2222         \tl_set:Nx #4
2223         {
2224             \int_case:nn { \int_min:nn {#1} {#2} }
2225             {
2226                 { 0 } \int_eval:n
2227                 { 1 } \dim_eval:n
2228                 { 2 } \skip_eval:n
2229             }
2230             { \int_compare:nNnT {#1} = { 3 } \tex_mutogluue:D #3 }
2231         }
2232     }
2233     {
2234         \int_case:nnF {#1}
2235         {
2236             { 0 } { \tl_set:Nx #4 { \int_eval:n {#3} } }
2237             { 3 } { \tl_set:Nx #4 { \muskip_eval:n {#3} } }
2238         }
2239         {
2240             \__unravel_tex_error:nV { incompatible-units } #3
2241             \tl_set:Nx #4 { \muskip_eval:n { \tex_gluetomu:D #3 } }
2242         }
2243     }
2244 }

```

(End definition for _unravel_thing_use_get:nmnn.)

```
\_unravel_scan_expr:N
\_unravel_scan_expr_aux:NN
\_unravel_scan_factor:N
2245 \cs_new_protected:Npn \_unravel_scan_expr:N #1
2246 { \_unravel_scan_expr_aux:NN #1 \c_false_bool }
2247 \cs_new_protected:Npn \_unravel_scan_expr_aux:NN #1#2
2248 {
2249   \_unravel_get_x_non_blank:
2250   \_unravel_scan_factor:N #1
2251   \_unravel_scan_expr_op:NN #1#2
2252 }
2253 \cs_new_protected:Npn \_unravel_scan_expr_op:NN #1#2
2254 {
2255   \_unravel_get_x_non_blank:
2256   \tl_case:NnF \l__unravel_head_tl
2257   {
2258     \c__unravel_plus_tl
2259     {
2260       \_unravel_prev_input:V \l__unravel_head_tl
2261       \_unravel_scan_expr_aux:NN #1#2
2262     }
2263     \c__unravel_minus_tl
2264     {
2265       \_unravel_prev_input:V \l__unravel_head_tl
2266       \_unravel_scan_expr_aux:NN #1#2
2267     }
2268     \c__unravel_times_tl
2269     {
2270       \_unravel_prev_input:V \l__unravel_head_tl
2271       \_unravel_get_x_non_blank:
2272       \_unravel_scan_factor:N \_unravel_scan_int:
2273       \_unravel_scan_expr_op:NN #1#2
2274     }
2275     \c__unravel_over_tl
2276     {
2277       \_unravel_prev_input:V \l__unravel_head_tl
2278       \_unravel_get_x_non_blank:
2279       \_unravel_scan_factor:N \_unravel_scan_int:
2280       \_unravel_scan_expr_op:NN #1#2
2281     }
2282     \c__unravel_rp_tl
2283     {
2284       \bool_if:NTF #2
2285       { \_unravel_prev_input:V \l__unravel_head_tl }
2286       { \_unravel_back_input: }
2287     }
2288   }
2289   {
2290     \bool_if:NTF #2
2291     {
2292       \_unravel_error:nmnnn { missing-rparen } { } { } { } { }
2293       \_unravel_back_input:
2294       \_unravel_prev_input:V \c__unravel_rp_tl
2295     }
  }
```



```

2296         {
2297             \token_if_eq_meaning:NNF \l__unravel_head_token \scan_stop:
2298             { \__unravel_back_input: }
2299         }
2300     }
2301 }
2302 \cs_new_protected:Npn \__unravel_scan_factor:N #1
2303 {
2304     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
2305     {
2306         \__unravel_prev_input:V \l__unravel_head_tl
2307         \__unravel_scan_expr_aux:NN #1 \c_true_bool
2308     }
2309     {
2310         \__unravel_back_input:
2311         #1
2312     }
2313 }

```

(End definition for __unravel_scan_expr:N, __unravel_scan_expr_aux:NN, and __unravel_scan_factor:N.)

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

```

2314 \cs_new_protected:Npn \__unravel_scan_signs:
2315 {
2316     \__unravel_get_x_non_blank:
2317     \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
2318     {
2319         \__unravel_prev_input:V \l__unravel_head_tl
2320         \__unravel_scan_signs:
2321     }
2322     {
2323         \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
2324         {
2325             \__unravel_prev_input:V \l__unravel_head_tl
2326             \__unravel_scan_signs:
2327         }
2328     }
2329 }

```

(End definition for __unravel_scan_signs:.)

```

\__unravel_scan_int:
\__unravel_scan_int_char: 2330 \cs_new_protected:Npn \__unravel_scan_int:
\__unravel_scan_int_lq: 2331 {
\__unravel_scan_int_explicit:n 2332     \__unravel_scan_signs:
2333     \__unravel_set_cmd:
2334     \__unravel_cmd_if_internal:TF
2335     { \__unravel_rescan_something_internal:n { 0 } }
2336     { \__unravel_scan_int_char: }
2337 }
2338 \cs_new_protected:Npn \__unravel_scan_int_char:
2339 {
2340     \tl_case:NnF \l__unravel_head_tl

```

```

2341 {
2342   \c__unravel_lq_tl { \__unravel_scan_int_lq: }
2343   \c__unravel_rq_tl
2344   {
2345     \__unravel_prev_input:V \l__unravel_head_tl
2346     \__unravel_get_x_next:
2347     \__unravel_scan_int_explicit:Nn \c_false_bool { ' }
2348   }
2349   \c__unravel_dq_tl
2350   {
2351     \__unravel_prev_input:V \l__unravel_head_tl
2352     \__unravel_get_x_next:
2353     \__unravel_scan_int_explicit:Nn \c_false_bool { " }
2354   }
2355 }
2356 { \__unravel_scan_int_explicit:Nn \c_false_bool { } }
2357 }
2358 \cs_new_protected:Npn \__unravel_scan_int_lq:
2359 {
2360   \__unravel_get_next:
2361   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2362   {
2363     \tl_set:Nx \l__unravel_head_tl
2364     { \__unravel_token_to_char:N \l__unravel_head_token }
2365   }
2366   \tl_set:Nx \l__unravel_tmpa_tl
2367   { \int_eval:n { \exp_after:wN ' \l__unravel_head_tl } }
2368   \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2369   \__unravel_print_action:x
2370   { ' \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
2371   \__unravel_skip_optional_space:
2372 }
2373 \cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
2374 {
2375   \if_int_compare:w 1
2376   < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
2377   \exp_after:wN \use_i:nn
2378   \else:
2379   \exp_after:wN \use_ii:nn
2380   \fi:
2381   {
2382     \__unravel_prev_input:V \l__unravel_head_tl
2383     \__unravel_get_x_next:
2384     \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
2385   }
2386   {
2387     \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
2388     { \__unravel_back_input: }
2389     \bool_if:NF #1
2390     {
2391       \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
2392       \__unravel_prev_input:n { 0 }
2393     }
2394   }

```

```
2395 }
```

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

```
\_unravel_scan_normal_dimen:
```

```
2396 \cs_new_protected:Npn \_unravel_scan_normal_dimen:
2397 { \_unravel_scan_dimen:nN { 2 } \c_false_bool }
```

(End definition for `_unravel_scan_normal_dimen`.)

```
\_unravel_scan_dimen:nN
```

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

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

```
2398 \cs_new_protected:Npn \_unravel_scan_dimen:nN #1#2
2399 {
2400   \_unravel_scan_signs:
2401   \_unravel_prev_input_gpush:
2402   \_unravel_set_cmd:
2403   \_unravel_cmd_if_internal:TF
2404   {
2405     \int_compare:nNnTF {#1} = { 3 }
2406       { \_unravel_rescan_something_internal:n { 3 } }
2407       { \_unravel_rescan_something_internal:n { 1 } }
2408     \int_compare:nNnT \g_unravel_val_level_int = { 0 }
2409       { \_unravel_scan_dim_unit:nN {#1} #2 }
2410   }
2411   { \_unravel_scan_dimen_char:nN {#1} #2 }
2412   \_unravel_prev_input_gpop:N \l_unravel_head_tl
2413   \_unravel_prev_input_silent:V \l_unravel_head_tl
2414 }
2415 \cs_new_protected:Npn \_unravel_scan_dimen_char:nN #1#2
2416 {
2417   \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_comma_tl
2418     { \tl_set_eq:NN \l_unravel_head_tl \c_unravel_point_tl }
2419   \tl_if_eq:NNTF \l_unravel_head_tl \c_unravel_point_tl
2420     {
2421       \_unravel_prev_input:n { . }
2422       \_unravel_scan_decimal_loop:
2423     }
2424     {
2425       \_unravel_tl_if_in:ooTF { 0123456789 } \l_unravel_head_tl
2426       {
2427         \_unravel_back_input:
2428         \_unravel_scan_int:
2429         \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_comma_tl
2430           { \tl_set_eq:NN \l_unravel_head_tl \c_unravel_point_tl }
2431         \tl_if_eq:NNT \l_unravel_head_tl \c_unravel_point_tl
2432           {
2433             \_unravel_input_gpop:N \l_unravel_tmpb_gtl
```

```

2434         \_unravel_prev_input:n { . }
2435         \_unravel_scan_decimal_loop:
2436     }
2437 }
2438 {
2439     \_unravel_back_input:
2440     \_unravel_scan_int:
2441 }
2442 }
2443 \_unravel_scan_dim_unit:nN {#1} #2
2444 }
2445 \cs_new_protected:Npn \_unravel_scan_dim_unit:nN #1#2
2446 {
2447     \bool_if:NT #2
2448     {
2449         \_unravel_scan_keyword:nT { fFiIlL }
2450         {
2451             \_unravel_scan_inf_unit_loop:
2452             \_unravel_break:w
2453         }
2454     }
2455     \_unravel_get_x_non_blank:
2456     \_unravel_set_cmd:
2457     \_unravel_cmd_if_internal:TF
2458     {
2459         \_unravel_prev_input_gpush:
2460         \_unravel_rescan_something_internal:n {#1}
2461         \int_compare:nNnTF \g_unravel_val_level_int = { 0 }
2462             { \_unravel_prev_input_join_get:nnN {#1} { sp } \l_unravel_tmpa_tl }
2463             { \_unravel_prev_input_join_get:nnN {#1} { } \l_unravel_tmpa_tl }
2464         \_unravel_prev_input_gpush:N \l_unravel_tmpa_tl
2465         \exp_after:wN \use_none:n \_unravel_break:w
2466     }
2467     { }
2468     \_unravel_back_input:
2469     \int_compare:nNnT {#1} = { 3 }
2470     {
2471         \_unravel_scan_keyword:nT { mMuU } { \_unravel_break:w }
2472         \_unravel_tex_error:nV { missing-mu } \l_unravel_head_tl
2473         \_unravel_prev_input:n { mu }
2474         \_unravel_break:w
2475     }
2476     \_unravel_scan_keyword:nT { eEmM } { \_unravel_break:w }
2477     \_unravel_scan_keyword:nT { eExX } { \_unravel_break:w }
2478     \_unravel_scan_keyword:nT { pPxX } { \_unravel_break:w }
2479     \_unravel_scan_keyword:nT { tTrRuUeE }
2480     { \_unravel_prepare_mag: }
2481     \_unravel_scan_keyword:nT { pPtT } { \_unravel_break:w }
2482     \_unravel_scan_keyword:nT { iInN } { \_unravel_break:w }
2483     \_unravel_scan_keyword:nT { pPcC } { \_unravel_break:w }
2484     \_unravel_scan_keyword:nT { cCmM } { \_unravel_break:w }
2485     \_unravel_scan_keyword:nT { mMmM } { \_unravel_break:w }
2486     \_unravel_scan_keyword:nT { bBpP } { \_unravel_break:w }
2487     \_unravel_scan_keyword:nT { dDdD } { \_unravel_break:w }

```

```

2488     \__unravel_scan_keyword:nT { cCc } { \__unravel_break:w }
2489     \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
2490     \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
2491     \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
2492     \__unravel_tex_error:nV { missing-pt } \l__unravel_head_tl
2493     \__unravel_prev_input:n { pt }
2494     \__unravel_break_point:
2495     \__unravel_skip_optional_space:
2496   }
2497   \cs_new_protected:Npn \__unravel_scan_inf_unit_loop:
2498     { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
2499   \cs_new_protected:Npn \__unravel_scan_decimal_loop:
2500     {
2501       \__unravel_get_x_next:
2502       \tl_if_empty:NTF \l__unravel_head_tl
2503         { \use_ii:nn }
2504         { \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl }
2505         {
2506           \__unravel_prev_input:V \l__unravel_head_tl
2507           \__unravel_scan_decimal_loop:
2508         }
2509       {
2510         \token_if_eq_catcode:NMF \l__unravel_head_token \c_space_token
2511         { \__unravel_back_input: }
2512         \__unravel_prev_input_silent:n { ~ }
2513       }
2514     }

```

(End definition for __unravel_scan_dimen:nN.)

```

\__unravel_scan_normal_glue:
  \__unravel_scan_mu_glue:

```

```

2515   \cs_new_protected:Npn \__unravel_scan_normal_glue:
2516     { \__unravel_scan_glue:n { 2 } }
2517   \cs_new_protected:Npn \__unravel_scan_mu_glue:
2518     { \__unravel_scan_glue:n { 3 } }

```

(End definition for __unravel_scan_normal_glue: and __unravel_scan_mu_glue:.)

```

\__unravel_scan_glue:n

```

```

2519   \cs_new_protected:Npn \__unravel_scan_glue:n #1
2520     {
2521       \__unravel_prev_input_gpush:
2522       \__unravel_scan_signs:
2523       \__unravel_prev_input_gpush:
2524       \__unravel_set_cmd:
2525       \__unravel_cmd_if_internal:TF
2526         {
2527           \__unravel_rescan_something_internal:n {#1}
2528           \int_case:nnF \g__unravel_val_level_int
2529             {
2530               { 0 } { \__unravel_scan_dim_unit:n {#1} \c_false_bool }
2531               { 1 } { }
2532             }
2533           { \__unravel_break:w }
2534         }

```

```

2535     { \__unravel_back_input: \__unravel_scan_dimen:nN {#1} \c_false_bool }
2536 \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
2537 \__unravel_prev_input_gpush:
2538 \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
2539 \__unravel_scan_keyword:nT { pPlLuUsS }
2540   { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2541 \__unravel_scan_keyword:nT { mMiInNuUsS }
2542   { \__unravel_scan_dimen:nN {#1} \c_true_bool }
2543 \__unravel_break_point:
2544 \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
2545 \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2546 }

```

(End definition for __unravel_scan_glue:n.)

__unravel_scan_file_name:

```

2547 \cs_new_protected:Npn \__unravel_scan_file_name:
2548 {
2549   \__unravel_get_x_non_relax:
2550   \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
2551   { \__unravel_scan_group_x:N \c_false_bool }
2552   {
2553     \__unravel_back_input:
2554     \bool_gset_true:N \g__unravel_name_in_progress_bool
2555     \bool_gset_false:N \g__unravel_quotes_bool
2556     \__unravel_get_x_non_blank:
2557     \__unravel_scan_file_name_loop:
2558     \bool_gset_false:N \g__unravel_name_in_progress_bool
2559     \__unravel_prev_input_silent:n { ~ }
2560   }
2561 }
2562 \cs_new_protected:Npn \__unravel_scan_file_name_loop:
2563 {
2564   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
2565   { \__unravel_back_input: }
2566   {
2567     \tl_set:Nx \l__unravel_tmpa_tl
2568     { \__unravel_token_to_char:N \l__unravel_head_token }
2569     \tl_if_eq:NNT \l__unravel_tmpa_tl \c__unravel_dq_tl
2570     {
2571       \bool_if:NTF \g__unravel_quotes_bool
2572       { \bool_set_false:N } { \bool_set_true:N } \g__unravel_quotes_bool
2573     }
2574     \bool_if:NTF \g__unravel_quotes_bool
2575     { \use:n } { \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl }
2576     {
2577       \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
2578       \__unravel_get_x_next:
2579       \__unravel_scan_file_name_loop:
2580     }
2581   }
2582 }
2583 \bool_new:N \g__unravel_quotes_bool

```

(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.

```

2584 \cs_new_protected:Npn \__unravel_scan_r_token:
2585 {
2586   \bool_do_while:nn
2587     { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
2588     { \__unravel_get_next: }
2589   \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
2590   {
2591     \__unravel_error:nnnnn { missing-cs } { } { } { } { }
2592     \__unravel_back_input:
2593     \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
2594   }
2595   \__unravel_prev_input_silent:V \l__unravel_head_tl
2596   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
2597 }

```

(End definition for `__unravel_scan_r_token:`.)

`__unravel_scan_toks_to_str:`

```

2598 \cs_new_protected:Npn \__unravel_scan_toks_to_str:
2599 {
2600   \__unravel_prev_input_gpush:
2601   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2602   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2603   \__unravel_prev_input_silent:x
2604   { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
2605 }

```

(End definition for `__unravel_scan_toks_to_str:`.)

`__unravel_scan_pdf_ext_toks:`

```

2606 \cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
2607 {
2608   \__unravel_prev_input_gpush:
2609   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
2610   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2611   \__unravel_prev_input_silent:x
2612   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
2613 }

```

(End definition for `__unravel_scan_pdf_ext_toks:`.)

`__unravel_scan_toks:NN` The boolean `#1` is true if we are making a definition (then we start by scanning the parameter text), false if we are simply scanning a general text. The boolean `#2` is true if we need to expand, false otherwise (for instance for `\lowercase`).

```

2614 \cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
2615 {
2616   \bool_if:NT #1 { \__unravel_scan_param: }
2617   \__unravel_scan_left_brace:
2618   \bool_if:NTF #2
2619     { \__unravel_scan_group_x:N #1 }
2620     { \__unravel_scan_group_n:N #1 }
2621 }

```

(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`.

```
2622 \cs_new_protected:Npn \_unravel_scan_param:
2623   {
2624     \tl_clear:N \l\_unravel_tmpa_tl
2625     \_unravel_scan_param_aux:
2626     \tl_put_right:NV \l\_unravel_defining_tl \l\_unravel_tmpa_tl
2627     \_unravel_prev_input_silent:V \l\_unravel_tmpa_tl
2628   }
2629 \cs_new_protected:Npn \_unravel_scan_param_aux:
2630   {
2631     \_unravel_get_next:
2632     \tl_concat:NNN \l\_unravel_tmpa_tl
2633     \l\_unravel_tmpa_tl \l\_unravel_head_tl
2634     \tl_if_empty:NTF \l\_unravel_head_tl
2635     { \_unravel_back_input: } { \_unravel_scan_param_aux: }
2636   }
```

(End definition for `_unravel_scan_param:` and `_unravel_scan_param_aux:.`)

`_unravel_scan_group_n:N` The boolean `#1` is true if we are making a definition, false otherwise. In both cases put the open brace back and grab the first item. The only difference is that when making a definition we store the data into `\l_unravel_defining_tl` as well.

```
2637 \cs_new_protected:Npn \_unravel_scan_group_n:N #1
2638   {
2639     \gtl_set_eq:NN \l\_unravel_head_gtl \c_group_begin_gtl
2640     \_unravel_back_input:
2641     \_unravel_input_gpop_item:NF \l\_unravel_head_tl
2642     {
2643       \_unravel_error:nnnnn { runaway-text } { } { } { } { }
2644       \_unravel_exit_hard:w
2645     }
2646     \tl_set:Nx \l\_unravel_head_tl { { \exp_not:V \l\_unravel_head_tl } }
2647     \bool_if:NT #1
2648     { \tl_put_right:NV \l\_unravel_defining_tl \l\_unravel_head_tl }
2649     \_unravel_prev_input_silent:V \l\_unravel_head_tl
2650   }
```

(End definition for `_unravel_scan_group_n:N`.)

`_unravel_scan_group_x:N` The boolean `#1` is true if we are making a definition, false otherwise.

```
2651 \cs_new_protected:Npn \_unravel_scan_group_x:N #1
2652   {
2653     \_unravel_input_gpop_tl:N \l\_unravel_head_tl
2654     \_unravel_back_input:V \l\_unravel_head_tl
2655     \bool_if:NTF #1
2656     {
2657       \_unravel_prev_input_silent:V \c_left_brace_str
2658       \tl_put_right:Nn \l\_unravel_defining_tl { { \if_false: } \fi: }
2659       \_unravel_scan_group_xdef:n { 1 }
2660     }
```



```

2661     {
2662     \__unravel_prev_input_gpush_gtl:
2663     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2664     \__unravel_scan_group_x:n { 1 }
2665     \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
2666     \__unravel_prev_input_silent:x
2667     { \gtl_left_tl:N \l__unravel_tmpb_gtl }
2668     }
2669   }

```

(End definition for __unravel_scan_group_x:N.)

__unravel_scan_group_xdef:n This is to scan the replacement text of an \edef or \xdef. The integer #1 counts the brace balance.

```

2670 \cs_new_protected:Npn \__unravel_scan_group_xdef:n #1
2671   {
2672     \__unravel_get_token_xdef:
2673     \tl_if_empty:NTF \l__unravel_head_tl
2674     {
2675       \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2676       {
2677         \__unravel_prev_input_silent:V \c_left_brace_str
2678         \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
2679         \__unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
2680       }
2681       {
2682         \__unravel_prev_input_silent:V \c_right_brace_str
2683         \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
2684         \int_compare:nNnF {#1} = 1
2685         { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
2686       }
2687     }
2688     {
2689       \__unravel_prev_input_silent:V \l__unravel_head_tl
2690       \tl_put_right:Nx \l__unravel_defining_tl
2691       { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
2692       \__unravel_scan_group_xdef:n {#1}
2693     }
2694   }
2695 \cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }

```

(End definition for __unravel_scan_group_xdef:n.)

__unravel_scan_group_x:n

```

2696 \cs_new_protected:Npn \__unravel_scan_group_x:n #1
2697   {
2698     \__unravel_get_token_x:
2699     \__unravel_prev_input_gtl:N \l__unravel_head_gtl
2700     \tl_if_empty:NTF \l__unravel_head_tl
2701     {
2702       \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
2703       { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
2704       {
2705         \int_compare:nNnF {#1} = 1

```

```

2706         { \_unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
2707     }
2708 }
2709 { \_unravel_scan_group_x:n {#1} }
2710 }
2711 \cs_generate_variant:Nn \_unravel_scan_group_x:n { f }

```

(End definition for _unravel_scan_group_x:n.)

_unravel_scan_alt_rule:

```

2712 \cs_new_protected:Npn \_unravel_scan_alt_rule:
2713 {
2714   \_unravel_scan_keyword:nTF { wWiIdDtThH }
2715   {
2716     \_unravel_scan_normal_dimen:
2717     \_unravel_scan_alt_rule:
2718   }
2719   {
2720     \_unravel_scan_keyword:nTF { hHeEiIgGhHtT }
2721     {
2722       \_unravel_scan_normal_dimen:
2723       \_unravel_scan_alt_rule:
2724     }
2725     {
2726       \_unravel_scan_keyword:nT { dDeEpPtThH }
2727       {
2728         \_unravel_scan_normal_dimen:
2729         \_unravel_scan_alt_rule:
2730       }
2731     }
2732   }
2733 }

```

(End definition for _unravel_scan_alt_rule:.)

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

```

2734 \cs_new_protected:Npn \_unravel_scan_spec:
2735 {
2736   \_unravel_scan_keyword:nTF { tToO } { \_unravel_scan_normal_dimen: }
2737   {
2738     \_unravel_scan_keyword:nT { sSpPrReEaAdD }
2739     { \_unravel_scan_normal_dimen: }
2740   }
2741   \_unravel_scan_left_brace:
2742 }

```

(End definition for _unravel_scan_spec:.)

2.8 Working with boxes

_unravel_do_box:N When this procedure is called, the last item in the previous-input sequence is

- empty if the box is meant to be put in the input stream,
- `\setbox<int>` if it is meant to be stored somewhere,

- `\moveright<dim>`, `\moveleft<dim>`, `\lower<dim>`, `\raise<dim>` if it is meant to be shifted,
- `\leaders` or `\cleaders` or `\xleaders`, in which case the argument is `\c_true_bool` (otherwise `\c_false_bool`).

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

```

2743 \cs_new_protected:Npn __unravel_do_box:N #1
2744 {
2745   \__unravel_get_x_non_relax:
2746   \__unravel_set_cmd:
2747   \int_compare:nNnTF
2748     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
2749     { \__unravel_do_begin_box:N #1 }
2750     {
2751       \bool_if:NTF #1
2752       {
2753         \int_case:nnTF \l__unravel_head_cmd_int
2754         {
2755           { \__unravel_tex_use:n { hrule } } { }
2756           { \__unravel_tex_use:n { vrule } } { }
2757         }
2758         { \__unravel_do_leaders_rule: }
2759         { \__unravel_do_box_error: }
2760       }
2761       { \__unravel_do_box_error: }
2762     }
2763 }

```

(End definition for `__unravel_do_box:N`.)

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

```

2764 \cs_new_protected:Npn __unravel_do_box_error:
2765 {
2766   \__unravel_back_input:
2767   \__unravel_error:nnnnn { missing-box } { } { } { } { }
2768   \__unravel_prev_input_gpop:N \l__unravel_head_tl
2769   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2770 }

```

(End definition for `__unravel_do_box_error:.`)

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

```

2771 \cs_new_protected:Npn __unravel_do_begin_box:N #1
2772 {
2773   \__unravel_prev_input:V \l__unravel_head_tl
2774   \int_case:nnTF \l__unravel_head_char_int

```

```

2775     {
2776     { 0 } { \__unravel_scan_int: } % box
2777     { 1 } { \__unravel_scan_int: } % copy
2778     { 2 } { } % lastbox
2779     { 3 } % vsplit
2780     {
2781     \__unravel_scan_int:
2782     \__unravel_scan_to:
2783     \__unravel_scan_normal_dimen:
2784     }
2785     }
2786     { \__unravel_do_simple_box:N #1 }
2787     { \__unravel_do_box_explicit:N #1 }
2788     }

```

(End definition for __unravel_do_begin_box:N.)

__unravel_do_simple_box:N For leaders, we need to fetch a glue. In all cases, retrieve the box construction (such as \raise3pt\vsplit7to5em). Finally, let T_EX run the code and print what we have done. In the case of \shipout, check that \mag has a value between 1 and 32768.

```

2789 \cs_new_protected:Npn \__unravel_do_simple_box:N #1
2790 {
2791   \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
2792   {
2793     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2794     \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
2795     { \__unravel_prepare_mag: }
2796     \tl_use:N \l__unravel_head_tl \scan_stop:
2797     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2798     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2799   }
2800 }

```

(End definition for __unravel_do_simple_box:N.)

__unravel_do_leaders_fetch_skip:

```

2801 \cs_new_protected:Npn \__unravel_do_leaders_fetch_skip:
2802 {
2803   \__unravel_get_x_non_relax:
2804   \__unravel_set_cmd:
2805   \int_compare:nNnTF \l__unravel_head_cmd_int
2806   = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
2807   {
2808     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
2809     \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
2810     \__unravel_do_append_glue:
2811   }
2812   {
2813     \__unravel_back_input:
2814     \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
2815     \__unravel_prev_input_gpop:N \l__unravel_head_tl
2816     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
2817   }
2818 }

```

(End definition for `_unravel_do_leaders_fetch_skip:`)

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

```
2819 \cs_new_protected:Npn \_unravel_do_box_explicit:N #1
2820   {
2821     \token_if_eq_meaning:NNTF \l__unravel_head_token \_unravel_hbox:w
2822       { \_unravel_box_hook:N \tex_everyhbox:D }
2823       { \_unravel_box_hook:N \tex_everyvbox:D }
2824     \_unravel_scan_spec:
2825     \_unravel_prev_input_gpop:N \l__unravel_head_tl
2826     \_unravel_set_action_text:x
2827     { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
2828     \seq_push:Nf \l__unravel_leaders_box_seq
2829     { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
2830     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
2831     \gtl_gconcat:NNN \g__unravel_output_gtl
2832     \g__unravel_output_gtl \c_group_begin_gtl
2833     \tl_use:N \l__unravel_head_tl
2834     \c_group_begin_token \_unravel_box_hook_end:
2835   }
```

(End definition for `_unravel_do_box_explicit:N`.)

`_unravel_box_hook:N` Used to capture the contents of an `\everyhbox` or similar, without altering `\everyhbox`
`_unravel_box_hook:w` too much (just add one token at the start). The various o-expansions remove `\prg_do_-`
`_unravel_box_hook_end:` `nothing:`, used to avoid losing braces.

```
2836 \cs_new_protected:Npn \_unravel_box_hook:N #1
2837   {
2838     \tl_set:NV \l__unravel_tmpa_tl #1
2839     \str_if_eq:eeF
2840     { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \_unravel_box_hook:w }
2841     {
2842       \_unravel_exp_args:Nx #1
2843       {
2844         \exp_not:n { \_unravel_box_hook:w \prg_do_nothing: }
2845         \exp_not:V #1
2846       }
2847     }
2848     \cs_gset_protected:Npn \_unravel_box_hook:w ##1 \_unravel_box_hook_end:
2849     {
2850       \exp_args:No #1 {##1}
2851       \cs_gset_eq:NN \_unravel_box_hook:w \prg_do_nothing:
2852       \gtl_clear:N \l__unravel_after_group_gtl
2853       \_unravel_print_action:
```

```

2854     \_unravel_back_input:o {##1}
2855     \_unravel_set_action_text:x
2856         { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
2857     \tl_if_empty:oF {##1} { \_unravel_print_action: }
2858 }
2859 }
2860 \cs_new_eq:NN \_unravel_box_hook:w \prg_do_nothing:
2861 \cs_new_eq:NN \_unravel_box_hook_end: \prg_do_nothing:

```

(End definition for _unravel_box_hook:N, _unravel_box_hook:w, and _unravel_box_hook_end:.)

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

```

2862 \cs_new_protected:Npn \_unravel_do_leaders_rule:
2863 {
2864     \_unravel_prev_input:V \l__unravel_head_tl
2865     \_unravel_scan_alt_rule:
2866     \_unravel_do_leaders_fetch_skip:
2867 }

```

(End definition for _unravel_do_leaders_rule:.)

2.9 Paragraphs

_unravel_charcode_if_safe:nTF

```

2868 \prg_new_protected_conditional:Npnn \_unravel_charcode_if_safe:n #1 { TF }
2869 {
2870     \bool_if:nTF
2871     {
2872         \int_compare_p:n { #1 = '!' }
2873         || \int_compare_p:n { ' ' <= #1 <= '[' }
2874         || \int_compare_p:n { #1 = ']' }
2875         || \int_compare_p:n { ' ' <= #1 <= 'z' }
2876     }
2877     { \prg_return_true: }
2878     { \prg_return_false: }
2879 }

```

(End definition for _unravel_charcode_if_safe:nTF.)

_unravel_char:n

_unravel_char:V

_unravel_char:x

```

2880 \cs_new_protected:Npn \_unravel_char:n #1
2881 {
2882     \tex_char:D #1 \scan_stop:
2883     \_unravel_charcode_if_safe:nTF {#1}
2884     {
2885         \tl_set:Nx \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } }
2886         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2887         \_unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2888     }
2889     {
2890         \tl_set:Nx \l__unravel_tmpa_tl
2891         { \exp_not:N \char \int_eval:n {#1} ~ }

```

```

2892     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2893     \__unravel_print_action:x
2894     { " \char_generate:nn {#1} { 12 } " = \tl_to_str:N \l__unravel_tmpa_tl }
2895   }
2896 }
2897 \cs_generate_variant:Nn \__unravel_char:n { V }
2898 \cs_new_protected:Npn \__unravel_char:x
2899 { \__unravel_exp_args:Nx \__unravel_char:n }

```

(End definition for `__unravel_char:n`.)

```

\__unravel_char_in_mmode:n
\__unravel_char_in_mmode:V
\__unravel_char_in_mmode:x
2900 \cs_new_protected:Npn \__unravel_char_in_mmode:n #1
2901 {
2902   \int_compare:nNnTF { \tex_mathcode:D #1 }
2903   = { \sys_if_engine luatex:TF { "1000000 } { "8000 } }
2904   { % math active
2905     \__unravel_active_do:nn {#1} { \gtl_set:Nn \l__unravel_head_gtl }
2906     \__unravel_back_input:
2907     \__unravel_print_action:x
2908     { \char_generate:nn {#1} { 12 } ~ active }
2909   }
2910   { \__unravel_char:n {#1} }
2911 }
2912 \cs_generate_variant:Nn \__unravel_char_in_mmode:n { V }
2913 \cs_new_protected:Npn \__unravel_char_in_mmode:x
2914 { \__unravel_exp_args:Nx \__unravel_char_in_mmode:n }

```

(End definition for `__unravel_char_in_mmode:n`.)

```

\__unravel_mathchar:n
\__unravel_mathchar:x
2915 \cs_new_protected:Npn \__unravel_mathchar:n #1
2916 {
2917   \tex_mathchar:D #1 \scan_stop:
2918   \tl_set:Nx \l__unravel_tmpa_tl
2919   { \exp_not:N \mathchar " \int_to_hex:n {#1} ~ }
2920   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
2921   \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
2922 }
2923 \cs_new_protected:Npn \__unravel_mathchar:x
2924 { \__unravel_exp_args:Nx \__unravel_mathchar:n }

```

(End definition for `__unravel_mathchar:n`.)

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

```

2925 \cs_new_protected:Npn \__unravel_new_graf:N #1
2926 {
2927   \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
2928   \__unravel_everypar:w { }
2929   \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
2930   \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl

```

```

2931   \__unravel_back_input:V \l__unravel_tmpa_tl
2932   \__unravel_print_action:x
2933   {
2934     \g__unravel_action_text_str \c_space_tl : ~
2935     \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
2936   }
2937 }

```

(End definition for __unravel_new_graf:N.)

__unravel_par_if_hmode: This is like the end_graf procedure in T_EX.

```

2938 \cs_new_protected:Npn \__unravel_par_if_hmode:
2939   { \mode_if_horizontal:T { \__unravel_par: } }

```

(End definition for __unravel_par_if_hmode:.)

__unravel_par:

```

2940 \cs_new_protected:Npn \__unravel_par:
2941   {
2942     \tex_par:D
2943     \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
2944     \__unravel_print_action:x { Paragraph~end. }
2945   }

```

(End definition for __unravel_par:.)

__unravel_build_page:

```

2946 \cs_new_protected:Npn \__unravel_build_page:
2947   {
2948   }

```

(End definition for __unravel_build_page:.)

2.10 Groups

\l__unravel_choice_int Used by \mathchoice etc to keep track of which argument we are currently in.

```

2949 \int_new:N \l__unravel_choice_int

```

(End definition for \l__unravel_choice_int.)

__unravel_handle_right_brace: When an end-group character is sensed, the result depends on the current group type. Suppress the after_group tokens in \discretionary or \mathchoice.

```

2950 \cs_new_protected:Npn \__unravel_handle_right_brace:
2951   {
2952     \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
2953     {
2954       \gtl_gconcat:NNN \g__unravel_output_gtl
2955         \g__unravel_output_gtl \c_group_end_gtl
2956       \int_case:nnF \__unravel_currentgrouptype:
2957         {
2958           { 10 } { } % disc
2959           { 13 } { } % math_choice
2960         }
2961       { \__unravel_back_input_gtl:N \l__unravel_after_group_gtl }
2962     } \int_case:nn \__unravel_currentgrouptype:

```



```

2963     {
2964     { 1 } { \_unravel_end_simple_group: } % simple
2965     { 2 } { \_unravel_end_box_group: } % hbox
2966     { 3 } { \_unravel_end_box_group: } % adjusted_hbox
2967     { 4 } { \_unravel_par_if_hmode: \_unravel_end_box_group: } % vbox
2968     { 5 } { \_unravel_par_if_hmode: \_unravel_end_box_group: } % vtop
2969     { 6 } { \_unravel_end_align_group: } % align
2970     { 7 } { \_unravel_end_no_align_group: } % no_align
2971     { 8 } { \_unravel_end_output_group: } % output
2972     { 9 } { \_unravel_end_simple_group: } % math
2973     { 10 } { \_unravel_end_choice_group:NN 2 \discretionary } % disc
2974     { 11 } { \_unravel_par_if_hmode: \_unravel_end_simple_group: } % insert
2975     { 12 } { \_unravel_par_if_hmode: \_unravel_end_simple_group: } % vcenter
2976     { 13 } { \_unravel_end_choice_group:NN 3 \mathchoice } % math_choice
2977     }
2978   }
2979   { % bottom_level, semi_simple, math_shift, math_left
2980     \l_unravel_head_token
2981     \_unravel_print_action:
2982   }
2983 }

```

(End definition for _unravel_handle_right_brace:.)

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

```

2984 \cs_new_protected:Npn \_unravel_end_simple_group:
2985   {
2986     \l_unravel_head_token
2987     \_unravel_print_action:
2988   }

```

(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.

```

2989 \cs_new_protected:Npn \_unravel_end_box_group:
2990   {
2991     \seq_pop:NN \l_unravel_leaders_box_seq \l_unravel_tmpa_tl
2992     \exp_args:No \_unravel_end_box_group_aux:n { \l_unravel_tmpa_tl }
2993   }
2994 \cs_new_protected:Npn \_unravel_end_box_group_aux:n #1
2995   {
2996     \str_if_eq:eeTF {#1} { Z }
2997     { \_unravel_end_simple_group: }
2998     {
2999       \_unravel_get_x_non_relax:
3000       \_unravel_set_cmd:
3001       \int_compare:nNnTF \l_unravel_head_cmd_int
3002         = { \_unravel_tex_use:n { #1 skip } }
3003         {
3004           \tl_put_left:Nn \l_unravel_head_tl { \c_group_end_token }
3005           \_unravel_do_append_glue:
3006         }

```

```

3007     {
3008         \__unravel_back_input:
3009         \c_group_end_token \group_begin: \group_end:
3010         \__unravel_print_action:
3011     }
3012 }
3013 }

```

(End definition for __unravel_end_box_group:.)

__unravel_end_align_group:

```

3014 \cs_new_protected:Npn \__unravel_end_align_group:
3015 {
3016     \__unravel_not_implemented:n { end_align_group }
3017     \__unravel_end_simple_group:
3018 }

```

(End definition for __unravel_end_align_group:.)

__unravel_end_no_align_group:

```

3019 \cs_new_protected:Npn \__unravel_end_no_align_group:
3020 {
3021     \__unravel_not_implemented:n { end_no_align_group }
3022     \__unravel_end_simple_group:
3023 }

```

(End definition for __unravel_end_no_align_group:.)

__unravel_end_output_group:

```

3024 \cs_new_protected:Npn \__unravel_end_output_group:
3025 {
3026     \__unravel_not_implemented:n { end_output_group }
3027     \__unravel_end_simple_group:
3028 }

```

(End definition for __unravel_end_output_group:.)

__unravel_end_choice_group:NN

__unravel_end_choice_group:nN

```

3029 \cs_new_protected:Npn \__unravel_end_choice_group:NN #1#2
3030 {
3031     \int_compare:nNnTF \l__unravel_choice_int > {#1}
3032     {
3033         \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3034         \c_group_end_token
3035         \__unravel_print_action:x
3036         { \token_to_str:N #2 \prg_replicate:nn { #1 + 1 } { {...} } }
3037     }
3038     { \exp_args:NV \__unravel_end_choice_group:nN \l__unravel_choice_int #2 }
3039 }
3040 \cs_new_protected:Npn \__unravel_end_choice_group:nN #1#2
3041 {
3042     \__unravel_scan_left_brace:
3043     \gtl_gconcat:NNN \g__unravel_output_gtl
3044     \g__unravel_output_gtl \c_group_begin_gtl
3045     \__unravel_back_input_gtl:N \l__unravel_after_group_gtl

```

```

3046 \use:n \c_group_end_token
3047 \use:n \c_group_begin_token
3048 \int_set:Nn \l__unravel_choice_int { #1 + 1 }
3049 \gtl_clear:N \l__unravel_after_group_gtl
3050 \__unravel_print_action:x
3051 {
3052   \token_to_str:N #2
3053   \prg_replicate:nn {#1} { { ... } }
3054   \iow_char:N \{
3055 }
3056 }

```

(End definition for __unravel_end_choice_group:NN and __unravel_end_choice_group:nN.)

__unravel_off_save:

```

3057 \cs_new_protected:Npn \__unravel_off_save:
3058 {
3059   \int_compare:nNnTF \__unravel_currentgroupstype: = { 0 }
3060   { % bottom-level
3061     \__unravel_error:nxxxx { extra-close }
3062     { \token_to_meaning:N \l__unravel_head_token } { } { } { }
3063   }
3064   {
3065     \__unravel_back_input:
3066     \int_case:nnF \__unravel_currentgroupstype:
3067     {
3068       { 14 } % semi_simple_group
3069       { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
3070       { 15 } % math_shift_group
3071       { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
3072       { 16 } % math_left_group
3073       { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
3074     }
3075     { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
3076     \__unravel_back_input:
3077     \__unravel_error:nxxxx { off-save }
3078     { \gtl_to_str:N \l__unravel_head_gtl } { } { } { }
3079   }
3080 }

```

(End definition for __unravel_off_save:.)

2.11 Modes

```

\__unravel_mode_math:n
\__unravel_mode_non_math:n
\__unravel_mode_vertical:n
3081 \cs_new_protected:Npn \__unravel_mode_math:n #1
3082 { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
3083 \cs_new_protected:Npn \__unravel_mode_non_math:n #1
3084 { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
3085 \cs_new_protected:Npn \__unravel_mode_vertical:n #1
3086 {
3087   \mode_if_math:TF
3088   { \__unravel_insert_dollar_error: }
3089   { \mode_if_horizontal:TF { \__unravel_head_for_vmodes: } {#1} }

```

```

3090 }
3091 \cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
3092 {
3093   \mode_if_vertical:TF
3094     { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
3095     { #1 }
3096 }

```

(End definition for `__unravel_mode_math:n`, `__unravel_mode_non_math:n`, and `__unravel_mode_non_vertical:n`.)

`__unravel_head_for_vmode:` See T_EX's `head_for_vmode`.

```

3097 \cs_new_protected:Npn \__unravel_head_for_vmode:
3098 {
3099   \mode_if_inner:TF
3100     {
3101       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrulerule:D
3102         {
3103           \__unravel_error:nnnnn { hrulerule-bad-mode } { } { } { } { }
3104           \__unravel_print_action:
3105         }
3106         { \__unravel_off_save: }
3107     }
3108     {
3109       \__unravel_back_input:
3110       \gtl_set:Nn \l__unravel_head_gtl { \par }
3111       \__unravel_back_input:
3112     }
3113 }

```

(End definition for `__unravel_head_for_vmode:.`)

`__unravel_goto_inner_math:`

```

3114 \cs_new_protected:Npn \__unravel_goto_inner_math:
3115 {
3116   \__unravel_box_hook:N \tex_everymath:D
3117   $ % $
3118   \__unravel_box_hook_end:
3119 }

```

(End definition for `__unravel_goto_inner_math:.`)

`__unravel_goto_display_math:`

```

3120 \cs_new_protected:Npn \__unravel_goto_display_math:
3121 {
3122   \__unravel_box_hook:N \tex_everydisplay:D
3123   $ $
3124   \__unravel_box_hook_end:
3125 }

```

(End definition for `__unravel_goto_display_math:.`)

`__unravel_after_math:` In display math mode, or in a group started by `\eqno` or `\leqno` (namely in inner math mode with non-zero `\l__unravel_choice_int`), search for another `$`; otherwise simply close the inner math.

```

3126 \cs_new_protected:Npn \__unravel_after_math:
3127 {
3128   \mode_if_inner:TF
3129     { \int_compare:nNnTF \l__unravel_choice_int > 0 }
3130     { \use_i:nn }
3131     {
3132       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3133       \__unravel_get_x_next:
3134       \token_if_eq_catcode:NNF
3135         \l__unravel_head_token \c_math_toggle_token
3136         {
3137           \__unravel_back_input:
3138           \tl_set:Nn \l__unravel_head_tl { $ } % $
3139           \__unravel_error:nmnnn { missing-dollar } { } { } { } { }
3140         }
3141       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3142       \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3143       $ $
3144     }
3145     {
3146       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3147       \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3148       $ % $
3149     }
3150   \__unravel_print_action:
3151 }

```

(End definition for `__unravel_after_math:.`)

2.12 Commands

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

`__unravel_forbidden_case:`

```

3152 \cs_new_protected:Npn \__unravel_forbidden_case:
3153 { \__unravel_tex_error:nV { forbidden-case } \l__unravel_head_tl }

```

(End definition for `__unravel_forbidden_case:.`)

2.12.1 Characters: from 0 to 15

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

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

- `tab_mark=4` for `\span`
- `alignment_char=4` for alignment tab characters (catcode 4).
- `car_ret=5` for `\cr` and `\crr`.
- `macro_char=6` for macro parameter characters (catcode 6).
- `superscript_char=7` for superscript characters (catcode 7).
- `subscript_char=8` for subscript characters (catcode 8).
- `endv=9` for ?.
- `blank_char=10` for blank spaces (catcode 10).
- `the_char=11` for letters (catcode 11).
- `other_char=12` for other characters (catcode 12).
- `par_end=13` for `\par`.
- `stop=14` for `\end` and `\dump`.
- `delim_num=15` for `\delimiter`.

Not implemented at all: `endv`.

`\relax` does nothing.

```

3154 \__unravel_new_tex_cmd:nn { relax } % 0
3155 {
3156   \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:
3157   {
3158     \exp_after:wN \__unravel_token_if_expandable:NTF \l__unravel_head_tl
3159     {
3160       \__unravel_set_action_text:x
3161       { \iow_char:N \notexpanded: \g__unravel_action_text_str }
3162     }
3163     { }
3164   }
3165   \__unravel_print_action:
3166 }

```

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.

```

3167 \__unravel_new_tex_cmd:nn { begin-group_char } % 1
3168 {
3169   \gtl_gconcat:NNN \g__unravel_output_gtl
3170   \g__unravel_output_gtl \c_group_begin_gtl
3171   \__unravel_print_action:
3172   \l__unravel_head_token
3173   \gtl_clear:N \l__unravel_after_group_gtl
3174 }
3175 \__unravel_new_tex_cmd:nn { end-group_char } % 2
3176 { \__unravel_handle_right_brace: }

```

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

```

3177 \__unravel_new_tex_cmd:nn { math_char } % 3
3178 {
3179   \__unravel_mode_non_vertical:n
3180   {
3181     \mode_if_math:TF
3182     {
3183       \int_compare:nNnTF
3184         \__unravel_currentgrouptype: = { 15 } % math_shift_group
3185         { \__unravel_after_math: }
3186         { \__unravel_off_save: }
3187     }
3188   {
3189     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3190     \__unravel_get_next:
3191     \token_if_eq_catcode:NNTF
3192       \l__unravel_head_token \c_math_toggle_token
3193     {
3194       \mode_if_inner:TF
3195         { \__unravel_back_input: \__unravel_goto_inner_math: }
3196         {
3197           \gtl_gput_right:NV
3198             \g__unravel_output_gtl \l__unravel_head_tl
3199             \__unravel_goto_display_math:
3200         }
3201     }
3202     { \__unravel_back_input: \__unravel_goto_inner_math: }
3203   }
3204 }
3205 }

```

Some commands are errors when they reach T_EX's stomach. Among others, `tab_mark=alignment_char`, `car_ret` and `macro_char`. We let T_EX insert the proper error.

```

3206 \__unravel_new_tex_cmd:nn { alignment_char } % 4
3207 { \l__unravel_head_token \__unravel_print_action: }
3208 \__unravel_new_tex_cmd:nn { car_ret } % 5
3209 { \l__unravel_head_token \__unravel_print_action: }
3210 \__unravel_new_tex_cmd:nn { macro_char } % 6
3211 { \l__unravel_head_token \__unravel_print_action: }
3212 \__unravel_new_tex_cmd:nn { superscript_char } % 7
3213 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3214 \__unravel_new_tex_cmd:nn { subscript_char } % 8
3215 { \__unravel_mode_math:n { \__unravel_sub_sup: } }
3216 \cs_new_protected:Npn \__unravel_sub_sup:
3217 {
3218   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3219   \__unravel_print_action:
3220   \__unravel_do_one_atom:
3221 }
3222 \cs_new_protected:Npn \__unravel_do_one_atom:
3223 {
3224   \__unravel_get_x_non_relax:
3225   \__unravel_set_cmd:

```

```

3226 \int_case:nnTF \l__unravel_head_cmd_int
3227 {
3228   { \__unravel_tex_use:n { the_char } }
3229     { \__unravel_prev_input:V \l__unravel_head_tl }
3230   { \__unravel_tex_use:n { other_char } }
3231     { \__unravel_prev_input:V \l__unravel_head_tl }
3232   { \__unravel_tex_use:n { char_given } }
3233     { \__unravel_prev_input:V \l__unravel_head_tl }
3234   { \__unravel_tex_use:n { char_num } }
3235     {
3236       \__unravel_prev_input:V \l__unravel_head_tl
3237       \__unravel_scan_int:
3238     }
3239   { \__unravel_tex_use:n { math_char_num } }
3240     {
3241       \__unravel_prev_input:V \l__unravel_head_tl
3242       \__unravel_scan_int:
3243     }
3244   { \__unravel_tex_use:n { math_given } }
3245     { \__unravel_prev_input:V \l__unravel_head_tl }
3246   { \__unravel_tex_use:n { delim_num } }
3247     { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
3248   }
3249   {
3250     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3251     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3252     \tl_use:N \l__unravel_head_tl \scan_stop:
3253   }
3254   {
3255     \__unravel_back_input:
3256     \__unravel_scan_left_brace:
3257     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3258     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3259     \gtl_gconcat:NNN \g__unravel_output_gtl
3260     \g__unravel_output_gtl \c_group_begin_gtl
3261     \tl_use:N \l__unravel_head_tl \c_group_begin_token
3262   }
3263   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3264 }
3265 \__unravel_new_tex_cmd:nn { endv } % 9
3266 {
3267   \__unravel_mode_non_math:n
3268   {
3269     \__unravel_not_implemented:n { alignments }
3270   }
3271 }

```

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.

```

3272 \__unravel_new_tex_cmd:nn { blank_char } % 10
3273 {
3274   \mode_if_horizontal:T
3275   {
3276     \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }

```



```

3277     \l__unravel_head_token
3278   }
3279   \__unravel_print_action:
3280 }
Letters and other characters leave vertical mode.
3281 \__unravel_new_tex_cmd:nn { the_char } % 11
3282 {
3283   \__unravel_mode_non_vertical:n
3284   {
3285     \tl_set:Nx \l__unravel_tmpa_tl
3286     { ' \__unravel_token_to_char:N \l__unravel_head_token }
3287     \mode_if_math:TF
3288     { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
3289     { \__unravel_char:V \l__unravel_tmpa_tl }
3290   }
3291 }
3292 \__unravel_new_eq_tex_cmd:nn { other_char } { the_char } % 12
3293 \__unravel_new_tex_cmd:nn { par_end } % 13
3294 {
3295   \__unravel_mode_non_math:n
3296   {
3297     \mode_if_vertical:TF
3298     { \__unravel_par: }
3299     {
3300       % if align_state<0 then off_save;
3301       \__unravel_par_if_hmode:
3302       \mode_if_vertical:T
3303       { \mode_if_inner:F { \__unravel_build_page: } }
3304     }
3305   }
3306 }
3307 \__unravel_new_tex_cmd:nn { stop } % 14
3308 {
3309   \__unravel_mode_vertical:n
3310   {
3311     \mode_if_inner:TF
3312     { \__unravel_forbidden_case: }
3313     {
3314       % ^^A todo: unless its_all_over
3315       \int_gdecr:N \g__unravel_ends_int
3316       \int_compare:nNnTF \g__unravel_ends_int > 0
3317       {
3318         \__unravel_back_input:
3319         \__unravel_back_input:n
3320         {
3321           \__unravel_hbox:w to \tex_hsize:D { }
3322           \tex_vfill:D
3323           \tex_penalty:D - '10000000000 ~
3324         }
3325         \__unravel_build_page:
3326         \__unravel_print_action:x { End-everything! }
3327       }
3328     }

```

```

3329         \l__unravel_print_outcome:
3330         \l__unravel_head_token
3331     }
3332 }
3333 }
3334 }
3335 \l__unravel_new_tex_cmd:nn { delim_num } % 15
3336 {
3337     \l__unravel_mode_math:n
3338     {
3339         \l__unravel_prev_input_gpush:N \l__unravel_head_tl
3340         \l__unravel_print_action:
3341         \l__unravel_scan_int:
3342         \l__unravel_prev_input_gpop:N \l__unravel_head_tl
3343         \tl_use:N \l__unravel_head_tl \scan_stop:
3344         \l__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3345     }
3346 }

```

2.12.2 Boxes: from 16 to 31

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

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

```

3347 \__unravel_new_tex_cmd:nn { char_num } % 16
3348 {
3349   \__unravel_mode_non_vertical:n
3350   {
3351     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3352     \__unravel_print_action:
3353     \__unravel_scan_int:
3354     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3355     \mode_if_math:TF
3356     { \__unravel_char_in_mmode:x { \tl_tail:N \l__unravel_head_tl } }
3357     { \__unravel_char:x { \tl_tail:N \l__unravel_head_tl } }
3358   }
3359 }

```

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.

```

3360 \__unravel_new_tex_cmd:nn { math_char_num } % 17
3361 {
3362   \__unravel_mode_math:n
3363   {
3364     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3365     \__unravel_print_action:
3366     \__unravel_scan_int:
3367     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3368     \__unravel_mathchar:x { \tl_tail:N \l__unravel_head_tl }
3369   }
3370 }

```

```

3371 \__unravel_new_tex_cmd:nn { mark } % 18
3372 {
3373   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3374   \__unravel_print_action:
3375   \int_compare:nNnF \l__unravel_head_char_int = 0
3376   { \__unravel_scan_int: }
3377   \__unravel_prev_input_gpush:
3378   \__unravel_scan_toks:NN \c_false_bool \c_true_bool
3379   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3380   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3381   \__unravel_print_action:x
3382   { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
3383   \tl_put_right:Nx \l__unravel_head_tl
3384   { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
3385   \tl_use:N \l__unravel_head_tl
3386 }

```

We now implement the primitives `\show`, `\showbox`, `\showthe`, `\showlists`, `\showgroups`, `\showtokens` and `\showifs`. Those with no operand are sent to `TEX` after printing the action. Those with operands print first, then scan their operands, then are sent to `TEX`. The case of `\show` is a bit special, as its operand is a single token, which

cannot easily be put into the the previous-input sequence in general. Since no expansion can occur, simply grab the token and show it.

```

3387 \__unravel_new_tex_cmd:nn { xray } % 19
3388 {
3389 \__unravel_prev_input_gpush:N \l__unravel_head_tl
3390 \__unravel_print_action:
3391 \int_case:nnF \l__unravel_head_char_int
3392 {
3393 { 0 }
3394 { % show
3395 \__unravel_get_next:
3396 \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3397 \token_if_eq_meaning:NNTF
3398 \l__unravel_head_token \__unravel_special_relax:
3399 {
3400 \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
3401 \exp_after:wN \exp_not:N \l__unravel_head_tl
3402 }
3403 { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
3404 }
3405 { 2 }
3406 { % showthe
3407 \__unravel_get_x_next:
3408 \__unravel_rescan_something_internal:n { 5 }
3409 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3410 \__unravel_exp_args:Nx \use:n
3411 { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
3412 }
3413 }
3414 { % no operand for showlists, showgroups, showifs
3415 \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
3416 { \__unravel_scan_int: }
3417 \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
3418 { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
3419 \__unravel_prev_input_gpop:N \l__unravel_head_tl
3420 \tl_use:N \l__unravel_head_tl \scan_stop:
3421 }
3422 }
3423 make_box=20 for \box, \copy, \lastbox, \vsplit, \vtop, \vbox, and \hbox (106).
3424 \__unravel_new_tex_cmd:nn { make_box } % 20
3425 {
3426 \__unravel_prev_input_gpush:
3427 \__unravel_back_input:
3428 \__unravel_do_box:N \c_false_bool
3429 }

```

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

```

3429 \cs_new_protected:Npn \__unravel_do_move:
3430 {
3431 \__unravel_prev_input_gpush:N \l__unravel_head_tl
3432 \__unravel_print_action:
3433 \__unravel_scan_normal_dimen:
3434 \__unravel_do_box:N \c_false_bool

```

```

3435 }
(End definition for \_unravel_do_move:.)
hmove=21 for \moveright and \moveleft.
3436 \_unravel_new_tex_cmd:nn { hmove } % 21
3437 {
3438 \mode_if_vertical:TF
3439 { \_unravel_do_move: } { \_unravel_forbidden_case: }
3440 }
vmove=22 for \lower and \raise.
3441 \_unravel_new_tex_cmd:nn { vmove } % 22
3442 {
3443 \mode_if_vertical:TF
3444 { \_unravel_forbidden_case: } { \_unravel_do_move: }
3445 }

```

_unravel_do_unpackage:

```

3446 \cs_new_protected:Npn \_unravel_do_unpackage:
3447 {
3448 \_unravel_prev_input_gpush:N \l__unravel_head_tl
3449 \_unravel_print_action:
3450 \_unravel_scan_int:
3451 \_unravel_prev_input_gpop:N \l__unravel_head_tl
3452 \tl_use:N \l__unravel_head_tl \scan_stop:
3453 \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3454 }
(End definition for \_unravel_do_unpackage:.)
un_hbox=23 for \unhbox and \unhcopy.
3455 \_unravel_new_tex_cmd:nn { un_hbox } % 23
3456 { \_unravel_mode_non_vertical:n { \_unravel_do_unpackage: } }
unvbox=24 for \unvbox, \unvcopy, \pagediscards, and \splitdiscards. The later two take no operands, so we just let TEX do its thing, then we show the action.
3457 \_unravel_new_tex_cmd:nn { un_vbox } % 24
3458 {
3459 \_unravel_mode_vertical:n
3460 {
3461 \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
3462 { \l__unravel_head_token \_unravel_print_action: }
3463 { \_unravel_do_unpackage: }
3464 }
3465 }

```

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

```

3466 \_unravel_new_tex_cmd:nn { remove_item } % 25
3467 { \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.

```

3468 \cs_new_protected:Npn \__unravel_do_append_glue:
3469 {
3470   \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
3471     { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
3472     {
3473       \__unravel_prev_input_gpush:N \l__unravel_head_tl
3474       \__unravel_print_action:
3475       \exp_args:Nf \__unravel_scan_glue:n
3476         { \int_eval:n { \l__unravel_head_char_int - 2 } }
3477       \__unravel_prev_input_gpop:N \l__unravel_head_tl
3478       \tl_use:N \l__unravel_head_tl \scan_stop:
3479       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3480     }
3481 }

```

(End definition for __unravel_do_append_glue:.)

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

```

3482 \__unravel_new_tex_cmd:nn { hskip } % 26
3483 { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }
3484 \__unravel_new_tex_cmd:nn { vskip } % 27
3485 { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }
3486 \__unravel_new_tex_cmd:nn { mskip } % 28
3487 { \__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.

```

3488 \cs_new_protected:Npn \__unravel_do_append_kern:
3489 {
3490   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3491   \__unravel_print_action:
3492   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
3493     { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
3494     { \__unravel_scan_dimen:nN { 3 } \c_false_bool }
3495   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3496   \tl_use:N \l__unravel_head_tl \scan_stop:
3497   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3498 }

```

(End definition for __unravel_do_append_kern:.)

kern=29 for \kern (1).

```

3499 \__unravel_new_tex_cmd:nn { kern } % 29
3500 { \__unravel_do_append_kern: }
3501 \__unravel_new_tex_cmd:nn { mkern } % 30
3502 { \__unravel_mode_math:n { \__unravel_do_append_kern: } }
3503 \__unravel_new_tex_cmd:nn { leader_ship } % 31
3504 {
3505   \__unravel_prev_input_gpush:N \l__unravel_head_tl

```

```

3506   \_unravel_print_action:
3507   \tl_if_head_eq_meaning:VNTF \l__unravel_head_tl \tex_shipout:D
3508     { \_unravel_do_box:N \c_false_bool }
3509     { \_unravel_do_box:N \c_true_bool }
3510   }

```

2.12.3 From 32 to 47

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

```

3511 \_unravel_new_tex_cmd:nn { halign } % 32
3512 { \_unravel_not_implemented:n { halign } }
3513 \_unravel_new_tex_cmd:nn { valign } % 33
3514 { \_unravel_not_implemented:n { valign } }
3515 \_unravel_new_tex_cmd:nn { no_align } % 34
3516 { \l__unravel_head_token \_unravel_print_action: }
3517 \_unravel_new_tex_cmd:nn { vrule } % 35
3518 { \_unravel_mode_non_vertical:n { \_unravel_do_rule: } }
3519 \_unravel_new_tex_cmd:nn { hrule } % 36
3520 { \_unravel_mode_vertical:n { \_unravel_do_rule: } }
3521 \cs_new_protected:Npn \_unravel_do_rule:
3522 {
3523   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3524   \_unravel_print_action:
3525   \_unravel_scan_alt_rule:
3526   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3527   \tl_use:N \l__unravel_head_tl \scan_stop:
3528   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3529 }

```

```

3530 \__unravel_new_tex_cmd:nn { insert } % 37
3531 {
3532   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3533   \__unravel_print_action:
3534   \__unravel_scan_int:
3535   \__unravel_begin_insert_or_adjust:
3536 }
3537 \__unravel_new_tex_cmd:nn { vadjust } % 38
3538 {
3539   \mode_if_vertical:TF
3540   { \__unravel_forbidden_case: }
3541   {
3542     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3543     \__unravel_print_action:
3544     \__unravel_scan_keyword:nTF { pPrReE }
3545     \__unravel_begin_insert_or_adjust:
3546   }
3547 }
3548 \cs_new_protected:Npn \__unravel_begin_insert_or_adjust:
3549 {
3550   \__unravel_scan_left_brace:
3551   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3552   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3553   \gtl_gconcat:NNN \g__unravel_output_gtl
3554   \g__unravel_output_gtl \c_group_begin_gtl
3555   \tl_use:N \l__unravel_head_tl \c_group_begin_token
3556   \__unravel_print_action:x
3557   { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
3558 }
3559 \__unravel_new_tex_cmd:nn { ignore_spaces } % 39
3560 {
3561   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
3562   {
3563     \__unravel_print_action:
3564     \__unravel_get_x_non_blank:
3565     \__unravel_set_cmd:
3566     \__unravel_do_step:
3567   }
3568   { \__unravel_not_implemented:n { pdfprimitive } }
3569 }
3570 \__unravel_new_tex_cmd:nn { after_assignment } % 40
3571 {
3572   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3573   \__unravel_get_next:
3574   \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
3575   \__unravel_print_action:x
3576   {
3577     Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
3578     \gtl_to_str:N \l__unravel_head_gtl
3579   }
3580 }

```

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


```

3581 \__unravel_new_tex_cmd:nn { after_group } % 41
3582 {
3583   \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
3584   \__unravel_get_next:
3585   \int_compare:nNnTF \__unravel_currentgrouptype: = 0
3586   {
3587     \__unravel_print_action:x
3588     {
3589       Aftergroup~(level~0~=>~dropped):~
3590       \tl_to_str:N \l__unravel_tmpa_tl
3591       \gtl_to_str:N \l__unravel_head_gtl
3592     }
3593   }
3594   {
3595     \gtl_concat:NNN \l__unravel_after_group_gtl
3596     \l__unravel_after_group_gtl \l__unravel_head_gtl
3597     \__unravel_print_action:x
3598     {
3599       Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
3600       \gtl_to_str:N \l__unravel_head_gtl
3601     }
3602   }
3603 }
See \__unravel_do_append_glue:.
3604 \__unravel_new_tex_cmd:nn { break_penalty } % 42
3605 {
3606   \__unravel_prev_input_gpush:N \l__unravel_head_tl
3607   \__unravel_print_action:
3608   \__unravel_scan_int:
3609   \__unravel_prev_input_gpop:N \l__unravel_head_tl
3610   \tl_use:N \l__unravel_head_tl \scan_stop:
3611   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3612 }
3613 \__unravel_new_tex_cmd:nn { start_par } % 43
3614 {
3615   \mode_if_vertical:TF
3616   {
3617     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
3618     { \__unravel_new_graf:N \c_false_bool }
3619     { \__unravel_new_graf:N \c_true_bool }
3620   }
3621   {
3622     \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
3623     {
3624       \__unravel_hbox:w width \tex_parindent:D { }
3625       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3626     }
3627     \__unravel_print_action:
3628   }
3629 }
3630 \__unravel_new_tex_cmd:nn { ital_corr } % 44
3631 {
3632   \mode_if_vertical:TF { \__unravel_forbidden_case: }

```

```

3633     { \l__unravel_head_token \__unravel_print_action: }
3634   }

```

__unravel_do_accent:

```

3635 \cs_new_protected:Npn \__unravel_do_accent:
3636   {
3637     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3638     \__unravel_print_action:
3639     \__unravel_scan_int:
3640     \__unravel_do_assignments:
3641     \bool_if:nTF
3642       {
3643         \token_if_eq_catcode_p:NN
3644           \l__unravel_head_token \c_catcode_letter_token
3645         ||
3646         \token_if_eq_catcode_p:NN
3647           \l__unravel_head_token \c_catcode_other_token
3648         ||
3649         \int_compare_p:nNn
3650           \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
3651       }
3652     { \__unravel_prev_input:V \l__unravel_head_tl }
3653     {
3654       \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
3655       {
3656         \__unravel_prev_input:V \l__unravel_head_tl
3657         \__unravel_scan_int:
3658       }
3659       { \__unravel_break:w }
3660     }
3661     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3662     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3663     \tl_use:N \l__unravel_head_tl \scan_stop:
3664     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3665     \__unravel_break_point:
3666   }

```

(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).

```

3667 \cs_new_protected:Npn \__unravel_do_math_accent:
3668   {
3669     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3670     \__unravel_print_action:
3671     \__unravel_scan_int:
3672     \__unravel_do_one_atom:
3673   }

```

(End definition for __unravel_do_math_accent:.)

```

3674 \__unravel_new_tex_cmd:nn { accent } % 45
3675   {
3676     \__unravel_mode_non_vertical:n
3677     {

```

```

3678     \mode_if_math:TF
3679         { \__unravel_do_math_accent: } { \__unravel_do_accent: }
3680     }
3681 }
3682 \__unravel_new_tex_cmd:nn { math_accent } % 46
3683 { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
3684 \__unravel_new_tex_cmd:nn { discretionary } % 47
3685 {
3686     \__unravel_mode_non_vertical:n
3687     {
3688         \int_compare:nNnTF \l__unravel_head_char_int = { 1 }
3689             { \__unravel_output_head_token: }
3690             { \__unravel_do_choice: }
3691     }
3692 }

```

2.12.4 Maths: from 48 to 56

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

```

3693 \__unravel_new_tex_cmd:nn { eq_no } % 48
3694 {
3695     \mode_if_math:TF
3696     {
3697         \mode_if_inner:TF
3698             { \__unravel_off_save: }
3699             {
3700                 \int_compare:nNnTF \tex_currentgrouptype:D = { 15 }
3701                     {
3702                         \__unravel_box_hook:N \tex_everymath:D
3703                         \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3704                         \l__unravel_head_token
3705                         \__unravel_box_hook_end:
3706                         \int_set:Nn \l__unravel_choice_int { 1 }
3707                     }
3708                 { \__unravel_off_save: }
3709             }
3710     }
3711     { \__unravel_forbidden_case: }
3712 }

```

```

3713 \__unravel_new_tex_cmd:nn { left_right } % 49
3714 {
3715   \__unravel_mode_math:n
3716   {
3717     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3718     \__unravel_print_action:
3719     \__unravel_scan_delimiter:
3720     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3721     \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_left:D
3722     {
3723       \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3724       \tl_use:N \l__unravel_head_tl \scan_stop:
3725       \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3726     }
3727     {
3728       \int_case:nnF \tex_currentgrouptype:D
3729       {
3730         { 16 }
3731         {
3732           \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3733           \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
3734           \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_middle:D
3735           {
3736             \tl_use:N \l__unravel_head_tl \scan_stop:
3737             \gtl_clear:N \l__unravel_after_group_gtl
3738           }
3739           { \tl_use:N \l__unravel_head_tl \scan_stop: }
3740           \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3741         }
3742         { 15 }
3743         { % todo: this is a TeX error
3744           \tl_use:N \l__unravel_head_tl \scan_stop:
3745         }
3746       }
3747       { \__unravel_off_save: }
3748     }
3749   }
3750 }
3751 \cs_new_protected:Npn \__unravel_scan_delimiter:
3752 {
3753   \__unravel_get_x_non_relax:
3754   \__unravel_set_cmd:
3755   \int_case:nnF \l__unravel_head_cmd_int
3756   {
3757     { \__unravel_tex_use:n { the_char } }
3758     { \__unravel_prev_input:V \l__unravel_head_tl }
3759     { \__unravel_tex_use:n { other_char } }
3760     { \__unravel_prev_input:V \l__unravel_head_tl }
3761     { \__unravel_tex_use:n { delim_num } }
3762     {
3763       \__unravel_prev_input:V \l__unravel_head_tl
3764       \__unravel_scan_int:
3765     }
3766   }

```

```

3767     {
3768         \__unravel_back_input:
3769         \__unravel_tex_error:nV { missing-delim } \l__unravel_head_tl
3770         \__unravel_prev_input:n { . }
3771     }
3772 }

3773 \__unravel_new_tex_cmd:nn { math_comp } % 50
3774 { \__unravel_mode_math:n { \__unravel_sub_sup: } }

3775 \__unravel_new_tex_cmd:nn { limit_switch } % 51
3776 { \__unravel_mode_math:n { \__unravel_output_head_token: } }
3777 \cs_new_protected:Npn \__unravel_output_head_token:
3778 {
3779     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3780     \l__unravel_head_token
3781     \__unravel_print_action:
3782 }

3783 \__unravel_new_tex_cmd:nn { above } % 52
3784 { \__unravel_mode_math:n { \__unravel_not_implemented:n { above } } }

3785 \__unravel_new_tex_cmd:nn { math_style } % 53
3786 { \__unravel_mode_math:n { \__unravel_output_head_token: } }

3787 \__unravel_new_tex_cmd:nn { math_choice } % 54
3788 { \__unravel_mode_math:n { \__unravel_do_choice: } }
3789 \cs_new_protected:Npn \__unravel_do_choice:
3790 {
3791     \__unravel_prev_input_gpush:N \l__unravel_head_tl
3792     \__unravel_print_action:
3793     \__unravel_scan_left_brace:
3794     \__unravel_prev_input_gpop:N \l__unravel_head_tl
3795     \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3796     \gtl_gconcat:NNN \g__unravel_output_gtl
3797     \g__unravel_output_gtl \c_group_begin_gtl
3798     \tl_use:N \l__unravel_head_tl \c_group_begin_token
3799     \gtl_clear:N \l__unravel_after_group_gtl
3800     \int_set:Nn \l__unravel_choice_int { 1 }
3801     \__unravel_print_action:x
3802     { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
3803 }

3804 \__unravel_new_tex_cmd:nn { non_script } % 55
3805 { \__unravel_mode_math:n { \__unravel_output_head_token: } }

3806 \__unravel_new_tex_cmd:nn { vcenter } % 56
3807 { \__unravel_mode_math:n { \__unravel_not_implemented:n { vcenter } } }

```

2.12.5 From 57 to 70

- case_shift=57
- message=58
- extension=59
- in_stream=60

- begin_group=61
- end_group=62
- omit=63
- ex_space=64
- no_boundary=65
- radical=66
- end_cs_name=67
- char_given=68
- math_given=69
- last_item=70

```

3808 \_unravel_new_tex_cmd:nn { case_shift } % 57
3809 {
3810   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3811   \_unravel_scan_toks:NN \c_false_bool \c_false_bool
3812   \_unravel_prev_input_gpop:N \l__unravel_tmpa_tl
3813   \exp_after:wN \_unravel_case_shift:Nn \l__unravel_tmpa_tl
3814 }
3815 \cs_new_protected:Npn \_unravel_case_shift:Nn #1#2
3816 {
3817   #1 { \_unravel_back_input:n {#2} }
3818   \_unravel_print_action:x
3819   { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
3820 }
3821 \_unravel_new_tex_cmd:nn { message } % 58
3822 {
3823   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3824   \_unravel_print_action:
3825   \_unravel_scan_toks_to_str:
3826   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3827   \tl_use:N \l__unravel_head_tl
3828   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3829 }

```

Extensions are implemented in a later section.

```

3830 \_unravel_new_tex_cmd:nn { extension } % 59
3831 {
3832   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3833   \_unravel_print_action:
3834   \_unravel_scan_extension_operands:
3835   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3836   \tl_use:N \l__unravel_head_tl \scan_stop:
3837   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3838 }

```

```

3839 \_unravel_new_tex_cmd:nn { in_stream } % 60
3840 {
3841   \_unravel_prev_input_gpush:N \l__unravel_head_tl
3842   \_unravel_print_action:
3843   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
3844   {
3845     \_unravel_scan_int:
3846     \_unravel_scan_optional_equals:
3847     \_unravel_scan_file_name:
3848   }
3849   { \_unravel_scan_int: }
3850   \_unravel_prev_input_gpop:N \l__unravel_head_tl
3851   \tl_use:N \l__unravel_head_tl \scan_stop:
3852   \_unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
3853 }

3854 \_unravel_new_tex_cmd:nn { begin_group } % 61
3855 {
3856   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3857   \l__unravel_head_token
3858   \gtl_clear:N \l__unravel_after_group_gtl
3859   \_unravel_print_action:
3860 }

3861 \_unravel_new_tex_cmd:nn { end_group } % 62
3862 {
3863   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
3864   \_unravel_back_input_gtl:N \l__unravel_after_group_gtl
3865   \l__unravel_head_token
3866   \_unravel_print_action:
3867 }

3868 \_unravel_new_tex_cmd:nn { omit } % 63
3869 { \l__unravel_head_token \_unravel_print_action: }

3870 \_unravel_new_tex_cmd:nn { ex_space } % 64
3871 {
3872   \_unravel_mode_non_vertical:n
3873   { \l__unravel_head_token \_unravel_print_action: }
3874 }

3875 \_unravel_new_tex_cmd:nn { no_boundary } % 65
3876 {
3877   \_unravel_mode_non_vertical:n
3878   { \l__unravel_head_token \_unravel_print_action: }
3879 }

3880 \_unravel_new_tex_cmd:nn { radical } % 66
3881 { \_unravel_mode_math:n { \_unravel_do_math_accent: } }

3882 \_unravel_new_tex_cmd:nn { end_cs_name } % 67
3883 {
3884   \_unravel_tex_error:nV { extra-endsname } \l__unravel_head_tl
3885   \_unravel_print_action:
3886 }

See the_char and other_char.

3887 \_unravel_new_tex_cmd:nn { char_given } % 68
3888 {

```

```

3889   \__unravel_mode_non_vertical:n
3890   {
3891     \mode_if_math:TF
3892     { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
3893     { \__unravel_char:V \l__unravel_head_char_int }
3894   }
3895 }

```

See `math_char_num`.

```

3896 \__unravel_new_tex_cmd:nn { math_given } % 69
3897 {
3898   \__unravel_mode_math:n
3899   { \__unravel_mathchar:x { \int_use:N \l__unravel_head_char_int } }
3900 }
3901 \__unravel_new_tex_cmd:nn { last_item } % 70
3902 { \__unravel_forbidden_case: }

```

2.12.6 Extensions

`__unravel_scan_extension_operands:`

```

3903 \cs_new_protected:Npn \__unravel_scan_extension_operands:
3904 {
3905   \int_case:nnF \l__unravel_head_char_int
3906   {
3907     { 0 } % openout
3908     {
3909       \__unravel_scan_int:
3910       \__unravel_scan_optional_equals:
3911       \__unravel_scan_file_name:
3912     }
3913     { 1 } % write
3914     {
3915       \__unravel_scan_int:
3916       \__unravel_scan_toks:NN \c_false_bool \c_false_bool
3917     }
3918     { 2 } % closeout
3919     { \__unravel_scan_int: }
3920     { 3 } % special
3921     { \__unravel_scan_toks_to_str: }
3922     { 4 } % immediate
3923     { \__unravel_scan_immediate_operands: }
3924     { 5 } % setlanguage
3925     {
3926       \mode_if_horizontal:TF
3927       { \__unravel_scan_int: }
3928       { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3929     }
3930     { 6 } % pdfliteral
3931     {
3932       \__unravel_scan_keyword:nF { dDiIrReEcCtT }
3933       { \__unravel_scan_keyword:n { pPaAgGeE } }
3934       \__unravel_scan_pdf_ext_toks:
3935     }
3936     { 7 } % pdfobj

```



```

3937     {
3938         \_unravel_scan_keyword:nTF
3939         { rReEsSeErRvVeEoObBjJnNuUmM }
3940         { \_unravel_skip_optional_space: }
3941         {
3942             \_unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
3943             { \_unravel_scan_int: }
3944             \_unravel_scan_keyword:nT { sStTrReEaAmM }
3945             {
3946                 \_unravel_scan_keyword:nT { aAtTtTrR }
3947                 { \_unravel_scan_pdf_ext_toks: }
3948             }
3949             \_unravel_scan_keyword:n { fFillLeE }
3950             \_unravel_scan_pdf_ext_toks:
3951         }
3952     }
3953 { 8 } % pdfrefobj
3954     { \_unravel_scan_int: }
3955 { 9 } % pdfxform
3956     {
3957         \_unravel_scan_keyword:nT { aAtTtTrR }
3958         { \_unravel_scan_pdf_ext_toks: }
3959         \_unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
3960         { \_unravel_scan_pdf_ext_toks: }
3961         \_unravel_scan_int:
3962     }
3963 { 10 } % pdfrefxform
3964     { \_unravel_scan_int: }
3965 { 11 } % pdfximage
3966     { \_unravel_scan_image: }
3967 { 12 } % pdfrefximage
3968     { \_unravel_scan_int: }
3969 { 13 } % pdfannot
3970     {
3971         \_unravel_scan_keyword:nTF
3972         { rReEsSeErRvVeEoObBjJnNuUmM }
3973         { \_unravel_scan_optional_space: }
3974         {
3975             \_unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
3976             { \_unravel_scan_int: }
3977             \_unravel_scan_alt_rule:
3978             \_unravel_scan_pdf_ext_toks:
3979         }
3980     }
3981 { 14 } % pdfstartlink
3982     {
3983         \mode_if_vertical:TF
3984         { \_unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3985         {
3986             \_unravel_scan_rule_attr:
3987             \_unravel_scan_action:
3988         }
3989     }
3990 { 15 } % pdfendlink

```

```

3991     {
3992         \mode_if_vertical:T
3993         { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
3994     }
3995 { 16 } % pdfoutline
3996     {
3997         \__unravel_scan_keyword:nT { aAtTtTrR }
3998         { \__unravel_scan_pdf_ext_toks: }
3999         \__unravel_scan_action:
4000         \__unravel_scan_keyword:nT { cCoOuUnNtT }
4001         { \__unravel_scan_int: }
4002         \__unravel_scan_pdf_ext_toks:
4003     }
4004 { 17 } % pdfdest
4005     { \__unravel_scan_pdfdest_operands: }
4006 { 18 } % pdfthread
4007     { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
4008 { 19 } % pdfstartthread
4009     { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
4010 { 20 } % pdfendthread
4011     { }
4012 { 21 } % pdfsavepos
4013     { }
4014 { 22 } % pdfinfo
4015     { \__unravel_scan_pdf_ext_toks: }
4016 { 23 } % pdfcatalog
4017     {
4018         \__unravel_scan_pdf_ext_toks:
4019         \__unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
4020         { \__unravel_scan_action: }
4021     }
4022 { 24 } % pdfnames
4023     { \__unravel_scan_pdf_ext_toks: }
4024 { 25 } % pdffontattr
4025     {
4026         \__unravel_scan_font_ident:
4027         \__unravel_scan_pdf_ext_toks:
4028     }
4029 { 26 } % pdfincludechars
4030     {
4031         \__unravel_scan_font_ident:
4032         \__unravel_scan_pdf_ext_toks:
4033     }
4034 { 27 } % pdfmapfile
4035     { \__unravel_scan_pdf_ext_toks: }
4036 { 28 } % pdfmapline
4037     { \__unravel_scan_pdf_ext_toks: }
4038 { 29 } % pdftrailer
4039     { \__unravel_scan_pdf_ext_toks: }
4040 { 30 } % pdfresettimer
4041     { }
4042 { 31 } % pdffontexpand
4043     {
4044         \__unravel_scan_font_ident:

```

```

4045     \_unravel_scan_optional_equals:
4046     \_unravel_scan_int:
4047     \_unravel_scan_int:
4048     \_unravel_scan_int:
4049     \_unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
4050     { \_unravel_skip_optional_space: }
4051   }
4052   { 32 } % pdfsetrandomseed
4053   { \_unravel_scan_int: }
4054   { 33 } % pdfsnaprefpoint
4055   { }
4056   { 34 } % pdfsnapy
4057   { \_unravel_scan_normal_glue: }
4058   { 35 } % pdfsnapycomp
4059   { \_unravel_scan_int: }
4060   { 36 } % pdfglyphtounicode
4061   {
4062     \_unravel_scan_pdf_ext_toks:
4063     \_unravel_scan_pdf_ext_toks:
4064   }
4065   { 37 } % pdfcolorstack
4066   { \_unravel_scan_pdfcolorstack_operands: }
4067   { 38 } % pdfsetmatrix
4068   { \_unravel_scan_pdf_ext_toks: }
4069   { 39 } % pdfsave
4070   { }
4071   { 40 } % pdfrestore
4072   { }
4073   { 41 } % pdfnobluiltintounicode
4074   { \_unravel_scan_font_ident: }
4075   }
4076   { } % no other cases.
4077 }

```

(End definition for _unravel_scan_extension_operands:.)

_unravel_scan_pdfcolorstack_operands:

```

4078 \cs_new_protected:Npn \_unravel_scan_pdfcolorstack_operands:
4079 {
4080   \_unravel_scan_int:
4081   \_unravel_scan_keyword:nF { sSeEtT }
4082   {
4083     \_unravel_scan_keyword:nF { pPuUsShH }
4084     {
4085       \_unravel_scan_keyword:nF { pPoOpP }
4086       {
4087         \_unravel_scan_keyword:nF { cCuUrRrReEnNtT }
4088         {
4089           \_unravel_error:nnnnn { color-stack-action-missing }
4090           { } { } { } { }
4091         }
4092       }
4093     }
4094   }
4095 }

```

(End definition for _unravel_scan_pdfcolorstack_operands:.)

_unravel_scan_rule_attr:

```
4096 \cs_new_protected:Npn \_unravel_scan_rule_attr:
4097 {
4098   \_unravel_scan_alt_rule:
4099   \_unravel_scan_keyword:nT { aAtTtTrR }
4100   { \_unravel_scan_pdf_ext_toks: }
4101 }
```

(End definition for _unravel_scan_rule_attr:.)

_unravel_scan_action:

```
4102 \cs_new_protected:Npn \_unravel_scan_action:
4103 {
4104   \_unravel_scan_keyword:nTF { uUsSeErR }
4105   { \_unravel_scan_pdf_ext_toks: }
4106   {
4107     \_unravel_scan_keyword:nF { gGoOtToO }
4108     {
4109       \_unravel_scan_keyword:nF { tThHrReEaAdD }
4110       { \_unravel_error:nnnnn { action-type-missing } { } { } { } { } }
4111     }
4112   }
4113   \_unravel_scan_keyword:nT { fFiIlLeE }
4114   { \_unravel_scan_pdf_ext_toks: }
4115   \_unravel_scan_keyword:nTF { pPaAgGeE }
4116   {
4117     \_unravel_scan_int:
4118     \_unravel_scan_pdf_ext_toks:
4119   }
4120   {
4121     \_unravel_scan_keyword:nTF { nNaAmMeE }
4122     { \_unravel_scan_pdf_ext_toks: }
4123     {
4124       \_unravel_scan_keyword:nTF { nNuUmM }
4125       { \_unravel_scan_int: }
4126       { \_unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
4127     }
4128   }
4129   \_unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
4130   { \_unravel_skip_optional_space: }
4131   {
4132     \_unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
4133     { \_unravel_skip_optional_space: }
4134   }
4135 }
```

(End definition for _unravel_scan_action:.)

_unravel_scan_image: Used by \pdfximage.

```
4136 \cs_new_protected:Npn \_unravel_scan_image:
4137 {
4138   \_unravel_scan_rule_attr:
```

```

4139 \__unravel_scan_keyword:nTF { nNaAmMeEdD }
4140   { \__unravel_scan_pdf_ext_toks: }
4141   {
4142     \__unravel_scan_keyword:nT { pPaAgGeE }
4143     { \__unravel_scan_int: }
4144   }
4145 \__unravel_scan_keyword:nT { cCo0lLo0rRsSpPaAcCeE }
4146   { \__unravel_scan_int: }
4147 \__unravel_scan_pdf_ext_toks:
4148 }

```

(End definition for __unravel_scan_image:.)

__unravel_scan_immediate_operands:

```

4149 \cs_new_protected:Npn \__unravel_scan_immediate_operands:
4150 {
4151   \__unravel_get_x_next:
4152   \__unravel_set_cmd:
4153   \int_compare:nNnTF
4154     \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
4155   {
4156     \int_compare:nNnTF
4157       \l__unravel_head_char_int < { 3 } % openout, write, closeout
4158       { \__unravel_scan_immediate_operands_aux: }
4159     {
4160       \int_case:nnF \l__unravel_head_char_int
4161       {
4162         { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
4163         { 9 }
4164         {
4165           \__unravel_prepare_mag:
4166           \__unravel_scan_extension_operands_aux:
4167           } % pdfxform
4168         { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
4169       }
4170       { \__unravel_scan_immediate_operands_bad: }
4171     }
4172   }
4173   { \__unravel_scan_immediate_operands_bad: }
4174 }
4175 \cs_new_protected:Npn \__unravel_scan_immediate_operands_aux:
4176 {
4177   \__unravel_prev_input:V \l__unravel_head_tl
4178   \__unravel_scan_extension_operands:
4179 }
4180 \cs_new_protected:Npn \__unravel_scan_immediate_operands_bad:
4181 {
4182   \__unravel_back_input:
4183   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4184   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl ignored }
4185   \__unravel_prev_input_gpush:
4186 }
4187

```

(End definition for __unravel_scan_immediate_operands:.)

_unravel_scan_pdfdest_operands:

```
4188 \cs_new_protected:Npn \_unravel_scan_pdfdest_operands:
4189 {
4190   \_unravel_scan_keyword:nTF { nNuUmM }
4191     { \_unravel_scan_int: }
4192     {
4193       \_unravel_scan_keyword:nTF { nNaAmMeE }
4194         { \_unravel_scan_pdf_ext_toks: }
4195         { \_unravel_error:nmnnn { identifier-type-missing } { } { } { } { } }
4196     }
4197   \_unravel_scan_keyword:nTF { xXyYzZ }
4198     {
4199       \_unravel_scan_keyword:nT { zZoOoOmM }
4200       { \_unravel_scan_int: }
4201     }
4202     {
4203       \_unravel_scan_keyword:nF { fFiItTbBhH }
4204       {
4205         \_unravel_scan_keyword:nF { fFiItTbBvV }
4206         {
4207           \_unravel_scan_keyword:nF { fFiItTbB }
4208           {
4209             \_unravel_scan_keyword:nF { fFiItThHhH }
4210             {
4211               \_unravel_scan_keyword:nF { fFiItTvV }
4212               {
4213                 \_unravel_scan_keyword:nTF
4214                   { fFiItTrR }
4215                   {
4216                     \_unravel_skip_optional_space:
4217                     \_unravel_scan_alt_rule:
4218                     \use_none:n
4219                   }
4220                   {
4221                     \_unravel_scan_keyword:nF
4222                       { fFiItT }
4223                       {
4224                         \_unravel_error:nmnnn { destination-type-missing }
4225                         { } { } { } { }
4226                       }
4227                   }
4228               }
4229             }
4230           }
4231         }
4232       }
4233     }
4234   \_unravel_skip_optional_space:
4235 }
```

(End definition for _unravel_scan_pdfdest_operands:.)

2.12.7 Assignments

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

```

4236 \cs_set_protected:Npn \__unravel_tmp:w
4237 {
4238   \__unravel_prev_input_gpush:
4239   \__unravel_prefixed_command:
4240 }
4241 \int_step_inline:nnnn
4242 { \__unravel_tex_use:n { max_non_prefixed_command } + 1 }
4243 { 1 }
4244 { \__unravel_tex_use:n { max_command } }
4245 { \cs_new_eq:cN { __unravel_cmd_#1: } \__unravel_tmp:w }

```

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

```

4246 \cs_new_protected:Npn \__unravel_prefixed_command:
4247 {
4248   \int_while_do:nNnn
4249   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
4250   {
4251     \__unravel_prev_input:V \l__unravel_head_tl
4252     \__unravel_get_x_non_relax:
4253     \__unravel_set_cmd:
4254     \int_compare:nNnF \l__unravel_head_cmd_int
4255     > { \__unravel_tex_use:n { max_non_prefixed_command } }
4256     {
4257       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4258       \__unravel_error:nxxxx { erroneous-prefixes }
4259       { \tl_to_str:N \l__unravel_tmpa_tl }
4260       { \tl_to_str:N \l__unravel_head_tl }
4261       { } { }
4262       \__unravel_back_input:
4263       \__unravel_omit_after_assignment:w
4264     }
4265   }
4266   % ^^A todo: Discard non-\global prefixes if they are irrelevant
4267   % ^^A todo: Adjust for the setting of \globaldefs
4268   \cs_if_exist_use:cF
4269   { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
4270   {
4271     \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
4272     \__unravel_omit_after_assignment:w
4273   }
4274   \__unravel_after_assignment:
4275 }

```

(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
4276 \cs_new_protected:Npn \__unravel_after_assignment:
4277 {
4278   \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
4279   \gtl_gclear:N \g__unravel_after_assignment_gtl
4280 }
4281 \cs_new_protected:Npn \__unravel_omit_after_assignment:w
4282   #1 \__unravel_after_assignment: { }

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

```

```

\__unravel_prefixed_new:nn
4283 \cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
4284 {
4285   \cs_new_protected:cpn
4286     { \__unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
4287 }

(End definition for \__unravel_prefixed_new:nn.)

```

```

\__unravel_assign_token:n
4288 \cs_new_protected:Npn \__unravel_assign_token:n #1
4289 {
4290   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4291   #1
4292   \tl_use:N \l__unravel_head_tl \scan_stop:
4293   \__unravel_print_assigned_token:
4294 }

(End definition for \__unravel_assign_token:n.)

```

```

\__unravel_assign_register:
4295 \cs_new_protected:Npn \__unravel_assign_register:
4296 {
4297   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4298   \tl_use:N \l__unravel_head_tl \scan_stop:
4299   \__unravel_print_assigned_register:
4300 }

(End definition for \__unravel_assign_register:.)

```

```

\__unravel_assign_value:nn
4301 \cs_new_protected:Npn \__unravel_assign_value:nn #1#2
4302 {
4303   \tl_if_empty:nF {#1}
4304   {
4305     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4306     \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4307     #1
4308     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4309   }
4310   \__unravel_prev_input:V \l__unravel_head_tl
4311   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4312   \__unravel_scan_optional_equals:
4313   #2

```



```

4314   \_unravel_assign_register:
4315   }

```

(End definition for _unravel_assign_value:nn.)

_unravel_assign_toks:

```

4316 \_unravel_prefixed_new:nn { toks_register }           % 71
4317 {
4318   \int_compare:nNnT \l__unravel_head_char_int = 0
4319   { % \toks
4320     \_unravel_prev_input_gpush:N \l__unravel_head_tl
4321     \_unravel_print_action:
4322     \_unravel_scan_int:
4323     \_unravel_prev_input_gpop:N \l__unravel_head_tl
4324   }
4325   \_unravel_assign_toks:
4326 }
4327 \_unravel_prefixed_new:nn { assign_toks }             % 72
4328 { \_unravel_assign_toks: }
4329 \cs_new_protected:Npn \_unravel_assign_toks:
4330 {
4331   \_unravel_prev_input_silent:V \l__unravel_head_tl
4332   \_unravel_print_action:
4333   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4334   \_unravel_scan_optional_equals:
4335   \_unravel_get_x_non_relax:
4336   \_unravel_set_cmd:
4337   \int_compare:nNnTF
4338     \l__unravel_head_cmd_int = { \_unravel_tex_use:n { toks_register } }
4339     {
4340       \_unravel_prev_input:V \l__unravel_head_tl
4341       \int_compare:nNnT \l__unravel_head_char_int = 0
4342       { \_unravel_scan_int: }
4343     }
4344     {
4345       \int_compare:nNnTF
4346         \l__unravel_head_cmd_int = { \_unravel_tex_use:n { assign_toks } }
4347         { \_unravel_prev_input:V \l__unravel_head_tl }
4348         {
4349           \_unravel_back_input:
4350           \_unravel_scan_toks:NN \c_false_bool \c_false_bool
4351         }
4352     }
4353   \_unravel_assign_register:
4354 }

```

(End definition for _unravel_assign_toks:.)

```

4355 \_unravel_prefixed_new:nn { assign_int }             % 73
4356 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4357 \_unravel_prefixed_new:nn { assign_dimen }           % 74
4358 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4359 \_unravel_prefixed_new:nn { assign_glue }           % 75
4360 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_glue: } }
4361 \_unravel_prefixed_new:nn { assign_mu_glue }        % 76

```

```

4362 { \_unravel_assign_value:nn { } { \_unravel_scan_mu_glue: } }
4363 \_unravel_prefixed_new:nn { assign_font_dimen } % 77
4364 {
4365   \_unravel_assign_value:nn
4366   { \_unravel_scan_int: \_unravel_scan_font_ident: }
4367   { \_unravel_scan_normal_dimen: }
4368 }
4369 \_unravel_prefixed_new:nn { assign_font_int } % 78
4370 {
4371   \_unravel_assign_value:nn
4372   { \_unravel_scan_font_int: } { \_unravel_scan_int: }
4373 }
4374 \_unravel_prefixed_new:nn { set_aux } % 79
4375 { % prevdepth = 1, spacefactor = 102
4376   \int_compare:nNnTF \l_unravel_head_char_int = 1
4377   { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4378   { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4379 }
4380 \_unravel_prefixed_new:nn { set_prev_graf } % 80
4381 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4382 \_unravel_prefixed_new:nn { set_page_dimen } % 81
4383 { \_unravel_assign_value:nn { } { \_unravel_scan_normal_dimen: } }
4384 \_unravel_prefixed_new:nn { set_page_int } % 82
4385 { \_unravel_assign_value:nn { } { \_unravel_scan_int: } }
4386 \_unravel_prefixed_new:nn { set_box_dimen } % 83
4387 {
4388   \_unravel_assign_value:nn
4389   { \_unravel_scan_int: } { \_unravel_scan_normal_dimen: }
4390 }
4391 \_unravel_prefixed_new:nn { set_shape } % 84
4392 {
4393   \_unravel_assign_value:nn { \_unravel_scan_int: }
4394   {
4395     \prg_replicate:nn
4396     {
4397       \tl_if_head_eq_meaning:VNT
4398       \l_unravel_defined_tl \tex_parshape:D { 2 * }
4399       \tl_tail:N \l_unravel_defined_tl
4400     }
4401     { \_unravel_scan_int: }
4402   }
4403 }
4404 \_unravel_prefixed_new:nn { def_code } % 85
4405 {
4406   \_unravel_assign_value:nn
4407   { \_unravel_scan_int: } { \_unravel_scan_int: }
4408 }
4409 \_unravel_prefixed_new:nn { def_family } % 86
4410 {
4411   \_unravel_assign_value:nn
4412   { \_unravel_scan_int: } { \_unravel_scan_font_ident: }
4413 }
4414 \_unravel_prefixed_new:nn { set_font } % 87

```

```

4415 {
4416   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4417   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4418   \tl_use:N \l__unravel_head_tl \scan_stop:
4419   \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
4420   \__unravel_print_action:
4421 }
4422 \__unravel_prefixed_new:nn { def_font } % 88
4423 {
4424   \__unravel_prev_input_silent:V \l__unravel_head_tl
4425   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4426   \__unravel_scan_r_token:
4427   \__unravel_print_action:x
4428     { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4429   \__unravel_scan_optional_equals:
4430   \__unravel_scan_file_name:
4431   \bool_gset_true:N \g__unravel_name_in_progress_bool
4432   \__unravel_scan_keyword:nTF { aAtT }
4433     { \__unravel_scan_normal_dimen: }
4434     {
4435       \__unravel_scan_keyword:nT { sScCaAlLeEdD }
4436       { \__unravel_scan_int: }
4437     }
4438   \bool_gset_false:N \g__unravel_name_in_progress_bool
4439   \__unravel_assign_token:n { }
4440 }

```

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

let, futurelet

```

4441 \__unravel_prefixed_new:nn { let } % 94
4442 {
4443   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4444   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
4445   { % |let|
4446     \__unravel_scan_r_token:
4447     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4448     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4449     \__unravel_get_next:
4450     \bool_while_do:nn
4451       { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
4452       { \__unravel_get_next: }
4453     \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
4454     { \__unravel_get_next: }
4455     \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
4456     { \__unravel_get_next: }
4457   }
4458   { % |futurelet|
4459     \__unravel_scan_r_token:
4460     \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4461     \__unravel_print_action:x { \tl_to_str:N \l__unravel_tmpa_tl }
4462     \__unravel_get_next:
4463     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4464     \__unravel_get_next:

```

```

4465     \__unravel_back_input:
4466     \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
4467     \__unravel_back_input:
4468   }
4469   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4470   \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
4471   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4472   \__unravel_exp_args:Nx \use:n
4473   {
4474     \exp_not:V \l__unravel_head_tl
4475     \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
4476   }
4477   \__unravel_print_assigned_token:
4478 }
4479 \__unravel_prefixed_new:nm { shorthand_def } % 95
4480 {
4481   \__unravel_prev_input_silent:V \l__unravel_head_tl
4482   \tl_set:Nx \l__unravel_prev_action_tl
4483     { \tl_to_str:N \l__unravel_head_tl }
4484   \__unravel_scan_r_token:
4485   \__unravel_print_action:x
4486     { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
4487   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
4488   \__unravel_just_print_assigned_token:
4489   \__unravel_scan_optional_equals:
4490   \__unravel_scan_int:
4491   \__unravel_assign_token:n { }
4492 }

```

__unravel_read_to_cs_safe:nTF
__unravel_read_to_cs_safe:fTF

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

```

4493 \__unravel_prefixed_new:nm { read_to_cs } % 96
4494 {
4495   \__unravel_prev_input_silent:V \l__unravel_head_tl
4496   \__unravel_print_action:x { \tl_to_str:N \l__unravel_head_tl }
4497   \__unravel_scan_int:
4498   \__unravel_scan_to:
4499   \__unravel_scan_r_token:
4500   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4501   \__unravel_read_to_cs_safe:fTF
4502     { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
4503     { \__unravel_assign_token:n { } }
4504   {
4505     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4506     \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
4507   }
4508 }
4509 \prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
4510 {

```

```

4511 \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
4512 { \prg_return_true: }
4513 {
4514   \int_compare:nNnTF {#1} < { 0 }
4515   { \prg_return_false: }
4516   {
4517     \int_compare:nNnTF {#1} > { 15 }
4518     { \prg_return_false: }
4519     {
4520       \tex_ifeof:D #1 \exp_stop_f:
4521       \prg_return_false:
4522       \else:
4523       \prg_return_true:
4524       \fi:
4525     }
4526   }
4527 }
4528 }
4529 \cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }

(End definition for \__unravel_read_to_cs_safe:nTF.)

4530 \__unravel_prefixed_new:nn { def } % 97
4531 {
4532   \__unravel_prev_input_get:N \l__unravel_tmpa_tl
4533   \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
4534   \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
4535   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4536   \int_compare:nNnTF \l__unravel_head_char_int < 2
4537   { % def/gdef
4538     \__unravel_scan_r_token:
4539     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4540     \__unravel_scan_toks:NN \c_true_bool \c_false_bool
4541   }
4542   { % edef/xdef
4543     \__unravel_scan_r_token:
4544     \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
4545     \__unravel_scan_toks:NN \c_true_bool \c_true_bool
4546   }
4547   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4548   \__unravel_prev_input:V \l__unravel_head_tl
4549   \__unravel_assign_token:n
4550   { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
4551 }

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

4552 \__unravel_prefixed_new:nn { set_box } % 98
4553 {
4554   \__unravel_prev_input:V \l__unravel_head_tl
4555   \__unravel_scan_int:
4556   \__unravel_scan_optional_equals:
4557   \bool_if:NTF \g__unravel_set_box_allowed_bool
4558   { \__unravel_do_box:N \c_false_bool }

```

```

4559     {
4560       \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
4561       \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4562       \__unravel_omit_after_assignment:w
4563     }
4564   }
  \hyphenation and \patterns
4565 \__unravel_prefixed_new:nn { hyph_data } % 99
4566 {
4567   \__unravel_prev_input:V \l__unravel_head_tl
4568   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4569   \__unravel_assign_token:n { }
4570 }
4571 \__unravel_prefixed_new:nn { set_interaction } % 100
4572 {
4573   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4574   \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
4575   \tl_use:N \l__unravel_head_tl \scan_stop:
4576   \__unravel_print_assignment:x { \tl_to_str:N \l__unravel_head_tl }
4577 }
4578 \__unravel_prefixed_new:nn { letterspace_font } % 101
4579 {
4580   \__unravel_prev_input_silent:V \l__unravel_head_tl
4581   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4582   \__unravel_scan_r_token:
4583   \__unravel_print_action:x
4584   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4585   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4586   \__unravel_just_print_assigned_token:
4587   \__unravel_scan_optional_equals:
4588   \__unravel_scan_font_ident:
4589   \__unravel_scan_int:
4590   \__unravel_assign_token:n { }
4591 }
4592 \__unravel_prefixed_new:nn { pdf_copy_font } % 102
4593 {
4594   \__unravel_prev_input_silent:V \l__unravel_head_tl
4595   \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
4596   \__unravel_scan_r_token:
4597   \__unravel_print_action:x
4598   { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
4599   \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
4600   \__unravel_just_print_assigned_token:
4601   \__unravel_scan_optional_equals:
4602   \__unravel_scan_font_ident:
4603   \__unravel_assign_token:n { }
4604 }
  Changes to numeric registers (\count, \dimen, \skip, \muskip, and commands with
  a built-in number).
4605 \__unravel_prefixed_new:nn { register } % 89
4606 { \__unravel_do_register:N 0 }

```

```

4607 \__unravel_prefixed_new:nm { advance } % 90
4608 { \__unravel_do_operation:N 1 }
4609 \__unravel_prefixed_new:nm { multiply } % 91
4610 { \__unravel_do_operation:N 2 }
4611 \__unravel_prefixed_new:nm { divide } % 92
4612 { \__unravel_do_operation:N 3 }

```

__unravel_do_operation:N

__unravel_do_operation_fail:w

```

4613 \cs_new_protected:Npn \__unravel_do_operation:N #1
4614 {
4615   \__unravel_prev_input_silent:V \l__unravel_head_tl
4616   \__unravel_print_action:
4617   \__unravel_get_x_next:
4618   \__unravel_set_cmd:
4619   \int_compare:nNnTF
4620     \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
4621     {
4622       \int_compare:nNnTF
4623         \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
4624         { \__unravel_do_register:N #1 }
4625         { \__unravel_do_operation_fail:w }
4626     }
4627     {
4628       \int_compare:nNnTF
4629         \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
4630         { \__unravel_do_operation_fail:w }
4631         {
4632           \__unravel_prev_input:V \l__unravel_head_tl
4633           \exp_args:NNf \__unravel_do_register_set:Nn #1
4634             {
4635               \int_eval:n
4636                 {
4637                   \l__unravel_head_cmd_int
4638                   - \__unravel_tex_use:n { assign_toks }
4639                 }
4640             }
4641         }
4642     }
4643 }
4644 \cs_new_protected:Npn \__unravel_do_operation_fail:w
4645 {
4646   \__unravel_error:nnnnn { after-advance } { } { } { } { }
4647   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4648   \__unravel_omit_after_assignment:w
4649 }

```

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

__unravel_do_register:N

__unravel_do_register_aux:Nn

```

4650 \cs_new_protected:Npn \__unravel_do_register:N #1
4651 {
4652   \exp_args:NNV \__unravel_do_register_aux:Nn #1
4653   \l__unravel_head_char_int
4654 }

```

```

4655 \cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
4656 {
4657   \int_compare:nNnTF { \tl_tail:n {#2} } = 0
4658   {
4659     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4660     \__unravel_print_assignment:
4661     \__unravel_scan_int:
4662     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4663     \__unravel_prev_input_silent:V \l__unravel_head_tl
4664   }
4665   {
4666     \__unravel_prev_input_silent:V \l__unravel_head_tl
4667     \__unravel_print_assignment:
4668   }
4669   \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
4670   \exp_args:Nnf \__unravel_do_register_set:Nn #1
4671   { \int_eval:n { #2 / 1 000 000 } }
4672 }

```

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

__unravel_do_register_set:Nn

```

4673 \cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
4674 {
4675   \int_compare:nNnTF {#1} = 0
4676   { % truly register command
4677     \__unravel_scan_optional_equals:
4678   }
4679   { % \advance, \multiply, \divide
4680     \__unravel_scan_keyword:nF { bByY }
4681     { \__unravel_prev_input_silent:n { by } }
4682   }
4683   \int_compare:nNnTF {#1} < 2
4684   {
4685     \int_case:nnF {#2}
4686     {
4687       { 1 } { \__unravel_scan_int:          } % count
4688       { 2 } { \__unravel_scan_normal_dimen: } % dim
4689       { 3 } { \__unravel_scan_normal_glue:  } % glue
4690       { 4 } { \__unravel_scan_mu_glue:     } % muglue
4691     }
4692     { \__unravel_error:nxxxx { internal } { do-reg=#2 } { } { } { } }
4693   }
4694   { \__unravel_scan_int: }
4695   \__unravel_assign_register:
4696 }

```

(End definition for __unravel_do_register_set:Nn.)

The following is used for instance when making accents.

```

4697 \cs_new_protected:Npn \__unravel_do_assignments:
4698 {
4699   \__unravel_get_x_non_relax:
4700   \__unravel_set_cmd:
4701   \int_compare:nNnT

```



```

4702 \l__unravel_head_cmd_int
4703 > { \__unravel_tex_use:n { max_non_prefixed_command } }
4704 {
4705   \bool_gset_false:N \g__unravel_set_box_allowed_bool
4706   \__unravel_prev_input_gpush:
4707   \__unravel_prefixed_command:
4708   \bool_gset_true:N \g__unravel_set_box_allowed_bool
4709   \__unravel_do_assignments:
4710 }
4711 }

```

2.13 Expandable primitives

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

- `undefined_cs=103` for undefined control sequences (not quite a primitive).
- `expand_after=104` for `\expandafter` and `\unless`.
- `no_expand=105` for `\noexpand` and `\pdfprimitive`.
- `input=106` for `\input`, `\endinput` and `\scantokens`.
- `if_test=107` for the conditionals, `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\if[vhm]mode`, `\if[tydm]dir`, `\ifinner`, `\ifvoid`, `\if[hvtydm]box`, `\ifx`, `\ifeof`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifprimitive`, `\ifabsnum`, `\ifabsdim`, `\ifjfont`, `\iftfont`.
- `fi_or_else=108` for `\fi`, `\else` and `\or`.
- `cs_name=109` for `\csname` and `\lastnamedcs`.
- `convert=110` for `\number`, `\romannumeral`, `\string`, `\meaning`, `\fontname`, `\eTeXrevision`, `\pdftexrevision`, `\pdftexbanner`, `\pdffontname`, `\pdffontobjnum`, `\pdffontsize`, `\pdfpageref`, `\pdfxformname`, `\pdfescapestring`, `\pdfescapename`, `\leftmarginkern`, `\rightmarginkern`, `\pdfstrcmp`, `\pdfcolorstackinit`, `\pdfescapehex`, `\pdfunescapehex`, `\pdfcreationdate`, `\pdffilemoddate`, `\pdffilesize`, `\pdfmdfivesum`, `\pdffiledump`, `\pdfmatch`, `\pdflastmatch`, `\pdfuniformdeviate`, `\pdfnormaldeviate`, `\pdfinsertht`, `\pdfximagebbox`, `\jobname`, `\expanded`, and in Lua \TeX `\directlua`, `\luaescapestring`, and in X \TeX `\Ucharcat`.
- `the=111` for `\the`, `\unexpanded`, and `\detokenize`.
- `top_bot_mark=112` `\topmark`, `\firstmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`, `\topmarks`, `\firstmarks`, `\botmarks`, `\splitfirstmarks`, and `\splitbotmarks`.
- `call=113` for macro calls, implemented by `__unravel_macro_call:.`
- `end_template=117` for \TeX 's end template.

Let \TeX trigger an error.

```

4712 \__unravel_new_tex_expandable:nm { undefined_cs } % 103
4713 { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }

```

```

\__unravel_expandafter:
  \__unravel_unless: 4714 \__unravel_new_tex_expandable:nn { expand_after } % 104
\__unravel_unless_bad: 4715 {
4716   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
4717   { \__unravel_expandafter: } { \__unravel_unless: }
4718 }
4719 \cs_new_protected:Npn \__unravel_expandafter:
4720 {
4721   \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
4722   \__unravel_get_next:
4723   \gtl_concat:NNN \l__unravel_head_gtl
4724   \l__unravel_tmpb_gtl \l__unravel_head_gtl
4725   \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
4726   \__unravel_print_expansion:x { \gtl_to_str:N \l__unravel_head_gtl }
4727   \__unravel_get_next:
4728   \__unravel_token_if_expandable:NTF \l__unravel_head_token
4729   { \__unravel_expand_do:N \prg_do_nothing: }
4730   { \__unravel_back_input: }
4731   \__unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
4732   \__unravel_set_action_text:x
4733   { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
4734   \gtl_pop_left:N \l__unravel_head_gtl
4735   \__unravel_back_input:
4736   \__unravel_print_expansion:
4737 }
4738 \cs_new_protected:Npn \__unravel_unless:
4739 {
4740   \__unravel_get_token:
4741   \int_compare:nNnTF
4742   \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
4743   {
4744     \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
4745     { \__unravel_unless_bad: }
4746     {
4747       \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
4748       % \int_add:Nn \l__unravel_head_char_int { 32 }
4749       \__unravel_expand_nonmacro:
4750     }
4751   }
4752   { \__unravel_unless_bad: }
4753 }
4754 \cs_new_protected:Npn \__unravel_unless_bad:
4755 {
4756   \__unravel_error:nnnnn { bad-unless } { } { } { } { }
4757   \__unravel_back_input:
4758 }

```

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

__unravel_noexpand:N Currently not fully implemented.

__unravel_noexpand_after: The argument of __unravel_noexpand:N is \prg_do_nothing: when \noexpand is hit by \expandafter; otherwise it is one of various loop commands (__unravel_get_x_next:, __unravel_get_x_or_protected:, __unravel_get_token_xdef:, __unravel_get_token_x:) that would call __unravel_get_next: and possibly expand the token

more. For these cases we simply stop after `__unravel_get_next:` and if the token is expandable we pretend its meaning is `\relax`.

The case of `\expandafter` (so `\prg_do_nothing:`) is tougher. Do nothing if the next token is an explicit non-active character (begin-group and end-group characters are detected by `\l__unravel_head_tl`, the rest by testing if the token is definable). Otherwise the token must be marked with `\notexpanded:` (even if the token is currently a non-expandable primitive, as its meaning can be changed by the code skipped over by `\expandafter`). That `\notexpanded:` marker should be removed if the token is taken as the argument of a macro, but we fail to do that. We set the `\notexpanded: . . .` command to be a special `\relax` marker to make it quickly recognizable in `__unravel_get_next:`. This is incidentally the same meaning used by `TEX` for expandable commands.

```

4759 \__unravel_new_tex_expandable:nm { no_expand } % 105
4760 {
4761   \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
4762   { \__unravel_noexpand:N }
4763   { \__unravel_pdfprimitive: }
4764 }
4765 \cs_new_protected:Npn \__unravel_noexpand:N #1
4766 {
4767   \__unravel_get_token:
4768   \cs_if_eq:NNTF #1 \prg_do_nothing:
4769   {
4770     \tl_if_empty:NTF \l__unravel_head_tl
4771     { \__unravel_back_input: }
4772     {
4773       \exp_after:wN \__unravel_token_if_definable:NTF \l__unravel_head_tl
4774       { \__unravel_noexpand_after: }
4775       { \__unravel_back_input: }
4776     }
4777   }
4778   {
4779     \__unravel_back_input:
4780     \__unravel_get_next:
4781     \__unravel_token_if_expandable:NT \l__unravel_head_token
4782     { \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax: }
4783   }
4784 }
4785 \cs_new_protected:Npn \__unravel_noexpand_after:
4786 {
4787   \group_begin:
4788   \__unravel_set_escapechar:n { 92 }
4789   \exp_args:NNc
4790   \group_end:
4791   \__unravel_noexpand_after:N
4792   { notexpanded: \exp_after:wN \token_to_str:N \l__unravel_head_tl }
4793 }
4794 \cs_new_protected:Npn \__unravel_noexpand_after:N #1
4795 {
4796   \cs_gset_eq:NN #1 \__unravel_special_relax:
4797   \__unravel_back_input:n {#1}
4798 }
4799 \cs_new_protected:Npn \__unravel_pdfprimitive:
4800 { \__unravel_not_implemented:n { pdfprimitive } }

```

(End definition for `__unravel_noexpand:N`, `__unravel_noexpand_after:`, and `__unravel_pdfprimitive:.`)

```

\__unravel_endinput:
\__unravel_scantokens: 4801 \__unravel_new_tex_expandable:nn { input } % 106
\__unravel_input:      4802 {
4803   \int_case:nnF \l__unravel_head_char_int
4804   {
4805     { 1 } { \__unravel_endinput: } % \endinput
4806     { 2 } { \__unravel_scantokens: } % \scantokens
4807   }
4808   { % 0=\input
4809     \bool_if:NTF \g__unravel_name_in_progress_bool
4810     { \__unravel_insert_relax: } { \__unravel_input: }
4811   }
4812 }
4813 \cs_new_protected:Npn \__unravel_endinput:
4814 {
4815   \group_begin:
4816   \msg_warning:nn { unravel } { endinput-ignored }
4817   \group_end:
4818   \__unravel_print_expansion:
4819 }
4820 \cs_new_protected:Npn \__unravel_scantokens:
4821 {
4822   \__unravel_prev_input_gpush:
4823   \__unravel_scan_toks:NN \c_false_bool \c_false_bool
4824   \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
4825   \tl_set_rescan:Nno \l__unravel_head_tl { } \l__unravel_tmpa_tl
4826   \__unravel_back_input:V \l__unravel_head_tl
4827   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_tmpa_tl }
4828 }
4829 \cs_new_protected:Npn \__unravel_input:
4830 {
4831   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4832   \__unravel_scan_file_name:
4833   \__unravel_prev_input_gpop:N \l__unravel_head_tl
4834   \tl_set:Nx \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
4835   \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
4836   \__unravel_back_input:V \l__unravel_tmpa_tl
4837   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
4838 }

```

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

```

\__unravel_csname_loop:
4839 \__unravel_new_tex_expandable:nn { cs_name } % 109
4840 {
4841   \int_compare:nNnTF \l__unravel_head_char_int = 0
4842   {
4843     \__unravel_prev_input_gpush:N \l__unravel_head_tl
4844     \__unravel_print_expansion:
4845     \__unravel_csname_loop:
4846     \__unravel_prev_input_silent:V \l__unravel_head_tl
4847     \__unravel_get_lastnamedcs:

```

```

4848     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4849     \__unravel_back_input_tl_o:
4850   }
4851   {
4852     \__unravel_back_input:V \g__unravel_lastnamedcs_tl
4853     \__unravel_print_expansion:x
4854     { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \g__unravel_lastnamedcs_tl }
4855   }
4856 }
4857 \cs_new_protected:Npn \__unravel_csname_loop:
4858 {
4859   \__unravel_get_x_next:
4860   \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
4861   {
4862     \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
4863     {
4864       \__unravel_back_input:
4865       \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
4866       \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
4867     }
4868   }
4869   {
4870     \__unravel_prev_input_silent:x
4871     { \__unravel_token_to_char:N \l__unravel_head_token }
4872     \__unravel_csname_loop:
4873   }
4874 }
4875 \cs_new_protected:Npn \__unravel_get_lastnamedcs:
4876 {
4877   \group_begin:
4878   \__unravel_prev_input_get:N \l__unravel_head_tl
4879   \tl_gset:No \g__unravel_lastnamedcs_tl
4880   { \cs:w \exp_after:wN \use_none:n \l__unravel_head_tl }
4881   \group_end:
4882 }

```

(End definition for __unravel_csname_loop:.)

```

4883 \__unravel_new_tex_expandable:nn { convert } % 110
4884 {
4885   \__unravel_prev_input_gpush:N \l__unravel_head_tl
4886   \__unravel_print_expansion:
4887   \int_case:nn \l__unravel_head_char_int
4888   {
4889     0     \__unravel_scan_int:
4890     1     \__unravel_scan_int:
4891     2     \__unravel_convert_string:
4892     3     \__unravel_convert_meaning:w
4893     4     \__unravel_scan_font_ident:
4894     8     \__unravel_scan_font_ident:
4895     9     \__unravel_scan_font_ident:
4896     { 10 } \__unravel_scan_font_ident:
4897     { 11 } \__unravel_scan_int:
4898     { 12 } \__unravel_scan_int:
4899     { 13 } \__unravel_scan_pdf_ext_toks:

```

```

4900     { 14 } \_unravel_scan_pdf_ext_toks:
4901     { 15 } \_unravel_scan_int:
4902     { 16 } \_unravel_scan_int:
4903     { 17 } \_unravel_scan_pdfstrcmp:
4904     { 18 } \_unravel_scan_pdfcolorstackinit:
4905     { 19 } \_unravel_scan_pdf_ext_toks:
4906     { 20 } \_unravel_scan_pdf_ext_toks:
4907     { 22 } \_unravel_scan_pdf_ext_toks:
4908     { 23 } \_unravel_scan_pdf_ext_toks:
4909     { 24 }
4910     {
4911         \_unravel_scan_keyword:n { fFillLeE }
4912         \_unravel_scan_pdf_ext_toks:
4913     }
4914     { 25 } \_unravel_scan_pdffiledump:
4915     { 26 } \_unravel_scan_pdfmatch:
4916     { 27 } \_unravel_scan_int:
4917     { 28 } \_unravel_scan_int:
4918     { 30 } \_unravel_scan_int:
4919     { 31 } \_unravel_scan_pdfximagebbox:
4920     { 33 } \_unravel_scan_directlua:
4921     { 34 } \_unravel_scan_pdf_ext_toks:
4922     { 35 } \_unravel_scan_pdf_ext_toks:
4923     { 40 }
4924     {
4925         \_unravel_scan_int:
4926         \_unravel_prev_input_silent:n { ~ }
4927         \_unravel_scan_int:
4928     }
4929 }
4930 \_unravel_prev_input_gpop:N \l_unravel_head_tl
4931 \_unravel_back_input_tl_o:
4932 }
4933 \cs_new_protected:Npn \_unravel_convert_string:
4934 {
4935     \_unravel_get_next:
4936     \tl_if_empty:NTF \l_unravel_head_tl
4937     { \_unravel_prev_input:x { \gtl_to_str:N \l_unravel_head_gtl } }
4938     { \_unravel_prev_input:V \l_unravel_head_tl }
4939 }
4940 \cs_new_protected:Npn \_unravel_convert_meaning:w
4941 \_unravel_prev_input_gpop:N \l_unravel_head_tl \_unravel_back_input_tl_o:
4942 {
4943     \_unravel_get_next:
4944     \tl_if_empty:NTF \l_unravel_head_tl
4945     {
4946         \gtl_set_eq:NN \l_unravel_tmpb_gtl \l_unravel_head_gtl
4947         \_unravel_prev_input_gpop:N \l_unravel_prev_input_tl
4948         \exp_args:NNV \gtl_put_left:Nn \l_unravel_tmpb_gtl \l_unravel_prev_input_tl
4949         \_unravel_prev_input_gpush_gtl:N \l_unravel_tmpb_gtl
4950         \_unravel_print_action:x { \gtl_to_str:N \l_unravel_tmpb_gtl }
4951         \_unravel_prev_input_gpop_gtl:N \l_unravel_tmpb_gtl
4952         \tl_set:Nx \l_unravel_tmpa_tl { \gtl_head_do:NN \l_unravel_head_gtl \tex_meaning:D
4953         \_unravel_back_input:V \l_unravel_tmpa_tl

```

```

4954     \__unravel_print_expansion:x
4955     { \gtl_to_str:N \l__unravel_tmpb_gtl = \tl_to_str:N \l__unravel_tmpa_tl }
4956   }
4957   {
4958     \__unravel_prev_input:V \l__unravel_head_tl
4959     \__unravel_prev_input_gpop:N \l__unravel_head_tl
4960     \__unravel_back_input_tl_o:
4961   }
4962 }
4963 \cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
4964 {
4965   \__unravel_scan_toks_to_str:
4966   \__unravel_scan_toks_to_str:
4967 }
4968 \cs_new_protected:Npn \__unravel_scan_pdfximagebbox:
4969 { \__unravel_scan_int: \__unravel_scan_int: }
4970 \cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:
4971 {
4972   \__unravel_scan_keyword:nTF { pPaAgGeE }
4973   { \bool_set_true:N \l__unravel_tmpa_bool }
4974   { \bool_set_false:N \l__unravel_tmpb_bool }
4975   \__unravel_scan_keyword:nF { dDiIrReEcCtT }
4976   { \__unravel_scan_keyword:n { pPaAgGeE } }
4977   \__unravel_scan_toks_to_str:
4978 }
4979 \cs_new_protected:Npn \__unravel_scan_pdffiledump:
4980 {
4981   \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
4982   \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
4983   \__unravel_scan_pdf_ext_toks:
4984 }
4985 \cs_new_protected:Npn \__unravel_scan_pdfmatch:
4986 {
4987   \__unravel_scan_keyword:n { iIcCaAsSeE }
4988   \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
4989   { \__unravel_scan_int: }
4990   \__unravel_scan_pdf_ext_toks:
4991   \__unravel_scan_pdf_ext_toks:
4992 }
4993 \sys_if_engine luatex:T
4994 {
4995   \cs_new_protected:Npn \__unravel_scan_directlua:
4996   {
4997     \__unravel_get_x_non_relax:
4998     \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
4999     { \__unravel_back_input: }
5000     {
5001       \__unravel_scan_int:
5002       \__unravel_get_x_non_relax:
5003     }
5004     \__unravel_scan_pdf_ext_toks:
5005   }
5006 }

```

```

\__unravel_get_the:N #1 is \__unravel_get_token_xdef: in \edef or \xdef, \__unravel_get_token_x: in
\message and the like, and can be other commands.

5007 \__unravel_new_tex_expandable:nm { the } % 111
5008 { \__unravel_get_the:N }
5009 \cs_new_protected:Npn \__unravel_get_the:N #1
5010 {
5011 \__unravel_prev_input_gpush:N \l__unravel_head_tl
5012 \__unravel_print_expansion:
5013 \int_if_odd:nTF \l__unravel_head_char_int
5014 { % \unexpanded, \detokenize
5015 \__unravel_scan_toks:NN \c_false_bool \c_false_bool
5016 \__unravel_prev_input_gpop:N \l__unravel_head_tl
5017 \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
5018 }
5019 { % \the
5020 \__unravel_get_x_next:
5021 \__unravel_rescan_something_internal:n { 5 }
5022 \__unravel_prev_input_gpop:N \l__unravel_head_tl
5023 \__unravel_set_action_text:x
5024 {
5025 \tl_head:N \l__unravel_head_tl
5026 => \tl_tail:N \l__unravel_head_tl
5027 }
5028 \tl_set:Nx \l__unravel_head_tl
5029 { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
5030 }
5031 \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
5032 {
5033 \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
5034 \__unravel_prev_input_silent:x { \l__unravel_head_tl }
5035 \__unravel_print_action:
5036 }
5037 {
5038 \cs_if_eq:NNTF #1 \__unravel_get_token_x:
5039 {
5040 \__unravel_exp_args:NNx \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
5041 \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
5042 }
5043 {
5044 \tl_set:Nx \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
5045 \__unravel_back_input:V \l__unravel_tmpa_tl
5046 }
5047 \__unravel_print_expansion:
5048 }
5049 #1
5050 }

```

(End definition for __unravel_get_the:N.)

```

5051 \__unravel_new_tex_expandable:nm { top_bot_mark } % 112
5052 { \__unravel_back_input_tl_o: }

5053 \__unravel_new_tex_expandable:nm { end_template } % 117
5054 { \__unravel_back_input_tl_o: }

```


2.13.1 Conditionals

```

    \_unravel_pass_text:
\_unravel_pass_text_done:w
5055 \cs_new_protected:Npn \_unravel_pass_text:
5056 {
5057   \_unravel_input_if_empty:TF
5058   { \_unravel_pass_text_empty: }
5059   {
5060     \_unravel_input_get:N \l__unravel_tmpb_gtl
5061     \if_true:
5062       \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
5063       \exp_after:wN \_unravel_pass_text_done:w
5064       \fi:
5065       \_unravel_input_gpop:N \l__unravel_tmpb_gtl
5066       \exp_after:wN \_unravel_pass_text:
5067     \else:
5068       \use:c { fi: }
5069       \int_set:Nn \l__unravel_if_nesting_int { 1 }
5070       \_unravel_input_gpop:N \l__unravel_tmpb_gtl
5071       \exp_after:wN \_unravel_pass_text_nested:
5072     \fi:
5073   }
5074 }
5075 \cs_new_protected:Npn \_unravel_pass_text_done:w
5076 {
5077   \_unravel_get_next:
5078   \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
5079   \else:
5080 }

```

(End definition for _unravel_pass_text: and _unravel_pass_text_done:w.)

_unravel_pass_text_nested: Again, if there is no more input we are in trouble. The construction otherwise essentially results in

```

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

```

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

```

5081 \cs_new_protected:Npn \_unravel_pass_text_nested:
5082 {
5083   \_unravel_input_if_empty:TF
5084   { \_unravel_pass_text_empty: }
5085   {
5086     \_unravel_input_get:N \l__unravel_tmpb_gtl
5087     \if_true:
5088       \if_true:
5089       \gtl_head_do:NN \l__unravel_tmpb_gtl \else:

```

```

5090         \int_decr:N \l__unravel_if_nesting_int
5091         \use_none:nnnnn
5092         \fi:
5093         \use_none:nmn
5094         \fi:
5095         \int_incr:N \l__unravel_if_nesting_int
5096         \fi:
5097         \__unravel_input_gpop:N \l__unravel_unused_gtl
5098         \int_compare:nNnTF \l__unravel_if_nesting_int = 0
5099         { \__unravel_pass_text: }
5100         { \__unravel_pass_text_nested: }
5101     }
5102 }

```

(End definition for __unravel_pass_text_nested:.)

__unravel_pass_text_empty:

```

5103 \cs_new_protected:Npn \__unravel_pass_text_empty:
5104 {
5105     \__unravel_error:nnnnn { runaway-if } { } { } { } { }
5106     \__unravel_exit_hard:w
5107 }

```

(End definition for __unravel_pass_text_empty:.)

__unravel_cond_push:

__unravel_cond_pop:

```

5108 \cs_new_protected:Npn \__unravel_cond_push:
5109 {
5110     \tl_gput_left:Nx \g__unravel_if_limit_tl
5111     { { \int_use:N \g__unravel_if_limit_int } }
5112     \int_gincr:N \g__unravel_if_depth_int
5113     \int_gzero:N \g__unravel_if_limit_int
5114 }
5115 \cs_new_protected:Npn \__unravel_cond_pop:
5116 {
5117     \fi:
5118     \int_gset:Nn \g__unravel_if_limit_int
5119     { \tl_head:N \g__unravel_if_limit_tl }
5120     \tl_gset:Nx \g__unravel_if_limit_tl
5121     { \tl_tail:N \g__unravel_if_limit_tl }
5122     \int_gdecr:N \g__unravel_if_depth_int
5123 }

```

(End definition for __unravel_cond_push: and __unravel_cond_pop:.)

__unravel_change_if_limit:nn

```

5124 \cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
5125 {
5126     \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
5127     { \int_gset:Nn \g__unravel_if_limit_int {#1} }
5128     {
5129         \tl_clear:N \l__unravel_tmpa_tl
5130         \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
5131         {
5132             \tl_put_right:Nx \l__unravel_tmpa_tl

```

```

5133         { { \tl_head:N \g__unravel_if_limit_tl } }
5134         \tl_gset:Nx \g__unravel_if_limit_tl
5135         { \tl_tail:N \g__unravel_if_limit_tl }
5136     }
5137     \tl_gset:Nx \g__unravel_if_limit_tl
5138     { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
5139 }
5140 }

```

(End definition for _unravel_change_if_limit:nn.)

```

5141 \_unravel_new_tex_expandable:nn { if_test } % 107
5142 {
5143     \_unravel_cond_push:
5144     \exp_args:NV \_unravel_cond_aux:n \g__unravel_if_depth_int
5145 }

```

_unravel_cond_aux:nn

```

5146 \cs_new_protected:Npn \_unravel_cond_aux:n #1
5147 {
5148     \int_case:nnF \l__unravel_head_char_int
5149     {
5150         { 0 } { \_unravel_test_two_chars:nn { 0 } {#1} } % if
5151         { 1 } { \_unravel_test_two_chars:nn { 1 } {#1} } % ifcat
5152         { 12 } { \_unravel_test_ifx:n {#1} }
5153         { 16 } { \_unravel_test_case:n {#1} }
5154         { 20 } { \if_true: \_unravel_test_incsname:n {#1} }
5155         { 21 } { \if_true: \_unravel_test_pdfprimitive:n {#1} }
5156     }
5157     {
5158         \_unravel_prev_input_gpush:N \l__unravel_head_tl
5159         \_unravel_print_expansion:
5160         \int_case:nn \l__unravel_head_char_int
5161         {
5162             { 2 } % ifnum
5163             { \_unravel_test_two_vals:N \_unravel_scan_int: }
5164             { 3 } % ifdim
5165             { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
5166             { 4 } { \_unravel_scan_int: } % ifodd
5167             % { 5 } { } % if[hvm]mode, ifinner, if[tydm]dir
5168             { 9 } { \_unravel_scan_int: } % ifvoid, ifhbox, ifvbox etc
5169             { 13 } { \_unravel_scan_int: } % ifeof
5170             % { 14 } { } % iftrue
5171             % { 15 } { } % iffalse
5172             { 17 } { \_unravel_test_ifdefined: } % ifdefined
5173             { 18 } { \_unravel_test_ifcsname: } % ifcsname
5174             { 19 } % iffontchar
5175             { \_unravel_scan_font_ident: \_unravel_scan_int: }
5176             { 22 } % ifabsnum
5177             { \_unravel_test_two_vals:N \_unravel_scan_int: }
5178             { 23 } % ifabsdim
5179             { \_unravel_test_two_vals:N \_unravel_scan_normal_dimen: }
5180             { 24 } { \_unravel_scan_font_ident: } % ifjfont, iftfont
5181         }
5182         \_unravel_prev_input_gpop:N \l__unravel_head_tl

```

```

5183     \__unravel_set_action_text:x { \tl_to_str:N \l__unravel_head_tl }
5184     \l__unravel_head_tl \scan_stop:
5185     \__unravel_cond_true:NNNn
5186     \else:
5187     \__unravel_cond_false:Nn
5188     \fi:
5189     {#1}
5190   }
5191 }

```

(End definition for __unravel_cond_aux:nn.)

__unravel_cond_true:NNNn

```

5192 \cs_new_protected:Npn \__unravel_cond_true:NNNn #1#2#3#4
5193 {
5194   \__unravel_change_if_limit:nn { 3 } {#4} % wait for else/fi
5195   \__unravel_print_expansion:x { \g__unravel_action_text_str = true }
5196 }

```

(End definition for __unravel_cond_true:NNNn.)

__unravel_cond_false:Nn

__unravel_cond_false_loop:n
 __unravel_cond_false_common:

```

5197 \cs_new_protected:Npn \__unravel_cond_false:Nn #1#2
5198 {
5199   \__unravel_cond_false_loop:n {#2}
5200   \__unravel_cond_false_common:
5201   \__unravel_print_expansion:x
5202   {
5203     \g__unravel_action_text_str = false ~
5204     => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
5205   }
5206 }
5207 \cs_new_protected:Npn \__unravel_cond_false_loop:n #1
5208 {
5209   \__unravel_pass_text:
5210   \int_compare:nNnTF \g__unravel_if_depth_int = {#1}
5211   {
5212     \token_if_eq_meaning:NNT \l__unravel_head_token \or:
5213     {
5214       \__unravel_error:nnnnn { extra-or } { } { } { } { }
5215       \__unravel_cond_false_loop:n {#1}
5216     }
5217   }
5218   {
5219     \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
5220     { \__unravel_cond_pop: }
5221     \__unravel_cond_false_loop:n {#1}
5222   }
5223 }
5224 \cs_new_protected:Npn \__unravel_cond_false_common:
5225 {
5226   \token_if_eq_meaning:NNTF \l__unravel_head_token \fi:
5227   { \__unravel_cond_pop: }
5228   { \int_gset:Nn \g__unravel_if_limit_int { 2 } } % wait for fi
5229 }

```

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

_unravel_test_two_vals:N

```

5230 \cs_new_protected:Npn \_unravel_test_two_vals:N #1
5231 {
5232   #1
5233   \_unravel_get_x_non_blank:
5234   \_unravel_tl_if_in:ooTF { < = > } \l__unravel_head_tl { }
5235   {
5236     \_unravel_error:nmnnn { missing-equals } { } { } { } { }
5237     \_unravel_back_input:
5238     \tl_set:Nn \l__unravel_head_tl { = }
5239   }
5240   \_unravel_prev_input:V \l__unravel_head_tl
5241   #1
5242 }

```

(End definition for _unravel_test_two_vals:N.)

_unravel_test_two_chars:nn

_unravel_test_two_chars_get:n
_unravel_test_two_chars_gtl:N

```

5243 \cs_new_protected:Npn \_unravel_test_two_chars:nn #1
5244 {
5245   \exp_args:NNo \gtl_set:Nn \l__unravel_head_gtl { \l__unravel_head_tl }
5246   \_unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
5247   \_unravel_print_expansion:
5248   \_unravel_test_two_chars_get:n {#1}
5249   \_unravel_test_two_chars_get:n {#1}
5250   \_unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
5251   \_unravel_set_action_text:x { \gtl_to_str:N \l__unravel_head_gtl }
5252   \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
5253   \exp_args:No \tl_if_head_eq_meaning:nNT \l__unravel_head_tl \reverse_if:N
5254   {
5255     \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
5256     \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
5257   }
5258   \gtl_pop_left:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
5259   \_unravel_test_two_chars_gtl:N \l__unravel_tmpb_gtl
5260   \_unravel_test_two_chars_gtl:N \l__unravel_head_gtl
5261   \l__unravel_head_tl \scan_stop:
5262   \_unravel_cond_true:NNNn
5263   \else:
5264   \_unravel_cond_false:Nn
5265   \fi:
5266 }
5267 \cs_new_protected:Npn \_unravel_test_two_chars_get:n #1
5268 {
5269   \_unravel_get_x_next:
5270   \int_compare:nNnT {#1} = 0
5271   {
5272     \gtl_if_head_is_N_type:NF \l__unravel_head_gtl
5273     { \gtl_set:Nx \l__unravel_head_gtl { \gtl_to_str:N \l__unravel_head_gtl } }
5274   }
5275   \_unravel_prev_input_gtl:N \l__unravel_head_gtl

```

```

5276     \__unravel_print_action:x { \gtl_to_str:N \l__unravel_head_gtl }
5277   }
5278   \cs_new_protected:Npn \__unravel_test_two_chars_gtl:N #1
5279   {
5280     \tl_put_right:Nx \l__unravel_head_tl
5281     {
5282       \gtl_if_head_is_group_begin:NTF #1 { \c_group_begin_token }
5283       {
5284         \gtl_if_head_is_group_end:NTF #1 { \c_group_end_token }
5285         {
5286           \exp_not:N \exp_not:N
5287           \exp_not:f { \gtl_head_do:NN #1 \exp_stop_f: }
5288         }
5289       }
5290     }
5291   }

```

(End definition for __unravel_test_two_chars:nn, __unravel_test_two_chars_get:n, and __unravel_test_two_chars_gtl:N.)

```

\__unravel_test_ifx:n
\__unravel_test_ifx_str:NN
\__unravel_test_ifx_aux:NNN
\__unravel_test_ifx_aux:w

```

The token equal to \ifx is pushed as a previous input to show an action nicely, then retrieved as \l__unravel_tmpa_tl after getting the next two tokens as tmpb and head. Then we call \l__unravel_tmpa_tl followed by these two tokens. A previous implementation made sure to get these tokens from unpacking the gtl, presumably (I should have documented, now I might be missing something) to deal nicely with the master counter in case these tokens are braces. On the other hand we must take care of tokens affected by \noexpand and whose current definition is expandable, in which case the trustworthy \meaning is that of the \l__unravel_head_token or \l__unravel_tmpb_token rather than that of the token in \l__unravel_head_gtl or \l__unravel_tmpb_gtl.

```

5292   \cs_new_protected:Npn \__unravel_test_ifx:n #1
5293   {
5294     \__unravel_prev_input_gpush:N \l__unravel_head_tl
5295     \__unravel_print_expansion:
5296     \__unravel_get_next:
5297     \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
5298     \cs_set_eq:NN \l__unravel_tmpb_token \l__unravel_head_token
5299     \__unravel_get_next:
5300     \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
5301     \__unravel_set_action_text:x
5302     {
5303       Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
5304       \__unravel_test_ifx_str:NN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
5305       \__unravel_test_ifx_str:NN \l__unravel_head_token \l__unravel_head_gtl
5306     }
5307     \__unravel_test_ifx_aux:NNN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
5308     \__unravel_test_ifx_aux:w
5309     \__unravel_cond_true:NNNn
5310     \else:
5311     \__unravel_cond_false:Nn
5312     \fi:
5313     {#1}
5314   }
5315   \cs_new:Npn \__unravel_test_ifx_str:NN #1#2
5316   {

```

```

5317     \token_if_eq_meaning:NNT #1 \__unravel_special_relax:
5318     { \iow_char:N \notexpanded: }
5319     \gtl_to_str:N #2
5320   }
5321 \cs_new_protected:Npn \__unravel_test_ifx_aux:NNN #1#2#3
5322 {
5323   \token_if_eq_meaning:NNTF #1 \__unravel_special_relax:
5324   {
5325     \gtl_head_do:NN #2 \__unravel_token_if_expandable:NTF
5326     { #3 #1 } { \gtl_head_do:NN #2 #3 }
5327   }
5328   { \gtl_head_do:NN #2 #3 }
5329 }
5330 \cs_new:Npn \__unravel_test_ifx_aux:w
5331 {
5332   \__unravel_test_ifx_aux:NNN \l__unravel_head_token \l__unravel_head_gtl
5333   \l__unravel_tmpa_tl
5334 }

```

(End definition for __unravel_test_ifx:n and others.)

```

\__unravel_test_case:n
\__unravel_test_case_aux:nn
5335 \cs_new_protected:Npn \__unravel_test_case:n #1
5336 {
5337   \if_case:w 0 ~
5338   \__unravel_prev_input_gpush:N \l__unravel_head_tl
5339   \__unravel_print_expansion:
5340   \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
5341   \__unravel_scan_int:
5342   \__unravel_prev_input_get:N \l__unravel_head_tl
5343   \tl_set:Nx \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
5344   % ^^A does text_case_aux use prev_input_seq?
5345   \int_compare:nNnF { \l__unravel_head_tl } = 0
5346   {
5347     \int_compare:nNnTF { \l__unravel_head_tl } > 0
5348     { \fi: \if_case:w 1 ~ \or: }
5349     { \fi: \if_case:w -1 ~ \else: }
5350   }
5351   \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
5352   \__unravel_prev_input_gpop:N \l__unravel_head_tl
5353   \__unravel_print_expansion:x { \tl_to_str:N \l__unravel_head_tl }
5354 }
5355 \cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
5356 {
5357   \int_compare:nNnTF {#1} = 0
5358   { \__unravel_change_if_limit:nn { 4 } {#2} }
5359   {
5360     \__unravel_pass_text:
5361     \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
5362     {
5363       \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
5364       {
5365         \exp_args:Nf \__unravel_test_case_aux:nn
5366         { \int_eval:n { #1 - 1 } } {#2}

```

```

5367     }
5368     { \_unravel_cond_false_common: }
5369   }
5370   {
5371     \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
5372     { \_unravel_cond_pop: }
5373     \_unravel_test_case_aux:nn {#1} {#2}
5374   }
5375 }
5376 }

```

(End definition for _unravel_test_case:n and _unravel_test_case_aux:nn.)

_unravel_test_incsname:n

```

5377 \cs_new_protected:Npn \_unravel_test_incsname:n #1
5378 { \_unravel_not_implemented:n { ifincsname } }

```

(End definition for _unravel_test_incsname:n.)

_unravel_test_pdfprimitive:n

```

5379 \cs_new_protected:Npn \_unravel_test_pdfprimitive:n #1
5380 { \_unravel_not_implemented:n { ifpdfprimitive } }

```

(End definition for _unravel_test_pdfprimitive:n.)

_unravel_test_ifdefined:

```

5381 \cs_new_protected:Npn \_unravel_test_ifdefined:
5382 {
5383   \_unravel_input_if_empty:TF
5384   { \_unravel_pass_text_empty: }
5385   {
5386     \_unravel_input_gpop:N \l__unravel_tmpb_gtl
5387     \_unravel_set_action_text:x
5388     {
5389       Conditional:~ \tl_to_str:N \l__unravel_head_tl
5390       \gtl_to_str:N \l__unravel_tmpb_gtl
5391     }
5392     \_unravel_prev_input:x
5393     {
5394       \gtl_if_tl:NTF \l__unravel_tmpb_gtl
5395       { \gtl_head:N \l__unravel_tmpb_gtl }
5396       { \gtl_to_str:N \l__unravel_tmpb_gtl }
5397     }
5398   }
5399 }

```

(End definition for _unravel_test_ifdefined:.)

_unravel_test_ifcsname:

```

5400 \cs_new_protected:Npn \_unravel_test_ifcsname:
5401 {
5402   \_unravel_csname_loop:
5403   \_unravel_prev_input:V \l__unravel_head_tl
5404   \_unravel_get_lastnamedcs:
5405 }

```


(End definition for `_unravel_test_ifcsname:`)

```

5406 \_unravel_new_tex_expandable:nn { fi_or_else } % 108
5407 {
5408   \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
5409   {
5410     \int_compare:nNnTF \g__unravel_if_limit_int = 0
5411     {
5412       \int_compare:nNnTF \g__unravel_if_depth_int = 0
5413       { \_unravel_error:nmnnn { extra-fi-or-else } { } { } { } { } }
5414       { \_unravel_insert_relax: }
5415     }
5416     { \_unravel_error:nmnnn { extra-fi-or-else } { } { } { } { } }
5417   }
5418   {
5419     \_unravel_set_action_text:
5420     \int_compare:nNnF \l__unravel_head_char_int = 2
5421     {
5422       \_unravel-fi-or-else-loop:
5423       \_unravel_set_action_text:x
5424       {
5425         \g__unravel_action_text_str \c_space_tl
5426         => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
5427       }
5428     }
5429     \_unravel_print_expansion:
5430     \_unravel_cond_pop:
5431   }
5432 }
5433 \cs_new_protected:Npn \_unravel-fi-or-else-loop:
5434 {
5435   \int_compare:nNnF \l__unravel_head_char_int = 2
5436   {
5437     \_unravel_pass_text:
5438     \_unravel_set_cmd:
5439     \_unravel-fi-or-else-loop:
5440   }
5441 }

```

2.14 User interaction

2.14.1 Print

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

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

`\l__unravel_print_tl`

```

5442 \tl_new:N \l__unravel_print_tl
5443 \sys_if_engine luatex:TF
5444 {
5445   \cs_new_protected:Npx \_unravel_print_normalize_null:
5446   {
5447     \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl

```

```

5448         { \char_generate:nn { 0 } { 12 } }
5449         { \tl_to_str:n { ^^ @ } }
5450     }
5451 }
5452 { \cs_new_protected:Npn \__unravel_print_normalize_null: { } }

```

(End definition for __unravel_print_normalize_null: and \l__unravel_print_tl.)

```

\__unravel_print:n
\__unravel_print:x
\__unravel_log:n

```

```

5453 \cs_new_protected:Npn \__unravel_print:n #1
5454 {
5455     \tl_set:Nn \l__unravel_print_tl {#1}
5456     \__unravel_print_normalize_null:
5457     \__unravel_exp_args:Nx \iow_term:n { \l__unravel_print_tl }
5458 }
5459 \cs_new_protected:Npn \__unravel_print:x
5460 { \__unravel_exp_args:Nx \__unravel_print:n }
5461 \cs_new_protected:Npn \__unravel_log:n #1
5462 {
5463     \tl_set:Nn \l__unravel_print_tl {#1}
5464     \__unravel_print_normalize_null:
5465     \__unravel_exp_args:Nx \iow_log:n { \l__unravel_print_tl }
5466 }

```

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

```

\__unravel_print_message:nn

```

The message to be printed should come already detokenized, as #2. It will be wrapped to 80 characters per line, with #1 before each line. The message is properly suppressed (or sent only to the log) according to \g__unravel_online_int.

```

5467 \cs_new_protected:Npn \__unravel_print_message:nn #1 #2
5468 {
5469     \int_compare:nNnF \g__unravel_online_int < 0
5470     {
5471         \int_compare:nNnTF \g__unravel_online_int = 0
5472         { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_log:n }
5473         { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_print:n }
5474     }
5475 }

```

(End definition for __unravel_print_message:nn.)

```

\__unravel_set_action_text:x

```

```

5476 \cs_new_protected:Npn \__unravel_set_action_text:x #1
5477 {
5478     \group_begin:
5479     \__unravel_set_escapechar:n { 92 }
5480     \str_gset:Nx \g__unravel_action_text_str {#1}
5481     \group_end:
5482 }

```

(End definition for __unravel_set_action_text:x.)

`__unravel_set_action_text:`

```
5483 \cs_new_protected:Npn \__unravel_set_action_text:
5484 {
5485   \__unravel_set_action_text:x
5486   {
5487     \tl_to_str:N \l__unravel_head_tl
5488     \tl_if_single_token:VT \l__unravel_head_tl
5489     { = ~ \token_to_meaning:N \l__unravel_head_token }
5490   }
5491 }
```

(End definition for __unravel_set_action_text:.)

`__unravel_print_state:`

```
5492 \cs_new_protected:Npn \__unravel_print_state:
5493 {
5494   \group_begin:
5495   \__unravel_set_escapechar:n { 92 }
5496   \tl_use:N \g__unravel_before_print_state_tl
5497   \int_compare:nNnT \g__unravel_online_int > 0
5498   {
5499     \__unravel_print_state_output:
5500     \__unravel_print_state_prev:
5501     \__unravel_print_state_input:
5502   }
5503   \group_end:
5504 }
```

(End definition for __unravel_print_state:.)

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

`__unravel_print_state_output:n`

```
5505 \cs_new_protected:Npn \__unravel_print_state_output:
5506 {
5507   \__unravel_exp_args:Nx \__unravel_print_state_output:n
5508   { \gtl_to_str:N \g__unravel_output_gtl }
5509 }
5510 \cs_new_protected:Npn \__unravel_print_state_output:n #1
5511 {
5512   \tl_if_empty:nF {#1}
5513   {
5514     \__unravel_print_message:nn { <| ~ }
5515     { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
5516   }
5517 }
```

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

`__unravel_print_state_prev:` Never trim ##1.

```
5518 \cs_new_protected:Npn \__unravel_print_state_prev:
5519 {
5520   \seq_set_map_x:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
5521   { \__unravel_to_str:Nn ##1 }
```

```

5522 \seq_remove_all:Nn \l__unravel_tmpa_seq { }
5523 \seq_if_empty:NTF \l__unravel_tmpa_seq
5524 { \__unravel_print_message:nn { || ~ } { } }
5525 {
5526 \seq_map_inline:Nn \l__unravel_tmpa_seq
5527 {
5528 \__unravel_print_message:nn { || ~ } {##1}
5529 }
5530 }
5531 }

```

(End definition for `__unravel_print_state_prev:.`)

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

`__unravel_print_state_input:n`

```

5532 \cs_new_protected:Npn \__unravel_print_state_input:
5533 {
5534 \__unravel_exp_args:Nx \__unravel_print_state_input:n
5535 { \__unravel_input_to_str: }
5536 }
5537 \cs_new_protected:Npn \__unravel_print_state_input:n #1
5538 {
5539 \__unravel_print_message:nn { |> ~ }
5540 { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
5541 }

```

(End definition for `__unravel_print_state_input:` and `__unravel_print_state_input:n.`)

`__unravel_print_meaning:`

```

5542 \cs_new_protected:Npn \__unravel_print_meaning:
5543 {
5544 \__unravel_input_if_empty:TF
5545 { \__unravel_print_message:nn { } { Empty-input! } }
5546 {
5547 \__unravel_input_get:N \l__unravel_tmpb_gtl
5548 \__unravel_print_message:nn { }
5549 {
5550 \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
5551 = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
5552 }
5553 }
5554 }

```

(End definition for `__unravel_print_meaning:.`)

`__unravel_print_action:` Some of these commands are currently synonyms but we may decide to make some options act differently on them.

`__unravel_print_action:x`

`__unravel_print_assignment:`

`__unravel_print_assignment:x`

`__unravel_print_expansion:`

`__unravel_print_expansion:x`

`__unravel_print_action_aux:N`

```

5555 \cs_new_protected:Npn \__unravel_print_action:
5556 { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
5557 \cs_new_protected:Npn \__unravel_print_action:x #1
5558 {
5559 \__unravel_set_action_text:x {#1}
5560 \__unravel_print_action:

```

```

5561 }
5562 \cs_new_protected:Npn \__unravel_print_assignment:
5563 { \__unravel_print_action_aux:N \g__unravel_trace_assigns_bool }
5564 \cs_new_protected:Npn \__unravel_print_assignment:x #1
5565 {
5566   \__unravel_set_action_text:x {#1}
5567   \__unravel_print_assignment:
5568 }
5569 \cs_new_protected:Npn \__unravel_print_expansion:
5570 { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
5571 \cs_new_protected:Npn \__unravel_print_expansion:x #1
5572 {
5573   \__unravel_set_action_text:x {#1}
5574   \__unravel_print_expansion:
5575 }
5576 \cs_new_protected:Npn \__unravel_print_action_aux:N #1
5577 {
5578   \int_gdecr:N \g__unravel_nonstop_int
5579   \int_gincr:N \g__unravel_step_int
5580   \bool_if:NT #1
5581   {
5582     \__unravel_print:x
5583     {
5584       [====
5585       \bool_if:NT \g__unravel_number_steps_bool
5586         { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
5587       =====]~
5588       \int_compare:nNnTF
5589         { \str_count:N \g__unravel_action_text_str
5590         > { \g__unravel_max_action_int }
5591         {
5592           \str_range:Nnn \g__unravel_action_text_str
5593             { 1 } { \g__unravel_max_action_int - 3 } ...
5594         }
5595         { \g__unravel_action_text_str }
5596       }
5597     \__unravel_print_state:
5598     \__unravel_prompt:
5599   }
5600 }

```

(End definition for __unravel_print_action: and others.)

```

\__unravel_just_print_assigned_token:
  \__unravel_print_assigned_token:
  \__unravel_print_assigned_register:

```

```

5601 \cs_new_protected:Npn \__unravel_just_print_assigned_token:
5602 {
5603   \__unravel_print_assignment:x
5604   {
5605     Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5606     = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
5607   }
5608 }
5609 \cs_new_protected:Npn \__unravel_print_assigned_token:
5610 {

```

```

5611     \__unravel_after_assignment:
5612     \__unravel_just_print_assigned_token:
5613     \__unravel_omit_after_assignment:w
5614 }
5615 \cs_new_protected:Npn \__unravel_print_assigned_register:
5616 {
5617     \__unravel_after_assignment:
5618     \__unravel_exp_args:Nx \__unravel_print_assignment:x
5619     {
5620         \exp_not:n
5621         {
5622             Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
5623             \tl_if_single:NT \l__unravel_defined_tl
5624             { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
5625         }
5626         = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
5627     }
5628     \__unravel_omit_after_assignment:w
5629 }

```

(End definition for __unravel_just_print_assigned_token:, __unravel_print_assigned_token:, and __unravel_print_assigned_register:.)

__unravel_print_welcome: Welcome message.

```

5630 \cs_new_protected:Npn \__unravel_print_welcome:
5631 {
5632     \__unravel_print_message:nn { }
5633     {
5634         \bool_if:NTF \g__unravel_welcome_message_bool
5635         {
5636             \\\
5637             =====~ Welcome~ to~ the~ unravel~ package~ =====\\
5638             \iow_indent:n
5639             {
5640                 "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\\
5641                 "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\\
5642                 "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\\
5643                 Press~<enter>~to~continue;~'h'~<enter>~for~help. \\\
5644             }
5645         }
5646         { [====~Start~====] }
5647     }
5648     \__unravel_print_state:
5649     \__unravel_prompt:
5650 }

```

(End definition for __unravel_print_welcome:.)

__unravel_print_outcome: Final message.

```

5651 \cs_new_protected:Npn \__unravel_print_outcome:
5652 { \__unravel_print_message:nn { } { [====~End~====] } }

```

(End definition for __unravel_print_outcome:.)

2.14.2 Prompt

```

\__unravel_ior_str_get:NN
\__unravel_ior_str_get:Nc
5653 \cs_new_protected:Npn \__unravel_ior_str_get:NN #1#2
5654 { \tex_readline:D #1 to #2 }
5655 \cs_generate_variant:Nn \__unravel_ior_str_get:NN { Nc }

(End definition for \__unravel_ior_str_get:NN.)

\__unravel_prompt:
5656 \cs_new_protected:Npn \__unravel_prompt:
5657 {
5658   \int_compare:nNnF \g__unravel_nonstop_int > 0
5659   {
5660     \group_begin:
5661     \__unravel_set_escapechar:n { -1 }
5662     \int_set:Nn \tex_endlinechar:D { -1 }
5663     \tl_use:N \g__unravel_before_prompt_tl
5664     \__unravel_prompt_aux:
5665     \group_end:
5666   }
5667 }
5668 \cs_new_protected:Npn \__unravel_prompt_aux:
5669 {
5670   \clist_if_empty:NTF \g__unravel_prompt_input_clist
5671   {
5672     \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
5673     {
5674       \bool_if:NTF \g__unravel_explicit_prompt_bool
5675       { \__unravel_ior_str_get:Nc \c__unravel_prompt_ior }
5676       { \__unravel_ior_str_get:Nc \c__unravel_noprompt_ior }
5677       { Your~input }
5678       \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
5679     }
5680   }
5681   {
5682     \clist_gpop:NN \g__unravel_prompt_input_clist \l__unravel_tmpa_tl
5683     \group_begin:
5684     \__unravel_set_escapechar:n { 92 }
5685     \__unravel_print:x
5686     {
5687       \bool_if:NT \g__unravel_explicit_prompt_bool { Your~input= }
5688       \tl_to_str:N \l__unravel_tmpa_tl
5689     }
5690     \group_end:
5691     \exp_args:Nv \__unravel_prompt_treat:n \l__unravel_tmpa_tl
5692   }
5693 }
5694 \cs_new_protected:Npn \__unravel_prompt_treat:n #1
5695 {
5696   \tl_if_empty:nF {#1}
5697   {
5698     \__unravel_exp_args:Nx \str_case:nnF { \tl_head:n {#1} }
5699     {

```

```

5700     { m } { \_unravel_print_meaning: \_unravel_prompt_aux: }
5701     { q }
5702     {
5703         \int_gset:Nn \g__unravel_online_int { -1 }
5704         \int_gzero:N \g__unravel_nonstop_int
5705     }
5706     { x }
5707     {
5708         \group_end:
5709         \_unravel_exit_hard:w
5710     }
5711     { X }
5712     {
5713         \tex_batchmode:D
5714         \tex_read:D -1 to \l__unravel_tmpa_tl
5715     }
5716     { s } { \_unravel_prompt_scan_int:nn {#1}
5717         \_unravel_prompt_silent_steps:n }
5718     { o } { \_unravel_prompt_scan_int:nn {#1}
5719         { \int_gset:Nn \g__unravel_online_int } }
5720     { C }
5721     {
5722         \_unravel_exp_args:Nx \use:n
5723         {
5724             \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
5725             { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
5726         }
5727         \tl_gput_left:Nn \g__unravel_tmpc_tl
5728         { \tl_gclear:N \g__unravel_tmpc_tl }
5729         \group_insert_after:N \g__unravel_tmpc_tl
5730         \group_insert_after:N \_unravel_prompt:
5731     }
5732     { | } { \_unravel_prompt_scan_int:nn {#1}
5733         \_unravel_prompt_vert:n }
5734     { u } { \_unravel_prompt_until:n {#1} }
5735     { a } { \_unravel_prompt_all: }
5736     }
5737     { \_unravel_prompt_help: }
5738 }
5739 }
5740 \cs_new_protected:Npn \_unravel_prompt_scan_int:nn #1
5741 {
5742     \tex_afterassignment:D \_unravel_prompt_scan_int_after:wn
5743     \l__unravel_prompt_tmpa_int =
5744     \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
5745     \use_ii:nn #1 \scan_stop:
5746 }
5747 \cs_new_protected:Npn \_unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
5748 {
5749     #2 \l__unravel_prompt_tmpa_int
5750     \tl_if_blank:nF {#1} { \_unravel_prompt_treat:n {#1} }
5751 }
5752 \cs_new_protected:Npn \_unravel_prompt_help:
5753 {

```



```

5754 \__unravel_print:n { "m":~meaning-of~first~token }
5755 \__unravel_print:n { "a":~print~state~again,~without~truncating }
5756 \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
5757 \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
5758 \__unravel_print:n { "u<text>":~silent~steps~until~the~input~starts~with~<text> }
5759 \__unravel_print:n
5760 { "o<num>":~1~<=>~log~and~terminal,~0~<=>~only~log,~-1~<=>~neither.}
5761 \__unravel_print:n { "q":~semi-quiet~(same~as~"o-1") }
5762 \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
5763 \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
5764 \__unravel_prompt_aux:
5765 }
5766 \cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
5767 {
5768   \int_compare:nNnF {#1} < 0
5769   {
5770     \int_gset:Nn \g__unravel_online_int { -1 }
5771     \tl_gset:Nn \g__unravel_before_prompt_tl
5772     {
5773       \int_gset:Nn \g__unravel_online_int { 1 }
5774       \tl_gclear:N \g__unravel_before_prompt_tl
5775     }
5776     \int_gset:Nn \g__unravel_nonstop_int {#1}
5777   }
5778 }
5779 \cs_new_protected:Npn \__unravel_prompt_vert:n #1
5780 {
5781   \int_compare:nNnTF {#1} < { 0 }
5782   { \__unravel_prompt_vert:Nn > {#1} }
5783   { \__unravel_prompt_vert:Nn < {#1} }
5784 }
5785 \cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
5786 {
5787   \int_gset:Nn \g__unravel_online_int { -1 }
5788   \tl_gset:Nf \g__unravel_before_print_state_tl
5789   {
5790     \exp_args:Nnf \exp_stop_f: \int_compare:nNnTF
5791     { \int_eval:n { \__unravel_prev_input_count: - #2 } }
5792     #1 { \__unravel_prev_input_count: }
5793     {
5794       \int_gset:Nn \g__unravel_nonstop_int
5795       { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5796     }
5797     {
5798       \int_gset:Nn \g__unravel_online_int { 1 }
5799       \tl_gclear:N \g__unravel_before_print_state_tl
5800     }
5801   }
5802 }
5803 \cs_new_protected:Npn \__unravel_prompt_all:
5804 {
5805   \tl_gset:Nx \g__unravel_tmpc_tl
5806   {
5807     \exp_not:n

```

```

5808     {
5809         \tl_gclear:N \g__unravel_tmpc_tl
5810         \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
5811         \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
5812         \__unravel_print_state:
5813         \int_gdecr:N \g__unravel_nonstop_int
5814         \__unravel_prompt:
5815     }
5816     \__unravel_prompt_all_aux:N \g__unravel_max_output_int
5817     \__unravel_prompt_all_aux:N \g__unravel_max_input_int
5818 }
5819 \group_insert_after:N \g__unravel_tmpc_tl
5820 }
5821 \cs_new:Npn \__unravel_prompt_all_aux:N #1
5822 { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }

```

(End definition for __unravel_prompt:.)

```

\__unravel_prompt_until:n
  \g__unravel_until_tl
5823 \tl_new:N \g__unravel_until_tl
5824 \cs_new_protected:Npn \__unravel_prompt_until:n #1
5825 {
5826     \tl_gset:Nx \g__unravel_until_tl { \tl_tail:n {#1} }
5827     \int_gset:Nn \g__unravel_online_int { -1 }
5828     \tl_gset:Nn \g__unravel_before_print_state_tl
5829     {
5830         \__unravel_input_get_left:N \l__unravel_tmpa_tl
5831         \__unravel_exp_args:Nx \use:n
5832         {
5833             \exp_not:N \tl_if_in:nnTF
5834             { \exp_not:N \__unravel:n \tl_to_str:N \l__unravel_tmpa_tl }
5835             { \exp_not:N \__unravel:n \tl_to_str:N \g__unravel_until_tl }
5836         }
5837         {
5838             \int_gzero:N \g__unravel_nonstop_int
5839             \int_gset:Nn \g__unravel_online_int { 1 }
5840             \tl_gclear:N \g__unravel_before_print_state_tl
5841         }
5842         {
5843             \int_gset:Nn \g__unravel_nonstop_int
5844             { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
5845         }
5846     }
5847 }

```

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

2.14.3 Errors

```

\__unravel_not_implemented:n
5848 \cs_new_protected:Npn \__unravel_not_implemented:n #1
5849 { \__unravel_error:nnnnn { not-implemented } {#1} { } { } { } }

```

(End definition for __unravel_not_implemented:n.)

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

```

\__unravel_error:nxxxx
5850 \cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
5851 {
5852   \group_begin:
5853   \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
5854   \group_end:
5855 }
5856 \cs_new_protected:Npn \__unravel_error:nxxxx #1#2#3#4#5
5857 {
5858   \group_begin:
5859   \msg_error:nnxxxx { unravel } {#1} {#2} {#3} {#4} {#5}
5860   \group_end:
5861 }

```

(End definition for __unravel_error:nnnnn.)

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

```

5862 \cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
5863 {
5864   \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
5865   \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
5866 }

```

(End definition for __unravel_tex_msg_new:nnn.)

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

```

\__unravel_tex_error:nV
5867 \cs_new_protected:Npn \__unravel_tex_error:nn #1#2
5868 {
5869   \group_begin:
5870   \msg_error:nnxxx { unravel } { tex-error }
5871   { \tl_to_str:n {#2} }
5872   { \use:c { __unravel_tex_msg_error_#1: } }
5873   { \use:c { __unravel_tex_msg_help_#1: } }
5874   \group_end:
5875 }
5876 \cs_generate_variant:Nn \__unravel_tex_error:nn { nV }

```

(End definition for __unravel_tex_error:nn.)

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

```

\__unravel_tex_fatal_error:nV
5877 \cs_new_protected:Npn \__unravel_tex_fatal_error:nn #1#2
5878 {
5879   \__unravel_error:nxxxx { tex-fatal }
5880   { \tl_to_str:n {#2} }
5881   { \use:c { __unravel_tex_msg_error_#1: } }
5882   { \use:c { __unravel_tex_msg_help_#1: } }
5883   { }
5884 }
5885 \cs_generate_variant:Nn \__unravel_tex_fatal_error:nn { nV }

```

(End definition for __unravel_tex_fatal_error:nn.)

2.15 Keys

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

```
5886 \keys_define:nn { unravel/defaults }
5887 {
5888   explicit-prompt .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
5889   internal-debug  .bool_gset:N = \g__unravel_default_internal_debug_bool ,
5890   max-action      .int_gset:N  = \g__unravel_default_max_action_int ,
5891   max-output      .int_gset:N  = \g__unravel_default_max_output_int ,
5892   max-input       .int_gset:N  = \g__unravel_default_max_input_int ,
5893   number-steps    .bool_gset:N = \g__unravel_default_number_steps_bool ,
5894   online          .int_gset:N  = \g__unravel_default_online_int ,
5895   prompt-input    .code:n
5896   = \__unravel_prompt_input:Nn \g__unravel_default_prompt_input_clist {#1} ,
5897   trace-assigns   .bool_gset:N = \g__unravel_default_trace_assigns_bool ,
5898   trace-expansion .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
5899   trace-other     .bool_gset:N = \g__unravel_default_trace_other_bool ,
5900   welcome-message .bool_gset:N = \g__unravel_default_welcome_message_bool ,
5901 }
5902 \keys_define:nn { unravel }
5903 {
5904   explicit-prompt .bool_gset:N = \g__unravel_explicit_prompt_bool ,
5905   internal-debug  .bool_gset:N = \g__unravel_internal_debug_bool ,
5906   max-action      .int_gset:N  = \g__unravel_max_action_int ,
5907   max-output      .int_gset:N  = \g__unravel_max_output_int ,
5908   max-input       .int_gset:N  = \g__unravel_max_input_int ,
5909   number-steps    .bool_gset:N = \g__unravel_number_steps_bool ,
5910   online          .int_gset:N  = \g__unravel_online_int ,
5911   prompt-input    .code:n
5912   = \__unravel_prompt_input:Nn \g__unravel_prompt_input_clist {#1} ,
5913   trace-assigns   .bool_gset:N = \g__unravel_trace_assigns_bool ,
5914   trace-expansion .bool_gset:N = \g__unravel_trace_expansion_bool ,
5915   trace-other     .bool_gset:N = \g__unravel_trace_other_bool ,
5916   welcome-message .bool_gset:N = \g__unravel_welcome_message_bool ,
5917 }
```

The `machine` and `trace` options are somewhat special so it is clearer to define them separately. The code is identical for `unravel/defaults` and `unravel` keys. To be sure of which options are set, use `.meta:nn` and give the path explicitly.

```
5918 \tl_map_inline:nn { { /defaults } { } }
5919 {
5920   \keys_define:nn { unravel #1 }
5921   {
5922     machine .meta:nn =
5923     { unravel #1 }
5924     {
5925       explicit-prompt = false ,
5926       internal-debug   = false ,
5927       max-action       = \c_max_int ,
5928       max-output       = \c_max_int ,
5929       max-input        = \c_max_int ,
```

```

5930         number-steps    = false ,
5931         welcome-message = false ,
5932     } ,
5933     mute                .meta:nn =
5934     { unravel #1 }
5935     {
5936         trace-assigns = false ,
5937         trace-expansion = false ,
5938         trace-other = false ,
5939         welcome-message = false ,
5940         online = -1 ,
5941     }
5942 }
5943 }

```

2.16 Main command

\unravel Simply call an underlying code-level command.

```
5944 \NewDocumentCommand \unravel { 0 { } +m } { \unravel:nn {#1} {#2} }
```

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

\unravelsetup Simply call an underlying code-level command.

```
5945 \NewDocumentCommand \unravelsetup { m } { \unravel_setup:n {#1} }
```

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

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

```

5946 \cs_new_protected:Npn \unravel_setup:n #1
5947 {
5948     \keys_set:nn { unravel/defaults } {#1}
5949     \keys_set:nn { unravel } {#1}
5950 }

```

(End definition for \unravel_setup:n. This function is documented on page 3.)

\unravel:nn The command starts with `__unravel_unravel_marker:` to detect nesting of `\unravel`

`__unravel:nn` in `\unravel` and avoid re-initializing important variables. Initialize and setup keys.

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

```

5951 \cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
5952 \cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
5953 \cs_new_protected:Npn \__unravel:nn #1#2
5954 {
5955     \__unravel_init_key_vars:
5956     \keys_set:nn { unravel } {#1}
5957     \__unravel_init_vars:
5958     \__unravel_input_gset:n {#2}
5959     \__unravel_print_welcome:
5960     \__unravel_main_loop:
5961     \__unravel_exit_point:

```

```

5962     \__unravel_print_outcome:
5963     \__unravel_final_test:
5964     \__unravel_exit_point:
5965   }
5966 \cs_new_protected:Npn \unravel_get:nnN #1#2#3
5967 {
5968     \unravel:nn {#1} {#2}
5969     \tl_set:Nx #3 { \gtl_left_tl:N \g__unravel_output_gtl }
5970 }

```

(End definition for `\unravel:nn`, `__unravel:nn`, and `__unravel_unravel_marker:.` This function is documented on page 2.)

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

```

5971 \cs_new_protected:Npn \__unravel_init_key_vars:
5972 {
5973     \sys_if_engine_luatex:T { \tl_gset:No \g__unravel_lastnamedcs_tl { \tex_lastnamedcs:D }
5974     \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bo
5975     \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
5976     \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
5977     \int_gset_eq:NN \g__unravel_online_int \g__unravel_default_online_int
5978     \clist_gset_eq:NN \g__unravel_prompt_input_clist \g__unravel_default_prompt_input_clist
5979     \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
5980     \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
5981     \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
5982     \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
5983     \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
5984     \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
5985     \int_gset_eq:NN \g__unravel_max_input_int \g__unravel_default_max_input_int
5986     \int_gzero:N \g__unravel_nonstop_int
5987 }

```

(End definition for `__unravel_init_key_vars:.`)

`__unravel_init_vars:` Give initial values to variables used during the processing. These have no reason to be modified by the user: neither directly nor through keys.

```

5988 \cs_new_protected:Npn \__unravel_init_vars:
5989 {
5990     \seq_gclear:N \g__unravel_prev_input_seq
5991     \gtl_gclear:N \g__unravel_output_gtl
5992     \int_gzero:N \g__unravel_step_int
5993     \tl_gclear:N \g__unravel_if_limit_tl
5994     \int_gzero:N \g__unravel_if_limit_int
5995     \int_gzero:N \g__unravel_if_depth_int
5996     \gtl_gclear:N \g__unravel_after_assignment_gtl
5997     \bool_gset_true:N \g__unravel_set_box_allowed_bool
5998     \bool_gset_false:N \g__unravel_name_in_progress_bool
5999     \gtl_clear:N \l__unravel_after_group_gtl
6000 }

```

(End definition for `__unravel_init_vars:.`)

`__unravel_main_loop:` Loop forever, getting the next token (with expansion) and performing the corresponding command. We use `__unravel_get_x_next_or_done:`, which is basically `__unravel_get_x_next:` but with a different behaviour when there are no more tokens: running out

of tokens here is a successful exit of `\unravel`. Note that we cannot put the logic into `__unravel_main_loop:` because `__unravel_expand_do:N` suppresses the loop when a token is marked with `\notexpanded:`, and we don't want that to suppress the main loop, only the expansion loop.

```

6001 \cs_new_protected:Npn \__unravel_get_x_next_or_done:
6002 {
6003   \__unravel_input_if_empty:TF { \__unravel_exit:w } { }
6004   \__unravel_get_next:
6005   \__unravel_token_if_expandable:NT \l__unravel_head_token
6006   { \__unravel_expand_do:N \__unravel_get_x_next_or_done: }
6007 }
6008 \cs_new_protected:Npn \__unravel_main_loop:
6009 {
6010   \__unravel_get_x_next_or_done:
6011   \__unravel_set_cmd:
6012   \__unravel_do_step:
6013   \__unravel_main_loop:
6014 }

```

(End definition for `__unravel_main_loop:` and `__unravel_get_x_next_or_done:`.)

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

```

6015 \cs_new_protected:Npn \__unravel_do_step:
6016 {
6017   \__unravel_set_action_text:
6018   \bool_if:NT \g__unravel_internal_debug_bool
6019   { \__unravel_exp_args:Nx \iow_term:n { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int }
6020     \cs_if_exist_use:cF
6021     { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
6022     { \__unravel_error:nxxxx { internal } { unknown-command } { } { } { } }
6023 }

```

(End definition for `__unravel_do_step:`.)

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

```

\__unravel_final_bad:
6024 \cs_new_protected:Npn \__unravel_final_test:
6025 {
6026   \__unravel_input_if_empty:TF
6027   {
6028     \seq_if_empty:NTF \g__unravel_prev_input_seq
6029     {
6030       \tl_if_empty:NTF \g__unravel_if_limit_tl
6031       { \int_compare:nNnF \g__unravel_if_limit_int = 0 { \__unravel_final_bad: } }
6032       { \__unravel_final_conditionals: }
6033     }
6034     { \__unravel_final_bad: }
6035   }
6036   { \__unravel_final_bad: }
6037   \__unravel_final_after_assignment:
6038 }
6039 \cs_new_protected:Npn \__unravel_final_bad:
6040 {
6041   \__unravel_error:nnnnn { internal }

```

```

6042     { the-last-unravel-finished-badly } { } { } { }
6043   }
6044   \cs_new_protected:Npn \__unravel_final_conditionals:
6045     {
6046     \group_begin:
6047     \msg_warning:nxx { unravel } { dangling-conditionals }
6048     { \tl_count:N \g__unravel_if_limit_tl }
6049     \group_end:
6050     \tl_greverse:N \g__unravel_if_limit_tl
6051     \tl_gput_right:NV \g__unravel_if_limit_tl \g__unravel_if_limit_int
6052     \tl_gset:Nx \g__unravel_if_limit_tl { \tl_tail:N \g__unravel_if_limit_tl } % remove the
6053     \prg_replicate:nn { \tl_count:N \g__unravel_if_limit_tl } { \fi: }
6054     \tl_map_function:NN \g__unravel_if_limit_tl \__unravel_final_cond_aux:n
6055     }
6056   \cs_new:Npn \__unravel_final_cond_aux:n #1
6057     {
6058     \int_case:nnF {#1}
6059     {
6060     { 2 } { \if_false: \else: }
6061     { 3 } { \if_true: }
6062     { 4 } { \if_case:w 0 ~ }
6063     }
6064     { \__unravel_final_bad: }
6065     }

```

(End definition for __unravel_final_test: and __unravel_final_bad:.)

```

\__unravel_final_after_assignment: Salvage any remaining \afterassignment token.
6066 \cs_new_protected:Npn \__unravel_final_after_assignment:
6067   {
6068   \gtl_if_empty:NF \g__unravel_after_assignment_gtl
6069   { \gtl_head_do:NN \g__unravel_after_assignment_gtl \tex_afterassignment:D }
6070   }

```

(End definition for __unravel_final_after_assignment:.)

2.17 Messages

```

6071 \msg_new:nnnn { unravel } { prev-input }
6072   { Internal~error:~unexpected~type~of~‘‘prev_input’’~entry. }
6073   {
6074   Found~type~#2~instead~of~#1~to~assign~to~variable~#3.~Contents:\\
6075   \iow_indent:n {#4}
6076   }
6077 \msg_new:nnn { unravel } { unknown-primitive }
6078   { Internal~error:~the~primitive~‘#1’~is~not~known. }
6079 \msg_new:nnn { unravel } { extra-fi-or-else }
6080   { Extra~fi,~or,~or~else. }
6081 \msg_new:nnn { unravel } { missing-dollar }
6082   { Missing~dollar~inserted. }
6083 \msg_new:nnn { unravel } { unknown-expandable }
6084   { Internal~error:~the~expandable~command~‘#1’~is~not~known. }
6085 \msg_new:nnn { unravel } { missing-font-id }
6086   { Missing~font~identifier.~\iow_char:N\mullfont~inserted. }
6087 \msg_new:nnn { unravel } { missing-rparen }

```



```

6088 { Missing~right~parenthesis~inserted~for~expression. }
6089 \msg_new:nnn { unravel } { missing-cs }
6090 { Missing~control~sequence.~\iow_char:N\\inaccessible~inserted. }
6091 \msg_new:nnn { unravel } { missing-box }
6092 { Missing~box~inserted. }
6093 \msg_new:nnn { unravel } { missing-to }
6094 { Missing~keyword~'to'~inserted. }
6095 \msg_new:nnn { unravel } { improper-leaders }
6096 { Leaders~not~followed~by~proper~glue. }
6097 \msg_new:nnn { unravel } { extra-close }
6098 { Extra~right~brace~or~\iow_char:N\\endgroup. }
6099 \msg_new:nnn { unravel } { off-save }
6100 { Something~is~wrong~with~groups. }
6101 \msg_new:nnn { unravel } { hrule-bad-mode }
6102 { \iow_char\\hrule~used~in~wrong~mode. }
6103 \msg_new:nnn { unravel } { invalid-mode }
6104 { Invalid~mode~for~this~command. }
6105 \msg_new:nnn { unravel } { color-stack-action-missing }
6106 { Missing~color~stack~action. }
6107 \msg_new:nnn { unravel } { action-type-missing }
6108 { Missing~action~type. }
6109 \msg_new:nnn { unravel } { identifier-type-missing }
6110 { Missing~identifier~type. }
6111 \msg_new:nnn { unravel } { destination-type-missing }
6112 { Missing~destination~type. }
6113 \msg_new:nnn { unravel } { erroneous-prefixes }
6114 { Prefixes~applied~to~non~assignment~command. }
6115 \msg_new:nnn { unravel } { improper-setbox }
6116 { \iow_char:N\\setbox~while~fetching~base~of~an~accent. }
6117 \msg_new:nnn { unravel } { after-advance }
6118 {
6119     Missing~register~after~\iow_char:N\\advance,~
6120     \iow_char:N\\multiply,~or~\iow_char:N\\divide.
6121 }
6122 \msg_new:nnn { unravel } { bad-unless }
6123 { \iow_char:N\\unless~not~followed~by~conditional. }
6124 \msg_new:nnn { unravel } { runaway-if }
6125 { Runaway~\iow_char:N\\if...~Exiting~\iow_char:N\\unravel }
6126 \msg_new:nnn { unravel } { runaway-macro-parameter }
6127 {
6128     Runaway~macro~parameter~\# #2~after \\ \\
6129     \iow_indent:n {#1}
6130 }
6131 \msg_new:nnn { unravel } { runaway-text }
6132 { Runaway~braced~argument~for~TeX~primitive.~Exiting~\iow_char:N\\unravel }
6133 \msg_new:nnn { unravel } { extra-or }
6134 { Extra~\iow_char:N\\or. }
6135 \msg_new:nnn { unravel } { missing-equals }
6136 { Missing~equals~for~\iow_char:N\\ifnum~or~\iow_char:N\\ifdim. }
6137 \msg_new:nnn { unravel } { internal }
6138 { Internal~error:~'#1'.~\ Please~report. }
6139 \msg_new:nnn { unravel } { not-implemented }
6140 { The~following~feature~is~not~implemented:~'#1'. }
6141 \msg_new:nnn { unravel } { endinput-ignored }

```

```

6142 { The~primitive~\iow_char:N\endinput~was~ignored. }
6143 \msg_new:nnn { unravel } { missing-something }
6144 { Something~is~missing,~sorry! }
6145 \msg_new:nnn { unravel } { nested-unravel }
6146 { The~\iow_char:N\unravel~command~may~not~be~nested. }
6147 \msg_new:nnnn { unravel } { tex-error }
6148 { TeX~sees~"#1"~and~throws~an~error:\\\\ \iow_indent:n {#2} }
6149 {
6150   \tl_if_empty:nTF {#3}
6151     { TeX~provides~no~further~help~for~this~error. }
6152     { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
6153 }
6154 \msg_new:nnnn { unravel } { tex-fatal }
6155 { TeX~sees~"#1"~and~throws~a~fatal~error:\\\\ \iow_indent:n {#2} }
6156 {
6157   \tl_if_empty:nTF {#3}
6158     { TeX~provides~no~further~help~for~this~error. }
6159     { TeX's~advice~is:\\\\ \iow_indent:n {#3} }
6160 }
6161 \msg_new:nnnn { unravel } { runaway-unravel }
6162 { Runaway~\iow_char:N\unravel,~so~\iow_char:N\relax~inserted. }
6163 {
6164   Some~TeX~command~expects~input~beyond~the~end~of~
6165   the~argument~of~\iow_char:N\unravel.
6166 }
6167 \msg_new:nnn { unravel } { dangling-conditionals }
6168 { Attempting~to~issue~#1~dangling~conditionals. }

Some error messages from TEX itself.
6169 \__unravel_tex_msg_new:nnn { forbidden-case }
6170 {
6171   You~can't~use~'\exp_after:wN \token_to_str:N \l__unravel_head_tl'~in~
6172   \mode_if_vertical:TF { vertical }
6173   {
6174     \mode_if_horizontal:TF { horizontal }
6175     { \mode_if_math:TF { math } { no } }
6176   } ~ mode.
6177 }
6178 {
6179   Sorry,~but~I'm~not~programmed~to~handle~this~case;~
6180   I'll~just~pretend~that~you~didn't~ask~for~it.~
6181   If~you're~in~the~wrong~mode,~you~might~be~able~to~
6182   return~to~the~right~one~by~typing~'\iow_char:N\}'~or~
6183   '\iow_char:N\$}'~or~'\iow_char:N\par'.
6184 }
6185 \__unravel_tex_msg_new:nnn { incompatible-mag }
6186 {
6187   Incompatible~magnification~
6188   ( \int_to_arabic:n { \__unravel_mag: } );~
6189   the~previous~value~will~be~retained
6190 }
6191 {
6192   I~can~handle~only~one~magnification~ratio~per~job.~So~I've~
6193   reverted~to~the~magnification~you~used~earlier~on~this~run.
6194 }

```

```

6195 \_unravel_tex_msg_new:nnn { illegal-mag }
6196 {
6197   Illegal~magnification~has~been~changed~to~1000~
6198   ( \int_to_arabic:n { \_unravel_mag: } )
6199 }
6200 { The~magnification~ratio~must~be~between~1~and~32768. }
6201 \_unravel_tex_msg_new:nnn { missing-number }
6202 { Missing~number,~treated~as~zero }
6203 {
6204   A~number~should~have~been~here;~I~inserted~'0'.~
6205   If~you~can't~figure~out~why~I~needed~to~see~a~number,~
6206   look~up~'weird~error'~in~the~index~to~The~TeXbook.
6207 }
6208 \_unravel_tex_msg_new:nnn { the-cannot }
6209 { You~can't~use~'\tl_to_str:N\l\_unravel_head_tl'~after~\iow_char:N\the }
6210 { I'm~forgetting~what~you~said~and~using~zero~instead. }
6211 \_unravel_tex_msg_new:nnn { incompatible-units }
6212 { Incompatible~glue~units }
6213 { I'm~going~to~assume~that~1mu=1pt~when~they're~mixed. }
6214 \_unravel_tex_msg_new:nnn { missing-mu }
6215 { Illegal~unit~of~measure~(mu~inserted) }
6216 {
6217   The~unit~of~measurement~in~math~glue~must~be~mu.~
6218   To~recover~gracefully~from~this~error,~it's~best~to~
6219   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
6220   two~letters.~(See~Chapter~27~of~The~TeXbook.)
6221 }
6222 \_unravel_tex_msg_new:nnn { missing-pt }
6223 { Illegal~unit~of~measure~(pt~inserted) }
6224 {
6225   Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
6226   cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
6227   I'll~assume~that~you~meant~to~say~pt,~for~printer's~points.~
6228   To~recover~gracefully~from~this~error,~it's~best~to~
6229   delete~the~erroneous~units;~e.g.,~type~'2'~to~delete~
6230   two~letters.~(See~Chapter~27~of~The~TeXbook.)
6231 }
6232 \_unravel_tex_msg_new:nnn { missing-lbrace }
6233 { Missing~\iow_char:N\{~inserted }
6234 {
6235   A~left~brace~was~mandatory~here,~so~I've~put~one~in.~
6236   You~might~want~to~delete~and/or~insert~some~corrections~
6237   so~that~I~will~find~a~matching~right~brace~soon.~
6238   (If~you're~confused~by~all~this,~try~typing~'\iow_char:N\}'~now.)
6239 }
6240 \_unravel_tex_msg_new:nnn { extra-endcsname }
6241 { Extra~\token_to_str:c{endcsname} }
6242 { I'm~ignoring~this,~since~I~wasn't~doing~a~\token_to_str:c{csname}. }
6243 \_unravel_tex_msg_new:nnn { missing-endcsname }
6244 { Missing~\token_to_str:c{endcsname}~inserted }
6245 {
6246   The~control~sequence~marked~<to~be~read~again>~should~
6247   not~appear~between~\token_to_str:c{csname}~and~
6248   \token_to_str:c{endcsname}.

```

```

6249 }
6250 \__unravel_tex_msg_new:nnn { missing-delim }
6251 { Missing-delimiter~(.~inserted) }
6252 {
6253   I~was~expecting~to~see~something~like~'('~or~'\token_to_str:N\{'~or~
6254   '\token_to_str:N\}'~here.~If~you~typed,~e.g.,~
6255   '\{'~instead~of~'\token_to_str:N\{'~you~
6256   should~probably~delete~the~'\{'~by~typing~'1'~now,~so~that~
6257   braces~don't~get~unbalanced.~Otherwise~just~proceed.~
6258   Acceptable~delimiters~are~characters~whose~\token_to_str:c{delcode}~is~
6259   nonnegative,~or~you~can~use~'\token_to_str:c{delimiter}~<delimiter~code>'.
6260 }

```

Fatal T_EX error messages.

```

6261 \__unravel_tex_msg_new:nnn { cannot-read }
6262 { ***~(cannot~\iow_char:N\read~from~terminal~in~nonstop~modes) }
6263 { }
6264 \__unravel_tex_msg_new:nnn { file-error }
6265 { ***~(job~aborted,~file~error~in~nonstop~mode) }
6266 { }
6267 \__unravel_tex_msg_new:nnn { interwoven-preambles }
6268 { (interwoven~alignment~preambles~are~not~allowed) }
6269 { }

```

Restore catcodes to their original values.

```

6270 \__unravel_setup_restore:
6271 </package>

```