The **unravel** package: watching TeX digest tokens*

Bruno Le Floch

2024/01/05

Contents

---

*This file has version number 0.3c, last revised 2024/01/05.

1 unravel documentation

The aim of this LaTeX package is to help debug complicated macros. This is done by letting the user step through the execution of some TeX 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 TeX in a terminal.

1.1 Commands

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

This command shows in the terminal the steps performed by TeX 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` `\unravelsetup {⟨options⟩}`

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

`\unravel:nn` `\unravel:nn {⟨options⟩} {⟨code⟩}`

See `\unravel`.

`\unravel_get:nnN` `\unravel_get:nnN {⟨options⟩} {⟨code⟩} ⟨tl var⟩`

Performs `\unravel:nn` with the ⟨*options*⟩ and ⟨*code*⟩ then saves the output into the ⟨*tl var*⟩. The option `mute` is useful in this case.

`\unravel_setup:n` `\unravel_setup:n {⟨options⟩}`

See `\unravelsetup`.

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, XETEX, 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 pdfTeX, 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 (default 1) 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.

`output-file` Name of a file where the output is written instead of the log file. Setting this option also sets `online` to zero. If `online` is further modified to be positive then the output is written to the terminal as well (hence inevitably to the log file), while if it is made negative then the output is written nowhere.

`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`, `\crcr`, `&`), some math mode primitives, and `\pdfprimitive`, as well as many primitives specific to engines other than pdfTeX. 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.

```
⟨*package⟩
⟨@@=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.

```
\begingroup\let\c\catcode\fam32\c\fam10\advance\fam5\c\fam14\c45 12 %
\c54 12\c55 12\c56 12\c57 12\c58 11\c95 11\c104 11\c106 11\c107 11 %
\c113 11\c115 11\c119 11\c120 11\c121 11\c122 11\endlinechar-1 %
\expandafter\ifx\csname unravel\endcsname\relax
\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.

```
\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.

```
\expandafter\ifx\csname numexpr\endcsname\relax
\errmessage{unravel requires \numexpr from eTeX}
\endinput\expandafter\endgroup\fi
\expandafter\ifx\csname protected\endcsname\relax
\errmessage{unravel requires \protected from eTeX}
\endinput\expandafter\endgroup\fi
```

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

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

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

```
\catcode 32 = 9 \relax
\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.

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

(*End of 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.

```
\protected \gdef \__unravel_setup_restore:
  {
    \l__unravel_setup_restore_tl
    \def \l__unravel_setup_restore_tl { }
  }
```

(*End of definition for* `\__unravel_setup_restore:`.)

`\__unravel_setup_save:`
`\__unravel_setup_save_aux:n`

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

```
\protected \gdef \__unravel_setup_save:
  {
    \edef \l__unravel_setup_restore_tl
      {
        \__unravel_setup_save_aux:w 0 =
        \endlinechar = \the \endlinechar
        \escapechar = \the \escapechar
        \newlinechar = \the \newlinechar
        \relax
      }
  }
\long \gdef \__unravel_setup_save_aux:w #1 =
  {
    \catcode #1 = \the \catcode #1 ~
    \ifnum 255 > #1 ~
```

```
      \expandafter \__unravel_setup_save_aux:w
      \the \numexpr #1 + 1 \expandafter =
    \fi
  }
```

(*End of definition for* `\__unravel_setup_save:` *and* `\__unravel_setup_save_aux:n`.)

`\__unravel_setup_catcodes:nnn`

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

```
\protected \long \gdef \__unravel_setup_catcodes:nnn #1 #2 #3
  {
    \ifnum #1 > #2 ~ \else
      \catcode #1 = #3 ~
      \expandafter \__unravel_setup_catcodes:nnn \expandafter
        { \the \numexpr #1 + 1 } {#2} {#3}
    \fi
  }
```

(*End of definition for* `\__unravel_setup_catcodes:nnn`.)

`\__unravel_setup_latexe:`

This saves the catcodes and related parameters, then sets them to the value they normally have in a LaTeX 2$_{\varepsilon}$ package (in particular, `@` is a letter).

```
\protected \gdef \__unravel_setup_latexe:
  {
    \__unravel_setup_save:
    \__unravel_setup_catcodes:nnn {0} {8} {15}
    \catcode 9 = 10 ~
    \catcode 10 = 12 ~
    \catcode 11 = 15 ~
    \catcode 12 = 13 ~
    \catcode 13 = 5 ~
    \__unravel_setup_catcodes:nnn {14} {31} {15}
    \catcode 32 = 10 ~
    \catcode 33 = 12 ~
    \catcode 34 = 12 ~
    \catcode 35 = 6 ~
    \catcode 36 = 3 ~
    \catcode 37 = 14 ~
    \catcode 38 = 4 ~
    \__unravel_setup_catcodes:nnn {39} {63} {12}
    \__unravel_setup_catcodes:nnn {64} {90} {11}
    \catcode 91 = 12 ~
    \catcode 92 = 0 ~
    \catcode 93 = 12 ~
    \catcode 94 = 7 ~
    \catcode 95 = 8 ~
    \catcode 96 = 12 ~
    \__unravel_setup_catcodes:nnn {97} {122} {11}
    \catcode 123 = 1 ~
    \catcode 124 = 12 ~
    \catcode 125 = 2 ~
    \catcode 126 = 13 ~
    \catcode 127 = 15 ~
    \__unravel_setup_catcodes:nnn {128} {255} {12}
    \endlinechar = 13 ~
    \escapechar = 92 ~
    \newlinechar = 10 ~
  }
```

(*End of definition for* `\__unravel_setup_latexe:`.)

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

```
\protected \gdef \__unravel_setup_unravel:
  {
    \__unravel_setup_save:
    \__unravel_setup_catcodes:nnn {0} {8} {15}
    \catcode 9 = 9 ~
    \catcode 10 = 12 ~
    \catcode 11 = 15 ~
    \catcode 12 = 13 ~
    \catcode 13 = 5 ~
    \__unravel_setup_catcodes:nnn {14} {31} {15}
    \catcode 32 = 9 ~
    \catcode 33 = 12 ~
    \catcode 34 = 12 ~
    \catcode 35 = 6 ~
    \catcode 36 = 3 ~
    \catcode 37 = 14 ~
    \catcode 38 = 4 ~
    \__unravel_setup_catcodes:nnn {39} {57} {12}
    \catcode 58 = 11 ~
    \__unravel_setup_catcodes:nnn {59} {64} {12}
    \__unravel_setup_catcodes:nnn {65} {90} {11}
    \catcode 91 = 12 ~
    \catcode 92 = 0 ~
    \catcode 93 = 12 ~
    \catcode 94 = 7 ~
    \catcode 95 = 11 ~
    \catcode 96 = 12 ~
    \__unravel_setup_catcodes:nnn {97} {122} {11}
    \catcode 123 = 1 ~
    \catcode 124 = 12 ~
    \catcode 125 = 2 ~
    \catcode 126 = 10 ~
    \catcode 127 = 15 ~
    \__unravel_setup_catcodes:nnn {128} {255} {12}
    \escapechar  = 92 ~
    \endlinechar  = 32 ~
    \newlinechar  = 10 ~
  }
```

(*End of definition for* `\__unravel_setup_unravel:`.)

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

```
\expandafter \endgroup \__unravel_setup_latexe:
```

Load the gtl dependency.

```
\RequirePackage{gtl}[2024/01/04]
```

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.

```
\csname use:n\endcsname
  {%
    \csname __unravel_setup_restore:\endcsname
    \ProvidesExplPackage
      {unravel} {2024/01/05} {0.3c} {Watching TeX digest tokens}%
    \csname __unravel_setup_unravel:\endcsname
  }%
```

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.

```
\cs_gset:Npn \group_align_safe_begin:
  { \exp:w \if_false: { \fi: -‘} \exp_stop_f: }
\cs_gset:Npn \group_align_safe_end:
  { \if_int_compare:w ‘{ = \c_zero_int } \fi: }
```

2.1.2 Renamed primitives

`\__unravel_currentgrouptype:`
`\__unravel_everyeof:w`
`\__unravel_everypar:w`
`\__unravel_set_escapechar:n`
`\__unravel_nullfont:`
`\__unravel_hbox:w`
`\__unravel_the:w`

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).

```
\cs_new_eq:NN \__unravel_currentgrouptype:     \tex_currentgrouptype:D
\cs_new_protected:Npn \__unravel_set_escapechar:n
  { \int_set:Nn \tex_escapechar:D }
\cs_new_eq:NN \__unravel_everyeof:w            \tex_everyeof:D
\cs_new_eq:NN \__unravel_everypar:w            \tex_everypar:D
\cs_new_eq:NN \__unravel_hbox:w                \tex_hbox:D
\cs_new_eq:NN \__unravel_mag:                  \tex_mag:D
\cs_new_eq:NN \__unravel_nullfont:             \tex_nullfont:D
\cs_new_eq:NN \__unravel_the:w                 \tex_the:D
\cs_new_eq:NN \__unravel_number:w              \tex_number:D
```

(*End of 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.

```
\exp_after:wN \cs_new_eq:NN
  \exp_after:wN \__unravel_special_relax:
  \exp_not:N \__unravel_special_relax:
```

(*End of 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.

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

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

2.1.3 Variants

Variants that we need.

```
\cs_generate_variant:Nn \seq_push:Nn { Nf }
\cs_generate_variant:Nn \str_head:n { f }
\cs_generate_variant:Nn \tl_to_str:n { o }
\cs_generate_variant:Nn \tl_if_eq:nnTF { o }
\cs_generate_variant:Nn \tl_if_head_eq_meaning:nNT { V }
\cs_generate_variant:Nn \tl_if_head_eq_meaning:nNTF { V }
\cs_generate_variant:Nn \tl_if_single_token:nT { V }
\cs_generate_variant:Nn \gtl_gput_right:Nn { NV }
\cs_generate_variant:Nn \gtl_if_empty:NTF { c }
\cs_generate_variant:Nn \gtl_if_tl:NT { c }
\cs_generate_variant:Nn \gtl_to_str:N { c }
\cs_generate_variant:Nn \gtl_gpop_left:NN { c }
\cs_generate_variant:Nn \gtl_get_left:NN { c }
\cs_generate_variant:Nn \gtl_gset:Nn { c }
\cs_generate_variant:Nn \gtl_gconcat:NNN { ccc , cNc }
\cs_generate_variant:Nn \gtl_gclear:N { c }
\cs_generate_variant:Nn \gtl_gclear_new:N { c }
\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).

```
\cs_new_protected:Npn \__unravel_tl_if_in:ooTF #1#2#3#4
  {
    \group_begin:
    \exp_args:Noo \tl_if_in:nnTF {#1} {#2}
      { \group_end: #3 } { \group_end: #4 }
  }
```

(*End of definition for* `\__unravel_tl_if_in:ooTF`.)

2.1.4 Miscellanous helpers

`\__unravel_tmp:w`

Temporary function used to define other functions.

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

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

`\__unravel_file_get:nN`
`\__unravel_file_get_aux:wN`

```
\cs_set_protected:Npn \__unravel_tmp:w #1
  {
    \cs_new_protected:Npn \__unravel_file_get:nN ##1##2
      {
        \group_begin:
          \__unravel_everyeof:w { #1 ##2 }
          \exp_after:wN \__unravel_file_get_aux:wN
          \exp_after:wN \prg_do_nothing:
            \tex_input:D ##1 \scan_stop:
      }
    \cs_new_protected:Npn \__unravel_file_get_aux:wN ##1 #1 ##2
      {
        \group_end:
        \tl_set:Ne ##2
          { \exp_not:o {##1} \exp_not:V \__unravel_everyeof:w }
      }
  }
\exp_args:No \__unravel_tmp:w { \token_to_str:N : : }
```

(*End of 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 ⟨*number*⟩ within the whole `\read` ⟨*number*⟩ `to` ⟨*cs*⟩ 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.

```
\cs_new:Npn \__unravel_tl_first_int:N #1
  {
    \int_eval:n
      {
        \exp_after:wN \__unravel_tl_first_int_aux:Nn
        \exp_after:wN \__unravel_tl_first_int_aux:Nn
        #1 ? 0 ? \q_stop
      }
  }
\cs_new:Npn \__unravel_tl_first_int_aux:Nn #1#2
  {
    \tl_if_single:nT {#2}
      {
        \token_if_eq_catcode:NNT + #2
          {
            \if_int_compare:w 1 < 1 #2 \exp_stop_f:
              #2
              \exp_after:wN \use_i_ii:nnn
              \exp_after:wN \__unravel_tl_first_int_aux:Nn
              \exp_after:wN \use_none_delimit_by_q_stop:w
            \fi:
          }
      }
    #1
  }
```

(*End of definition for* `\__unravel_tl_first_int:N` *and* `\__unravel_tl_first_int_aux:Nn`*.*)

`\__unravel_use_ii_i:nn`

```
\cs_new:Npn \__unravel_use_ii_i:nn #1#2 { #2 #1 }
```

(*End of definition for* `\__unravel_use_ii_i:nn`*.*)

`\__unravel_prompt_input:Nn`
`\__unravel_prompt_input:w`
`\__unravel_prompt_input_aux:w`
`avel_use_none_delimit_by_q_recursion_tail:w`
`\q__unravel_recursion_tail`

```
\cs_new_protected:Npn \__unravel_prompt_input:Nn #1#2
  {
    \clist_gset:Ne #1
      { \__unravel_prompt_input:w \prg_do_nothing: #2 , \q__unravel_recursion_tail , }
  }
\cs_new:Npn \__unravel_prompt_input:w #1 ,
  {
    \tl_trim_spaces_apply:oN {#1} \__unravel_use_ii_i:nn
    \__unravel_prompt_input_aux:w ,
  }
\cs_new:Npn \__unravel_prompt_input_aux:w #1 ,
  {
    \__unravel_use_none_delimit_by_q_recursion_tail:w #1
      \use_none:nnnnn \q__unravel_recursion_tail
    { \tl_to_str:n {#1} } ,
    \__unravel_prompt_input:w \prg_do_nothing:
  }
\cs_new:Npn \__unravel_use_none_delimit_by_q_recursion_tail:w
    #1 \q__unravel_recursion_tail { }
\quark_new:N \q__unravel_recursion_tail
```

(*End of definition for* `\__unravel_prompt_input:Nn` *and others.*)

2.1.5 String helpers

`\__unravel_strip_escape:w`
`\__unravel_strip_escape_aux:N`
`\__unravel_strip_escape_aux:w`

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

```
\cs_new:Npn \__unravel_strip_escape:w
  {
    \tex_romannumeral:D
      \if_charcode:w \token_to_str:N \ \__unravel_strip_escape_aux:w \fi:
      \__unravel_strip_escape_aux:N
  }
\cs_new:Npn \__unravel_strip_escape_aux:N #1 { \c_zero_int }
\cs_new:Npn \__unravel_strip_escape_aux:w #1#2
  { - \__unravel_number:w #1 \c_zero_int }
```

(*End of 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.

```
\cs_new:Npn \__unravel_to_str:Nn #1
  {
    \if_meaning:w T #1
      \exp_after:wN \tl_to_str:n
    \else:
      \exp_after:wN \gtl_to_str:n
    \fi:
  }
```

(*End of 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.

```
\cs_new:Npn \__unravel_str_truncate_left:nn #1#2
  {
    \exp_args:Nf \__unravel_str_truncate_left_aux:nnn
      { \str_count:n {#1} } {#1} {#2}
  }
\cs_new:Npn \__unravel_str_truncate_left_aux:nnn #1#2#3
  {
    \int_compare:nNnTF {#1} > {#3}
      {
        ( \int_eval:n { #1 - #3 + 25 } ~ more~chars ) ~
        \str_range:nnn {#2} { #1 - #3 + 26 } {#1}
      }
      { \tl_to_str:n {#2} }
  }
```

(*End of 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.

```
\cs_new:Npn \__unravel_str_truncate_right:nn #1#2
  {
    \exp_args:Nf \__unravel_str_truncate_right_aux:nnn
      { \str_count:n {#1} } {#1} {#2}
  }
\cs_new:Npn \__unravel_str_truncate_right_aux:nnn #1#2#3
  {
    \int_compare:nNnTF {#1} > {#3}
      {
        \str_range:nnn {#2} { 1 } { #3 - 25 } ~
        ( \int_eval:n { #1 - #3 + 25 } ~ more~chars )
      }
      { \tl_to_str:n {#2} }
  }
```

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

2.1.6 Helpers for control flow

\__unravel_exit:w
\__unravel_exit_hard:w
\__unravel_exit_point:

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

```
\cs_new_eq:NN \__unravel_exit_point: \prg_do_nothing:
\cs_new:Npn \__unravel_exit:w #1 \__unravel_exit_point: { }
\cs_new:Npn \__unravel_exit_hard:w #1 \__unravel_exit_point: #2 \__unravel_exit_point: { }
```

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

\__unravel_break:w
\__unravel_break_point:

Useful to jump out of complicated conditionals.

```
\cs_new_eq:NN \__unravel_break_point: \prg_do_nothing:
\cs_new:Npn \__unravel_break:w #1 \__unravel_break_point: { }
```

(*End of 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).

```
\prg_new_conditional:Npnn \__unravel_cmd_if_internal: { TF }
  {
    \int_compare:nNnTF
      \l__unravel_head_cmd_int < { \__unravel_tex_use:n { min_internal } }
      { \prg_return_false: }
      {
        \int_compare:nNnTF
          \l__unravel_head_cmd_int
          > { \__unravel_tex_use:n { max_internal } }
          { \prg_return_false: }
          { \prg_return_true: }
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_active_do:nn #1#2
  {
    \group_begin:
    \char_set_active_eq:nN {#1} \scan_stop:
    \use:e
      {
        \group_end:
        \exp_not:n {#2} { \char_generate:nn {#1} { 13 } }
      }
  }
```

(*End of definition for* `\__unravel_active_do:nn`*.*)

\__unravel_token_to_char:N
\__unravel_meaning_to_char:n
\__unravel_meaning_to_char:o
\__unravel_meaning_to_char_auxi:w
\__unravel_meaning_to_char_auxii:w

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

```
\cs_new:Npn \__unravel_meaning_to_char:n #1
  { \__unravel_meaning_to_char_auxi:w #1 \q_mark ~ {} ~ \q_mark \q_stop }
\cs_new:Npn \__unravel_meaning_to_char_auxi:w #1 ~ #2 ~ #3 \q_mark #4 \q_stop
```

```
  { \__unravel_meaning_to_char_auxii:w #3 ~ #3 ~ \q_stop }
\cs_new:Npn \__unravel_meaning_to_char_auxii:w #1 ~ #2 ~ #3 \q_stop
  { \tl_if_empty:nTF {#2} { ~ } {#2} }
\cs_generate_variant:Nn \__unravel_meaning_to_char:n { o }
\cs_new:Npn \__unravel_token_to_char:N #1
  { \__unravel_meaning_to_char:o { \token_to_meaning:N #1 } }
```

(*End of definition for* `\__unravel_token_to_char:N` *and others.*)

__unravel_token_if_expandable_p:N
__unravel_token_if_expandable:N*TF*

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

```
\prg_new_conditional:Npnn \__unravel_token_if_expandable:N #1
  { p , T ,  F , TF }
  {
    \exp_after:wN \if_meaning:w \exp_not:N #1 #1
      \prg_return_false:
    \else:
      \prg_return_true:
    \fi:
  }
```

(*End of definition for* `\__unravel_token_if_expandable:NTF`*.*)

__unravel_token_if_protected_p:N
__unravel_token_if_protected:N*TF*

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

```
\prg_new_conditional:Npnn \__unravel_token_if_protected:N #1
  { p , T ,  F , TF }
  {
    \__unravel_token_if_expandable:NTF #1
      {
        \token_if_protected_macro:NTF #1
          { \prg_return_true: }
          {
            \token_if_protected_long_macro:NTF #1
              { \prg_return_true: }
              { \prg_return_false: }
          }
      }
      { \prg_return_true: }
  }
```

(*End of definition for* `\__unravel_token_if_protected:NTF`*.*)

__unravel_token_if_active_char:N*TF*

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.

```
\group_begin:
  \char_set_catcode_active:n { `Z }
  \prg_new_protected_conditional:Npnn \__unravel_token_if_active_char:N #1
    { TF }
    {
      \group_begin:
        \exp_args:Ne \char_set_lccode:nn
          { ` \exp_args:No \str_head:n { \token_to_str:N #1 } }
          { ` Z }
        \tex_lowercase:D { \tl_if_eq:nnTF {#1} } { Z }
          { \group_end: \prg_return_true: }
          { \group_end: \prg_return_false: }
    }
\group_end:
```

(*End of 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.

```
\group_begin:
  \char_set_catcode_active:n { `Z }
  \prg_new_protected_conditional:Npnn \__unravel_token_if_definable:N #1
    { TF }
    {
      \group_begin:
        \__unravel_set_escapechar:n { 92 }
        \tl_set:Ne \l__unravel_tmpa_tl
          { \exp_args:No \str_tail:n { \token_to_str:N #1 } }
        \tl_if_empty:NTF \l__unravel_tmpa_tl
          {
            \__unravel_token_if_active_char:NTF #1
              { \group_end: \prg_return_true: }
              { \group_end: \prg_return_false: }
          }
          { \group_end: \prg_return_true: }
    }
\group_end:
```

(*End of 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.

```
\prg_new_protected_conditional:Npnn \__unravel_gtl_if_head_is_definable:N #1
  { TF , F }
  {
    \gtl_if_single_token:NTF #1
      {
        \gtl_if_head_is_N_type:NTF #1
          {
            \gtl_head_do:NN #1 \__unravel_token_if_definable:NTF
              { \prg_return_true: }
              { \prg_return_false: }
          }
          { \prg_return_false: }
      }
      { \prg_return_false: }
  }
```

(*End of definition for* `\__unravel_gtl_if_head_is_definable:NTF`*.*)

2.1.8 Helpers for previous input

`\__unravel_prev_input_count:`
`\__unravel_prev_input_count_aux:n`
`\__unravel_prev_input_count_aux:Nn`

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

```
\cs_new:Npn \__unravel_prev_input_count:
  {
    \int_eval:n
      {
        0
        \seq_map_function:NN \g__unravel_prev_input_seq
          \__unravel_prev_input_count_aux:n
      }
  }
\cs_new:Npn \__unravel_prev_input_count_aux:n #1
  { \__unravel_prev_input_count_aux:Nn #1 }
\cs_new:Npn \__unravel_prev_input_count_aux:Nn #1#2
  {
    \if_meaning:w T #1
      \exp_after:wN \tl_if_empty:nF
    \else:
      \exp_after:wN \str_if_eq:onF \exp_after:wN \c_empty_gtl
    \fi:
    {#2} { + 1 }
  }
```

(*End of 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`
`\__unravel_prev_input_gpush_gtl:`
`\__unravel_prev_input_gpush_gtl:N`
`\__unravel_prev_input_gpush_aux:NN`

```
\cs_new_protected:Npn \__unravel_prev_input_gpush:
  { \seq_gput_right:Nn \g__unravel_prev_input_seq { T { } } }
\cs_new_protected:Npn \__unravel_prev_input_gpush:N
  { \__unravel_prev_input_gpush_aux:NN T }
\cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:
  { \__unravel_prev_input_gpush_gtl:N \c_empty_gtl }
\cs_new_protected:Npn \__unravel_prev_input_gpush_gtl:N
  { \__unravel_prev_input_gpush_aux:NN G }
\cs_new_protected:Npn \__unravel_prev_input_gpush_aux:NN #1#2
  { \seq_gput_right:Ne \g__unravel_prev_input_seq { #1 { \exp_not:o {#2} } } }
```

(*End of definition for* `\__unravel_prev_input_gpush:` *and others.*)

`\l__unravel_prev_aux_tl`
`\__unravel_prev_input_get:N`
`\__unravel_prev_input_gpop:N`
`\__unravel_prev_input_gpop_gtl:N`
`\__unravel_prev_input_aux:NNN`
`\__unravel_prev_input_aux:NNNn`

```
\tl_new:N \l__unravel_prev_aux_tl
\cs_new_protected:Npn \__unravel_prev_input_get:N
  { \__unravel_prev_input_aux:NNN \seq_get_right:NN T }
\cs_new_protected:Npn \__unravel_prev_input_gpop:N
  { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN T }
\cs_new_protected:Npn \__unravel_prev_input_gpop_gtl:N
  { \__unravel_prev_input_aux:NNN \seq_gpop_right:NN G }
\cs_new_protected:Npn \__unravel_prev_input_aux:NNN #1#2#3
```

```
  {
    #1 \g__unravel_prev_input_seq \l__unravel_prev_aux_tl
    \exp_after:wN \__unravel_prev_input_aux:NNNn
    \exp_after:wN #2 \exp_after:wN #3 \l__unravel_prev_aux_tl
  }
\cs_new_protected:Npn \__unravel_prev_input_aux:NNNn #1#2#3
  {
    \token_if_eq_meaning:NNTF #1#3
      { \tl_set:Nn }
      { \__unravel_error:nnnnn { prev-input } {#1} {#3} }
    #2
  }
```

(*End of definition for* `\l__unravel_prev_aux_tl` *and others.*)

__unravel_prev_input_silent:n
__unravel_prev_input_silent:V
__unravel_prev_input_silent:e
__unravel_prev_input:n
__unravel_prev_input:V
__unravel_prev_input:e

```
\cs_new_protected:Npn \__unravel_prev_input_silent:n #1
  {
    \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
    \tl_put_right:Nn \l__unravel_prev_input_tl {#1}
    \__unravel_prev_input_gpush:N \l__unravel_prev_input_tl
  }
\cs_generate_variant:Nn \__unravel_prev_input_silent:n { V , e }
\cs_new_protected:Npn \__unravel_prev_input:n #1
  {
    \__unravel_prev_input_silent:n {#1}
    \__unravel_print_action:e { \tl_to_str:n {#1} }
  }
\cs_generate_variant:Nn \__unravel_prev_input:n { V , e }
```

(*End of definition for* `\__unravel_prev_input_silent:n` *and* `\__unravel_prev_input:n`*.*)

__unravel_prev_input_gtl:N

```
\cs_new_protected:Npn \__unravel_prev_input_gtl:N #1
  {
    \__unravel_prev_input_gpop_gtl:N \l__unravel_prev_input_gtl
    \gtl_concat:NNN \l__unravel_prev_input_gtl \l__unravel_prev_input_gtl #1
    \__unravel_prev_input_gpush_gtl:N \l__unravel_prev_input_gtl
  }
```

(*End of definition for* `\__unravel_prev_input_gtl:N`*.*)

__unravel_prev_input_join_get:nnN
__unravel_join_get_aux:NNnN

Pops the previous-input sequence twice to get some value in `\l__unravel_head_tl` and some sign or decimal number in `\l__unravel_tmpa_tl`. Combines them into a value, using the appropriate evaluation function, determined based on `#1`.

```
\cs_new_protected:Npn \__unravel_prev_input_join_get:nnN #1
  {
    \int_case:nnF {#1}
      {
        { 2 } { \__unravel_join_get_aux:NNnN \skip_eval:n \tex_glueexpr:D }
        { 3 } { \__unravel_join_get_aux:NNnN \muskip_eval:n \tex_muexpr:D }
      }
      {
        \__unravel_error:nnnnn { internal } { join-factor } { } { } { }
        \__unravel_join_get_aux:NNnN \use:n \prg_do_nothing:
      }
  }
\cs_new_protected:Npn \__unravel_join_get_aux:NNnN #1#2#3#4
  {
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \tl_set:Ne #4 { #1 { \l__unravel_tmpa_tl #2 \l__unravel_head_tl #3 } }
  }
```

(*End of definition for* `\__unravel_prev_input_join_get:nnN` *and* `\__unravel_join_get_aux:NNnN`.)

2.2 Variables

2.2.1 User interaction

\g__unravel_iow
\g__unravel_current_output_file_tl

The stream is used to implement the `output-file` option. At any given time it points to the file named `\g__unravel_current_output_file_tl`, unless that is empty, in which case the stream is closed. The idea is that we do not want to close the file in between different `\unravel` calls since file writing is not additive in TEX.

```
\iow_new:N \g__unravel_iow
\tl_new:N \g__unravel_current_output_file_tl
```

(*End of definition for* `\g__unravel_iow` *and* `\g__unravel_current_output_file_tl`.)

\g__unravel_before_print_state_tl
\g__unravel_before_prompt_tl

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

```
\tl_new:N \g__unravel_before_print_state_tl
\tl_new:N  \g__unravel_before_prompt_tl
```

(*End of definition for* `\g__unravel_before_print_state_tl` *and* `\g__unravel_before_prompt_tl`.)

\l__unravel_prompt_tmpa_int

```
\int_new:N \l__unravel_prompt_tmpa_int
```

(*End of definition for* `\l__unravel_prompt_tmpa_int`.)

\g__unravel_nonstop_int

The number of prompts to skip.

```
\int_new:N \g__unravel_nonstop_int
```

(*End of definition for* `\g__unravel_nonstop_int`.)

\g__unravel_current_online_int

Temporary value replacing `\g__unravel_online_int` and that can be set through the prompt.

```
\int_new:N \g__unravel_current_online_int
```

(*End of definition for* `\g__unravel_current_online_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_output_file_tl
\g__unravel_output_file_tl
\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

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`.

```
\bool_new:N \g__unravel_default_explicit_prompt_bool
\bool_new:N \g__unravel_default_internal_debug_bool
\bool_new:N \g__unravel_default_number_steps_bool
\int_new:N  \g__unravel_default_online_int
\tl_new:N   \g__unravel_default_output_file_tl
\clist_new:N \g__unravel_default_prompt_input_clist
```

```
\bool_new:N \g__unravel_default_trace_assigns_bool
\bool_new:N \g__unravel_default_trace_expansion_bool
\bool_new:N \g__unravel_default_trace_other_bool
\bool_new:N \g__unravel_default_welcome_message_bool
\bool_gset_true:N \g__unravel_default_number_steps_bool
\int_gset:Nn   \g__unravel_default_online_int { 1 }
\bool_gset_true:N \g__unravel_default_trace_assigns_bool
\bool_gset_true:N \g__unravel_default_trace_expansion_bool
\bool_gset_true:N \g__unravel_default_trace_other_bool
\bool_gset_true:N \g__unravel_default_welcome_message_bool
\bool_new:N \g__unravel_explicit_prompt_bool
\bool_new:N \g__unravel_internal_debug_bool
\bool_new:N \g__unravel_number_steps_bool
\int_new:N  \g__unravel_online_int
\tl_new:N   \g__unravel_output_file_tl
\clist_new:N \g__unravel_prompt_input_clist
\bool_new:N \g__unravel_trace_assigns_bool
\bool_new:N \g__unravel_trace_expansion_bool
\bool_new:N \g__unravel_trace_other_bool
\bool_new:N \g__unravel_welcome_message_bool
```

(*End of definition for* `\g__unravel_default_explicit_prompt_bool` *and others.*)

`\g__unravel_step_int`

Current expansion step.

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

(*End of 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:e`, which sets the escape character as appropriate before converting the argument to a string.

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

(*End of definition for* `\g__unravel_action_text_str`*.*)

`\g__unravel_default_max_action_int`
`\g__unravel_default_max_output_int`
`\g__unravel_default_max_input_int`
`\g__unravel_max_action_int`
`\g__unravel_max_output_int`
`\g__unravel_max_input_int`

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

```
\int_new:N \g__unravel_default_max_action_int
\int_new:N \g__unravel_default_max_output_int
\int_new:N \g__unravel_default_max_input_int
\int_gset:Nn \g__unravel_default_max_action_int { 50 }
\int_gset:Nn \g__unravel_default_max_output_int { 300 }
\int_gset:Nn \g__unravel_default_max_input_int { 300 }
\int_new:N \g__unravel_max_action_int
\int_new:N \g__unravel_max_output_int
\int_new:N \g__unravel_max_input_int
```

(*End of 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.

```
\bool_new:N \g__unravel_speedup_macros_bool
\bool_gset_true:N \g__unravel_speedup_macros_bool
```

(*End of definition for* `\g__unravel_speedup_macros_bool`*.*)

`\l__unravel_print_int`

The length of one piece of the terminal output.

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

(*End of 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 ⟨n⟩ of lists is `\g__unravel_input_int`. The highest numbered `gtl` represents input that comes to the left of lower numbered ones.

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

(*End of definition for* `\g__unravel_input_int`*.*)

`\g__unravel_input_tmpa_int`
`\l__unravel_input_tmpa_tl`

```
\int_new:N \g__unravel_input_tmpa_int
\tl_new:N \l__unravel_input_tmpa_tl
```

(*End of 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.

```
\seq_new:N \g__unravel_prev_input_seq
\tl_new:N \l__unravel_prev_input_tl
\gtl_new:N \l__unravel_prev_input_gtl
```

(*End of 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.

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

(*End of 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.

```
\gtl_new:N \l__unravel_head_gtl
\tl_new:N  \l__unravel_head_tl
\cs_new_eq:NN \l__unravel_head_token ?
\int_new:N \l__unravel_head_cmd_int
\int_new:N \l__unravel_head_char_int
```

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

`\l__unravel_head_meaning_tl`

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

(*End of definition for* `\l__unravel_head_meaning_tl`*.*)

`\l__unravel_argi_tl`
`\l__unravel_argii_tl`

Token list variables used to store first/second arguments.

```
\tl_new:N \l__unravel_argi_tl
\tl_new:N \l__unravel_argii_tl
```

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

`\l__unravel_tmpa_tl`
`\l__unravel_tmpb_gtl`
`\g__unravel_tmpc_tl`
`\l__unravel_tmpa_seq`
`\l__unravel_unused_gtl`
`\l__unravel_tmpb_token`

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

```
\tl_new:N \l__unravel_tmpa_tl
\gtl_new:N \l__unravel_unused_gtl
\gtl_new:N \l__unravel_tmpb_gtl
\tl_new:N \g__unravel_tmpc_tl
\seq_new:N \l__unravel_tmpa_seq
\cs_new_eq:NN \l__unravel_tmpb_token ?
```

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

`\l__unravel_defined_tl`
`\l__unravel_defining_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 because this sequence contains a string representation of the code, which is not suitable for the definition. Using a single variable here 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. It is also helpful in tracking some register assignments.

```
\tl_new:N \l__unravel_defined_tl
\tl_new:N \l__unravel_defining_tl
```

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

`\__unravel_inaccessible:w`

```
\cs_new_eq:NN \__unravel_inaccessible:w ?
```

(*End of definition for* `\__unravel_inaccessible:w`.)

`\g__unravel_lastnamedcs_tl`

Used for LuaTeX's `\lastnamedcs` primitive.

```
\tl_new:N \g__unravel_lastnamedcs_tl
```

(*End of definition for* `\g__unravel_lastnamedcs_tl`.)

`\g__unravel_after_assignment_gtl`
`\g__unravel_set_box_allowed_bool`
`\g__unravel_name_in_progress_bool`

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

```
\gtl_new:N \g__unravel_after_assignment_gtl
\bool_new:N \g__unravel_set_box_allowed_bool
\bool_new:N \g__unravel_name_in_progress_bool
```

(*End of 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.

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

(*End of definition for* `\l__unravel_after_group_gtl`.)

`\c__unravel_parameters_tl`

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

```
\group_begin:
  \cs_set_nopar:Npe \__unravel_tmp:w #1 { \c_hash_str #1 }
  \tl_const:Ne \c__unravel_parameters_tl
    { ^ \tl_map_function:nN { 123456789 } \__unravel_tmp:w }
\group_end:
```

(*End of 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.

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

(*End of definition for* `\g__unravel_val_level_int`*.*)

`\g__unravel_if_limit_tl`
`\g__unravel_if_limit_int`
`\g__unravel_if_depth_int`

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

```
\tl_new:N \g__unravel_if_limit_tl
\int_new:N \g__unravel_if_limit_int
\int_new:N \g__unravel_if_depth_int
```

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

`\l__unravel_if_nesting_int`

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

(*End of 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.

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

(*End of 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.

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

(*End of definition for* `\g__unravel_ends_int`*.*)

2.2.5 Constants

\c__unravel_plus_tl
\c__unravel_minus_tl
\c__unravel_times_tl
\c__unravel_over_tl
\c__unravel_lq_tl
\c__unravel_rq_tl
\c__unravel_dq_tl
\c__unravel_lp_tl
\c__unravel_rp_tl
\c__unravel_eq_tl
\c__unravel_comma_tl
\c__unravel_point_tl

```
\tl_const:Nn \c__unravel_plus_tl { + }
\tl_const:Nn \c__unravel_minus_tl { - }
\tl_const:Nn \c__unravel_times_tl { * }
\tl_const:Nn \c__unravel_over_tl { / }
\tl_const:Nn \c__unravel_lq_tl { ‘ }
\tl_const:Nn \c__unravel_rq_tl { ’ }
\tl_const:Nn \c__unravel_dq_tl { " } %"
\tl_const:Nn \c__unravel_lp_tl { ( }
\tl_const:Nn \c__unravel_rp_tl { ) }
\tl_const:Nn \c__unravel_eq_tl { = }
\tl_const:Nn \c__unravel_comma_tl { , }
\tl_const:Nn \c__unravel_point_tl { . }
```

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

\c__unravel_frozen_relax_gtl

TeX's `frozen_relax`, inserted by `\__unravel_insert_relax:`.

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

(*End of 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`.

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

(*End of definition for* `\g__unravel_mag_set_int`.)

__unravel_prepare_mag:

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

```
\cs_new_protected:Npn \__unravel_prepare_mag:
  {
    \int_compare:nNnT { \g__unravel_mag_set_int } > { 0 }
      {
        \int_compare:nNnF { \__unravel_mag: } = { \g__unravel_mag_set_int }
          {
            \__unravel_tex_error:nn { incompatible-mag } { }
            \int_gset_eq:NN \__unravel_mag: \g__unravel_mag_set_int
          }
      }
    \int_compare:nF { 1 <= \__unravel_mag: <= 32768 }
      {
        \__unravel_tex_error:nV { illegal-mag } \l__unravel_head_tl
        \int_gset:Nn \__unravel_mag: { 1000 }
      }
    \int_gset_eq:NN \g__unravel_mag_set_int \__unravel_mag:
  }
```

(*End of definition for* `\__unravel_prepare_mag:`.)

2.3 Numeric codes

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

`\__unravel_tex_const:nn`
`\__unravel_tex_use:n`

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

(*End of definition for* `\__unravel_tex_const:nn` *and* `\__unravel_tex_use:n`*.*)

`\__unravel_tex_primitive:nnn`
`\__unravel_tex_primitive_pdf:nnn`

```
\cs_new_protected:Npn \__unravel_tex_primitive:nnn #1#2#3
  {
    \tl_const:ce { c__unravel_tex_#1_tl }
      { { \__unravel_tex_use:n {#2} } {#3} }
  }
\cs_new_protected:Npn \__unravel_tex_primitive_pdf:nnn #1#2#3
  {
    \sys_if_engine_pdftex:F
      { \__unravel_tex_primitive:nnn {#1} {#2} {#3} }
    \__unravel_tex_primitive:nnn { pdf #1 } {#2} {#3}
  }
```

(*End of definition for* `\__unravel_tex_primitive:nnn` *and* `\__unravel_tex_primitive_pdf:nnn`*.*)

`\__unravel_new_tex_cmd:nn`
`\__unravel_new_eq_tex_cmd:nn`

```
\cs_new_protected:Npn \__unravel_new_tex_cmd:nn #1#2
  {
    \cs_new_protected:cpn
      { __unravel_cmd_ \__unravel_tex_use:n {#1} : } {#2}
  }
\cs_new_protected:Npn \__unravel_new_eq_tex_cmd:nn #1#2
  {
    \cs_new_eq:cc
      { __unravel_cmd_ \__unravel_tex_use:n {#1} : }
      { __unravel_cmd_ \__unravel_tex_use:n {#2} : }
  }
```

(*End of definition for* `\__unravel_new_tex_cmd:nn` *and* `\__unravel_new_eq_tex_cmd:nn`*.*)

`\__unravel_new_tex_expandable:nn`

```
\cs_new_protected:Npn \__unravel_new_tex_expandable:nn #1#2
  {
    \cs_new_protected:cpn
      { __unravel_expandable_ \__unravel_tex_use:n {#1} : } {#2}
  }
```

(*End of definition for* `\__unravel_new_tex_expandable:nn`*.*)

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

```
\__unravel_tex_const:nn { relax                    } { 0 }
\__unravel_tex_const:nn { begin-group_char         } { 1 }
\__unravel_tex_const:nn { end-group_char           } { 2 }
```

```
\__unravel_tex_const:nn { math_char                } { 3 }
\__unravel_tex_const:nn { tab_mark                 } { 4 }
\__unravel_tex_const:nn { alignment_char           } { 4 }
\__unravel_tex_const:nn { car_ret                  } { 5 }
\__unravel_tex_const:nn { macro_char               } { 6 }
\__unravel_tex_const:nn { superscript_char         } { 7 }
\__unravel_tex_const:nn { subscript_char           } { 8 }
\__unravel_tex_const:nn { endv                     } { 9 }
\__unravel_tex_const:nn { blank_char               } { 10 }
\__unravel_tex_const:nn { the_char                 } { 11 }
\__unravel_tex_const:nn { other_char               } { 12 }
\__unravel_tex_const:nn { par_end                  } { 13 }
\__unravel_tex_const:nn { stop                     } { 14 }
\__unravel_tex_const:nn { delim_num                } { 15 }
\__unravel_tex_const:nn { max_char_code            } { 15 }
\__unravel_tex_const:nn { char_num                 } { 16 }
\__unravel_tex_const:nn { math_char_num            } { 17 }
\__unravel_tex_const:nn { mark                     } { 18 }
\__unravel_tex_const:nn { xray                     } { 19 }
\__unravel_tex_const:nn { make_box                 } { 20 }
\__unravel_tex_const:nn { hmove                    } { 21 }
\__unravel_tex_const:nn { vmove                    } { 22 }
\__unravel_tex_const:nn { un_hbox                  } { 23 }
\__unravel_tex_const:nn { un_vbox                  } { 24 }
\__unravel_tex_const:nn { remove_item              } { 25 }
\__unravel_tex_const:nn { hskip                    } { 26 }
\__unravel_tex_const:nn { vskip                    } { 27 }
\__unravel_tex_const:nn { mskip                    } { 28 }
\__unravel_tex_const:nn { kern                     } { 29 }
\__unravel_tex_const:nn { mkern                    } { 30 }
\__unravel_tex_const:nn { leader_ship              } { 31 }
\__unravel_tex_const:nn { halign                   } { 32 }
\__unravel_tex_const:nn { valign                   } { 33 }
\__unravel_tex_const:nn { no_align                 } { 34 }
\__unravel_tex_const:nn { vrule                    } { 35 }
\__unravel_tex_const:nn { hrule                    } { 36 }
\__unravel_tex_const:nn { insert                   } { 37 }
\__unravel_tex_const:nn { vadjust                  } { 38 }
\__unravel_tex_const:nn { ignore_spaces            } { 39 }
\__unravel_tex_const:nn { after_assignment         } { 40 }
\__unravel_tex_const:nn { after_group              } { 41 }
\__unravel_tex_const:nn { break_penalty            } { 42 }
\__unravel_tex_const:nn { start_par                } { 43 }
\__unravel_tex_const:nn { ital_corr                } { 44 }
\__unravel_tex_const:nn { accent                   } { 45 }
\__unravel_tex_const:nn { math_accent              } { 46 }
\__unravel_tex_const:nn { discretionary            } { 47 }
\__unravel_tex_const:nn { eq_no                    } { 48 }
\__unravel_tex_const:nn { left_right               } { 49 }
\__unravel_tex_const:nn { math_comp                } { 50 }
\__unravel_tex_const:nn { limit_switch             } { 51 }
\__unravel_tex_const:nn { above                    } { 52 }
\__unravel_tex_const:nn { math_style               } { 53 }
\__unravel_tex_const:nn { math_choice              } { 54 }
```

```
\__unravel_tex_const:nn { non_script               } { 55 }
\__unravel_tex_const:nn { vcenter                  } { 56 }
\__unravel_tex_const:nn { case_shift               } { 57 }
\__unravel_tex_const:nn { message                  } { 58 }
\__unravel_tex_const:nn { extension                } { 59 }
\__unravel_tex_const:nn { in_stream                } { 60 }
\__unravel_tex_const:nn { begin_group              } { 61 }
\__unravel_tex_const:nn { end_group                } { 62 }
\__unravel_tex_const:nn { omit                     } { 63 }
\__unravel_tex_const:nn { ex_space                 } { 64 }
\__unravel_tex_const:nn { no_boundary              } { 65 }
\__unravel_tex_const:nn { radical                  } { 66 }
\__unravel_tex_const:nn { end_cs_name              } { 67 }
\__unravel_tex_const:nn { min_internal             } { 68 }
\__unravel_tex_const:nn { char_given               } { 68 }
\__unravel_tex_const:nn { math_given               } { 69 }
\__unravel_tex_const:nn { last_item                } { 70 }
\__unravel_tex_const:nn { max_non_prefixed_command } { 70 }
\__unravel_tex_const:nn { toks_register            } { 71 }
\__unravel_tex_const:nn { assign_toks              } { 72 }
\__unravel_tex_const:nn { assign_int               } { 73 }
\__unravel_tex_const:nn { assign_dimen             } { 74 }
\__unravel_tex_const:nn { assign_glue              } { 75 }
\__unravel_tex_const:nn { assign_mu_glue           } { 76 }
\__unravel_tex_const:nn { assign_font_dimen        } { 77 }
\__unravel_tex_const:nn { assign_font_int          } { 78 }
\__unravel_tex_const:nn { set_aux                  } { 79 }
\__unravel_tex_const:nn { set_prev_graf            } { 80 }
\__unravel_tex_const:nn { set_page_dimen           } { 81 }
\__unravel_tex_const:nn { set_page_int             } { 82 }
\__unravel_tex_const:nn { set_box_dimen            } { 83 }
\__unravel_tex_const:nn { set_shape                } { 84 }
\__unravel_tex_const:nn { def_code                 } { 85 }
\__unravel_tex_const:nn { def_family               } { 86 }
\__unravel_tex_const:nn { set_font                 } { 87 }
\__unravel_tex_const:nn { def_font                 } { 88 }
\__unravel_tex_const:nn { register                 } { 89 }
\__unravel_tex_const:nn { max_internal             } { 89 }
\__unravel_tex_const:nn { advance                  } { 90 }
\__unravel_tex_const:nn { multiply                 } { 91 }
\__unravel_tex_const:nn { divide                   } { 92 }
\__unravel_tex_const:nn { prefix                   } { 93 }
\__unravel_tex_const:nn { let                      } { 94 }
\__unravel_tex_const:nn { shorthand_def            } { 95 }
\__unravel_tex_const:nn { read_to_cs               } { 96 }
\__unravel_tex_const:nn { def                      } { 97 }
\__unravel_tex_const:nn { set_box                  } { 98 }
\__unravel_tex_const:nn { hyph_data                } { 99 }
\__unravel_tex_const:nn { set_interaction          } { 100 }
\__unravel_tex_const:nn { letterspace_font         } { 101 }
\__unravel_tex_const:nn { pdf_copy_font            } { 102 }
\__unravel_tex_const:nn { max_command              } { 102 }
\__unravel_tex_const:nn { undefined_cs             } { 103 }
\__unravel_tex_const:nn { expand_after             } { 104 }
```

```
\__unravel_tex_const:nn { no_expand                } { 105 }
\__unravel_tex_const:nn { input                    } { 106 }
\__unravel_tex_const:nn { if_test                  } { 107 }
\__unravel_tex_const:nn { fi_or_else               } { 108 }
\__unravel_tex_const:nn { cs_name                  } { 109 }
\__unravel_tex_const:nn { convert                  } { 110 }
\__unravel_tex_const:nn { the                      } { 111 }
\__unravel_tex_const:nn { top_bot_mark             } { 112 }
\__unravel_tex_const:nn { call                     } { 113 }
\__unravel_tex_const:nn { end_template             } { 117 }
```

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

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

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

A special case for LuaTeX 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.

```
\sys_if_engine_luatex:T
  {
    \__unravel_tex_primitive:nnn
      { \exp_after:wN \use_none:n \token_to_meaning:N \__unravel_special_relax: }
      { relax } { 1 }
  }
\__unravel_tex_primitive:nnn { relax             } { relax    } { 256 }
\__unravel_tex_primitive:nnn { span              } { tab_mark } { 256 }
\__unravel_tex_primitive:nnn { cr                } { car_ret  } { 257 }
\__unravel_tex_primitive:nnn { crcr              } { car_ret  } { 258 }
\__unravel_tex_primitive:nnn { par               } { par_end  } { 256 }
\__unravel_tex_primitive:nnn { end               } { stop } { 0 }
\__unravel_tex_primitive:nnn { dump              } { stop } { 1 }
\__unravel_tex_primitive:nnn { delimiter         } { delim_num } { 0 }
\__unravel_tex_primitive:nnn { char              } { char_num } { 0 }
\__unravel_tex_primitive:nnn { mathchar          } { math_char_num } { 0 }
\__unravel_tex_primitive:nnn { mark              } { mark } { 0 }
\__unravel_tex_primitive:nnn { marks             } { mark } { 5 }
\__unravel_tex_primitive:nnn { show              } { xray } { 0 }
\__unravel_tex_primitive:nnn { showbox           } { xray } { 1 }
\__unravel_tex_primitive:nnn { showthe           } { xray } { 2 }
\__unravel_tex_primitive:nnn { showlists         } { xray } { 3 }
\__unravel_tex_primitive:nnn { showgroups        } { xray } { 4 }
```

```
\__unravel_tex_primitive:nnn { showtokens       } { xray } { 5 }
\__unravel_tex_primitive:nnn { showifs          } { xray } { 6 }
\__unravel_tex_primitive:nnn { box              } { make_box } { 0 }
\__unravel_tex_primitive:nnn { copy             } { make_box } { 1 }
\__unravel_tex_primitive:nnn { lastbox          } { make_box } { 2 }
\__unravel_tex_primitive:nnn { vsplit           } { make_box } { 3 }
\__unravel_tex_primitive:nnn { vtop             } { make_box } { 4 }
\__unravel_tex_primitive:nnn { vbox             } { make_box } { 5 }
\__unravel_tex_primitive:nnn { hbox             } { make_box } { 106 }
\__unravel_tex_primitive:nnn { moveright        } { hmove } { 0 }
\__unravel_tex_primitive:nnn { moveleft         } { hmove } { 1 }
\__unravel_tex_primitive:nnn { lower            } { vmove } { 0 }
\__unravel_tex_primitive:nnn { raise            } { vmove } { 1 }
\__unravel_tex_primitive:nnn { unhbox           } { un_hbox } { 0 }
\__unravel_tex_primitive:nnn { unhcopy          } { un_hbox } { 1 }
\__unravel_tex_primitive:nnn { unvbox           } { un_vbox } { 0 }
\__unravel_tex_primitive:nnn { unvcopy          } { un_vbox } { 1 }
\__unravel_tex_primitive:nnn { pagediscards     } { un_vbox } { 2 }
\__unravel_tex_primitive:nnn { splitdiscards    } { un_vbox } { 3 }
\__unravel_tex_primitive:nnn { unpenalty        } { remove_item } { 12 }
\__unravel_tex_primitive:nnn { unkern           } { remove_item } { 11 }
\__unravel_tex_primitive:nnn { unskip           } { remove_item } { 10 }
\__unravel_tex_primitive:nnn { hfil             } { hskip } { 0 }
\__unravel_tex_primitive:nnn { hfill            } { hskip } { 1 }
\__unravel_tex_primitive:nnn { hss              } { hskip } { 2 }
\__unravel_tex_primitive:nnn { hfilneg          } { hskip } { 3 }
\__unravel_tex_primitive:nnn { hskip            } { hskip } { 4 }
\__unravel_tex_primitive:nnn { vfil             } { vskip } { 0 }
\__unravel_tex_primitive:nnn { vfill            } { vskip } { 1 }
\__unravel_tex_primitive:nnn { vss              } { vskip } { 2 }
\__unravel_tex_primitive:nnn { vfilneg          } { vskip } { 3 }
\__unravel_tex_primitive:nnn { vskip            } { vskip } { 4 }
\__unravel_tex_primitive:nnn { mskip            } { mskip } { 5 }
\__unravel_tex_primitive:nnn { kern             } { kern } { 1 }
\__unravel_tex_primitive:nnn { mkern            } { mkern } { 99 }
\__unravel_tex_primitive:nnn { shipout          } { leader_ship } { 99 }
\__unravel_tex_primitive:nnn { leaders          } { leader_ship } { 100 }
\__unravel_tex_primitive:nnn { cleaders         } { leader_ship } { 101 }
\__unravel_tex_primitive:nnn { xleaders         } { leader_ship } { 102 }
\__unravel_tex_primitive:nnn { halign           } { halign } { 0 }
\__unravel_tex_primitive:nnn { valign           } { valign } { 0 }
\__unravel_tex_primitive:nnn { beginL           } { valign } { 4 }
\__unravel_tex_primitive:nnn { endL             } { valign } { 5 }
\__unravel_tex_primitive:nnn { beginR           } { valign } { 8 }
\__unravel_tex_primitive:nnn { endR             } { valign } { 9 }
\__unravel_tex_primitive:nnn { noalign          } { no_align } { 0 }
\__unravel_tex_primitive:nnn { vrule            } { vrule } { 0 }
\__unravel_tex_primitive:nnn { hrule            } { hrule } { 0 }
\__unravel_tex_primitive:nnn { insert           } { insert } { 0 }
\__unravel_tex_primitive:nnn { vadjust          } { vadjust } { 0 }
\__unravel_tex_primitive:nnn { ignorespaces     } { ignore_spaces } { 0 }
\__unravel_tex_primitive:nnn { afterassignment  } { after_assignment } { 0 }
\__unravel_tex_primitive:nnn { aftergroup       } { after_group } { 0 }
\__unravel_tex_primitive:nnn { penalty          } { break_penalty } { 0 }
```

```
\__unravel_tex_primitive:nnn { indent            } { start_par } { 1 }
\__unravel_tex_primitive:nnn { noindent          } { start_par } { 0 }
\__unravel_tex_primitive:nnn { quitvmode         } { start_par } { 2 }
\__unravel_tex_primitive:nnn { /                 } { ital_corr } { 0 }
\__unravel_tex_primitive:nnn { accent            } { accent } { 0 }
\__unravel_tex_primitive:nnn { mathaccent        } { math_accent } { 0 }
\sys_if_engine_luatex:T
  {
    \__unravel_tex_primitive:nnn
      { explicitdiscretionary } { discretionary } { 1 }
  }
\__unravel_tex_primitive:nnn { -                 } { discretionary } { 1 }
\__unravel_tex_primitive:nnn { discretionary     } { discretionary } { 0 }
\__unravel_tex_primitive:nnn { eqno              } { eq_no } { 0 }
\__unravel_tex_primitive:nnn { leqno             } { eq_no } { 1 }
\__unravel_tex_primitive:nnn { left              } { left_right } { 30 }
\__unravel_tex_primitive:nnn { right             } { left_right } { 31 }
\__unravel_tex_primitive:nnn { middle            } { left_right } { 17 }
\__unravel_tex_primitive:nnn { mathord           } { math_comp } { 16 }
\__unravel_tex_primitive:nnn { mathop            } { math_comp } { 17 }
\__unravel_tex_primitive:nnn { mathbin           } { math_comp } { 18 }
\__unravel_tex_primitive:nnn { mathrel           } { math_comp } { 19 }
\__unravel_tex_primitive:nnn { mathopen          } { math_comp } { 20 }
\__unravel_tex_primitive:nnn { mathclose         } { math_comp } { 21 }
\__unravel_tex_primitive:nnn { mathpunct         } { math_comp } { 22 }
\__unravel_tex_primitive:nnn { mathinner         } { math_comp } { 23 }
\__unravel_tex_primitive:nnn { underline         } { math_comp } { 26 }
\__unravel_tex_primitive:nnn { overline          } { math_comp } { 27 }
\__unravel_tex_primitive:nnn { displaylimits     } { limit_switch } { 0 }
\__unravel_tex_primitive:nnn { limits            } { limit_switch } { 1 }
\__unravel_tex_primitive:nnn { nolimits          } { limit_switch } { 2 }
\__unravel_tex_primitive:nnn { above             } { above } { 0 }
\__unravel_tex_primitive:nnn { over              } { above } { 1 }
\__unravel_tex_primitive:nnn { atop              } { above } { 2 }
\__unravel_tex_primitive:nnn { abovewithdelims   } { above } { 3 }
\__unravel_tex_primitive:nnn { overwithdelims    } { above } { 4 }
\__unravel_tex_primitive:nnn { atopwithdelims    } { above } { 5 }
\__unravel_tex_primitive:nnn { displaystyle      } { math_style } { 0 }
\__unravel_tex_primitive:nnn { textstyle         } { math_style } { 2 }
\__unravel_tex_primitive:nnn { scriptstyle       } { math_style } { 4 }
\__unravel_tex_primitive:nnn { scriptscriptstyle } { math_style } { 6 }
\__unravel_tex_primitive:nnn { mathchoice        } { math_choice } { 0 }
\__unravel_tex_primitive:nnn { nonscript         } { non_script } { 0 }
\__unravel_tex_primitive:nnn { vcenter           } { vcenter } { 0 }
\__unravel_tex_primitive:nnn { lowercase         } { case_shift } { 256 }
\__unravel_tex_primitive:nnn { uppercase         } { case_shift } { 512 }
\__unravel_tex_primitive:nnn { message           } { message } { 0 }
\__unravel_tex_primitive:nnn { errmessage        } { message } { 1 }
\__unravel_tex_primitive:nnn { openout           } { extension } { 0 }
\__unravel_tex_primitive:nnn { write             } { extension } { 1 }
\__unravel_tex_primitive:nnn { closeout          } { extension } { 2 }
\__unravel_tex_primitive:nnn { special           } { extension } { 3 }
\__unravel_tex_primitive:nnn { immediate         } { extension } { 4 }
\__unravel_tex_primitive:nnn { setlanguage       } { extension } { 5 }
```

```
\__unravel_tex_primitive_pdf:nnn { literal       } { extension } { 6 }
\__unravel_tex_primitive_pdf:nnn { obj           } { extension } { 7 }
\__unravel_tex_primitive_pdf:nnn { refobj        } { extension } { 8 }
\__unravel_tex_primitive_pdf:nnn { xform         } { extension } { 9 }
\__unravel_tex_primitive_pdf:nnn { refxform      } { extension } { 10 }
\__unravel_tex_primitive_pdf:nnn { ximage        } { extension } { 11 }
\__unravel_tex_primitive_pdf:nnn { refximage     } { extension } { 12 }
\__unravel_tex_primitive_pdf:nnn { annot         } { extension } { 13 }
\__unravel_tex_primitive_pdf:nnn { startlink     } { extension } { 14 }
\__unravel_tex_primitive_pdf:nnn { endlink       } { extension } { 15 }
\__unravel_tex_primitive_pdf:nnn { outline       } { extension } { 16 }
\__unravel_tex_primitive_pdf:nnn { dest          } { extension } { 17 }
\__unravel_tex_primitive_pdf:nnn { thread        } { extension } { 18 }
\__unravel_tex_primitive_pdf:nnn { startthread   } { extension } { 19 }
\__unravel_tex_primitive_pdf:nnn { endthread     } { extension } { 20 }
\__unravel_tex_primitive_pdf:nnn { savepos       } { extension } { 21 }
\__unravel_tex_primitive_pdf:nnn { info          } { extension } { 22 }
\__unravel_tex_primitive_pdf:nnn { catalog       } { extension } { 23 }
\__unravel_tex_primitive_pdf:nnn { names         } { extension } { 24 }
\__unravel_tex_primitive_pdf:nnn { fontattr      } { extension } { 25 }
\__unravel_tex_primitive_pdf:nnn { includechars  } { extension } { 26 }
\__unravel_tex_primitive_pdf:nnn { mapfile       } { extension } { 27 }
\__unravel_tex_primitive_pdf:nnn { mapline       } { extension } { 28 }
\__unravel_tex_primitive_pdf:nnn { trailer       } { extension } { 29 }
\__unravel_tex_primitive_pdf:nnn { resettimer    } { extension } { 30 }
\__unravel_tex_primitive_pdf:nnn { fontexpand    } { extension } { 31 }
\__unravel_tex_primitive_pdf:nnn { setrandomseed } { extension } { 32 }
\__unravel_tex_primitive_pdf:nnn { snaprefpoint  } { extension } { 33 }
\__unravel_tex_primitive_pdf:nnn { snapy         } { extension } { 34 }
\__unravel_tex_primitive_pdf:nnn { snapycomp     } { extension } { 35 }
\__unravel_tex_primitive_pdf:nnn { glyphtounicode} { extension } { 36 }
\__unravel_tex_primitive_pdf:nnn { colorstack    } { extension } { 37 }
\__unravel_tex_primitive_pdf:nnn { setmatrix     } { extension } { 38 }
\__unravel_tex_primitive_pdf:nnn { save          } { extension } { 39 }
\__unravel_tex_primitive_pdf:nnn { restore       } { extension } { 40 }
\__unravel_tex_primitive_pdf:nnn { nobuiltintounicode } { extension } { 41 }
\__unravel_tex_primitive:nnn { openin                } { in_stream } { 1 }
\__unravel_tex_primitive:nnn { closein               } { in_stream } { 0 }
\__unravel_tex_primitive:nnn { begingroup            } { begin_group } { 0 }
\__unravel_tex_primitive:nnn { endgroup              } { end_group } { 0 }
\__unravel_tex_primitive:nnn { omit                  } { omit } { 0 }
\__unravel_tex_primitive:nnn { ~                     } { ex_space } { 0 }
\__unravel_tex_primitive:nnn { noboundary            } { no_boundary } { 0 }
\__unravel_tex_primitive:nnn { radical               } { radical } { 0 }
\__unravel_tex_primitive:nnn { endcsname             } { end_cs_name } { 0 }
\__unravel_tex_primitive:nnn { lastpenalty           } { last_item } { 0 }
\__unravel_tex_primitive:nnn { lastkern              } { last_item } { 1 }
\__unravel_tex_primitive:nnn { lastskip              } { last_item } { 2 }
\__unravel_tex_primitive:nnn { lastnodetype          } { last_item } { 3 }
\__unravel_tex_primitive:nnn { inputlineno           } { last_item } { 4 }
\__unravel_tex_primitive:nnn { badness               } { last_item } { 5 }
\__unravel_tex_primitive_pdf:nnn { texversion        } { last_item } { 6 }
\__unravel_tex_primitive_pdf:nnn { lastobj           } { last_item } { 7 }
\__unravel_tex_primitive_pdf:nnn { lastxform         } { last_item } { 8 }
```

```
\__unravel_tex_primitive_pdf:nnn { lastximage        } { last_item } { 9 }
\__unravel_tex_primitive_pdf:nnn { lastximagepages   } { last_item } { 10 }
\__unravel_tex_primitive_pdf:nnn { lastannot         } { last_item } { 11 }
\__unravel_tex_primitive_pdf:nnn { lastxpos          } { last_item } { 12 }
\__unravel_tex_primitive_pdf:nnn { lastypos          } { last_item } { 13 }
\__unravel_tex_primitive_pdf:nnn { retval            } { last_item } { 14 }
\__unravel_tex_primitive_pdf:nnn { lastximagecolordepth } { last_item } { 15 }
\__unravel_tex_primitive_pdf:nnn { elapsedtime       } { last_item } { 16 }
\__unravel_tex_primitive_pdf:nnn { shellescape       } { last_item } { 17 }
\__unravel_tex_primitive_pdf:nnn { randomseed        } { last_item } { 18 }
\__unravel_tex_primitive_pdf:nnn { lastlink          } { last_item } { 19 }
\__unravel_tex_primitive:nnn { eTeXversion           } { last_item } { 20 }
\__unravel_tex_primitive:nnn { currentgrouplevel     } { last_item } { 21 }
\__unravel_tex_primitive:nnn { currentgrouptype      } { last_item } { 22 }
\__unravel_tex_primitive:nnn { currentiflevel        } { last_item } { 23 }
\__unravel_tex_primitive:nnn { currentiftype         } { last_item } { 24 }
\__unravel_tex_primitive:nnn { currentifbranch       } { last_item } { 25 }
\__unravel_tex_primitive:nnn { gluestretchorder      } { last_item } { 26 }
\__unravel_tex_primitive:nnn { glueshrinkorder       } { last_item } { 27 }
\__unravel_tex_primitive:nnn { fontcharwd            } { last_item } { 28 }
\__unravel_tex_primitive:nnn { fontcharht            } { last_item } { 29 }
\__unravel_tex_primitive:nnn { fontchardp            } { last_item } { 30 }
\__unravel_tex_primitive:nnn { fontcharic            } { last_item } { 31 }
\__unravel_tex_primitive:nnn { parshapelength        } { last_item } { 32 }
\__unravel_tex_primitive:nnn { parshapeindent        } { last_item } { 33 }
\__unravel_tex_primitive:nnn { parshapedimen         } { last_item } { 34 }
\__unravel_tex_primitive:nnn { gluestretch           } { last_item } { 35 }
\__unravel_tex_primitive:nnn { glueshrink            } { last_item } { 36 }
\__unravel_tex_primitive:nnn { mutoglue              } { last_item } { 37 }
\__unravel_tex_primitive:nnn { gluetomu              } { last_item } { 38 }
\__unravel_tex_primitive:nnn { numexpr               } { last_item } { 39 }
\__unravel_tex_primitive:nnn { dimexpr               } { last_item } { 40 }
\__unravel_tex_primitive:nnn { glueexpr              } { last_item } { 41 }
\__unravel_tex_primitive:nnn { muexpr                } { last_item } { 42 }
\__unravel_tex_primitive:nnn { toks } { toks_register } { 0 }
\__unravel_tex_primitive:nnn { output                } { assign_toks } { 1 }
\__unravel_tex_primitive:nnn { everypar              } { assign_toks } { 2 }
\__unravel_tex_primitive:nnn { everymath             } { assign_toks } { 3 }
\__unravel_tex_primitive:nnn { everydisplay          } { assign_toks } { 4 }
\__unravel_tex_primitive:nnn { everyhbox             } { assign_toks } { 5 }
\__unravel_tex_primitive:nnn { everyvbox             } { assign_toks } { 6 }
\__unravel_tex_primitive:nnn { everyjob              } { assign_toks } { 7 }
\__unravel_tex_primitive:nnn { everycr               } { assign_toks } { 8 }
\__unravel_tex_primitive:nnn { errhelp               } { assign_toks } { 9 }
\__unravel_tex_primitive_pdf:nnn { pagesattr         } { assign_toks } { 10 }
\__unravel_tex_primitive_pdf:nnn { pageattr          } { assign_toks } { 11 }
\__unravel_tex_primitive_pdf:nnn { pageresources     } { assign_toks } { 12 }
\__unravel_tex_primitive_pdf:nnn { pkmode            } { assign_toks } { 13 }
\__unravel_tex_primitive:nnn { everyeof              } { assign_toks } { 14 }
\__unravel_tex_primitive:nnn { pretolerance          } { assign_int } { 0 }
\__unravel_tex_primitive:nnn { tolerance             } { assign_int } { 1 }
\__unravel_tex_primitive:nnn { linepenalty           } { assign_int } { 2 }
\__unravel_tex_primitive:nnn { hyphenpenalty         } { assign_int } { 3 }
\__unravel_tex_primitive:nnn { exhyphenpenalty       } { assign_int } { 4 }
```

```
\__unravel_tex_primitive:nnn { clubpenalty          } { assign_int } { 5 }
\__unravel_tex_primitive:nnn { widowpenalty         } { assign_int } { 6 }
\__unravel_tex_primitive:nnn { displaywidowpenalty  } { assign_int } { 7 }
\__unravel_tex_primitive:nnn { brokenpenalty        } { assign_int } { 8 }
\__unravel_tex_primitive:nnn { binoppenalty         } { assign_int } { 9 }
\__unravel_tex_primitive:nnn { relpenalty           } { assign_int } { 10 }
\__unravel_tex_primitive:nnn { predisplaypenalty    } { assign_int } { 11 }
\__unravel_tex_primitive:nnn { postdisplaypenalty   } { assign_int } { 12 }
\__unravel_tex_primitive:nnn { interlinepenalty     } { assign_int } { 13 }
\__unravel_tex_primitive:nnn { doublehyphendemerits } { assign_int } { 14 }
\__unravel_tex_primitive:nnn { finalhyphendemerits  } { assign_int } { 15 }
\__unravel_tex_primitive:nnn { adjdemerits          } { assign_int } { 16 }
\__unravel_tex_primitive:nnn { mag                  } { assign_int } { 17 }
\__unravel_tex_primitive:nnn { delimiterfactor      } { assign_int } { 18 }
\__unravel_tex_primitive:nnn { looseness            } { assign_int } { 19 }
\__unravel_tex_primitive:nnn { time                 } { assign_int } { 20 }
\__unravel_tex_primitive:nnn { day                  } { assign_int } { 21 }
\__unravel_tex_primitive:nnn { month                } { assign_int } { 22 }
\__unravel_tex_primitive:nnn { year                 } { assign_int } { 23 }
\__unravel_tex_primitive:nnn { showboxbreadth       } { assign_int } { 24 }
\__unravel_tex_primitive:nnn { showboxdepth         } { assign_int } { 25 }
\__unravel_tex_primitive:nnn { hbadness             } { assign_int } { 26 }
\__unravel_tex_primitive:nnn { vbadness             } { assign_int } { 27 }
\__unravel_tex_primitive:nnn { pausing              } { assign_int } { 28 }
\__unravel_tex_primitive:nnn { tracingonline        } { assign_int } { 29 }
\__unravel_tex_primitive:nnn { tracingmacros        } { assign_int } { 30 }
\__unravel_tex_primitive:nnn { tracingstats         } { assign_int } { 31 }
\__unravel_tex_primitive:nnn { tracingparagraphs    } { assign_int } { 32 }
\__unravel_tex_primitive:nnn { tracingpages         } { assign_int } { 33 }
\__unravel_tex_primitive:nnn { tracingoutput        } { assign_int } { 34 }
\__unravel_tex_primitive:nnn { tracinglostchars     } { assign_int } { 35 }
\__unravel_tex_primitive:nnn { tracingcommands      } { assign_int } { 36 }
\__unravel_tex_primitive:nnn { tracingrestores      } { assign_int } { 37 }
\__unravel_tex_primitive:nnn { uchyph               } { assign_int } { 38 }
\__unravel_tex_primitive:nnn { outputpenalty        } { assign_int } { 39 }
\__unravel_tex_primitive:nnn { maxdeadcycles        } { assign_int } { 40 }
\__unravel_tex_primitive:nnn { hangafter            } { assign_int } { 41 }
\__unravel_tex_primitive:nnn { floatingpenalty      } { assign_int } { 42 }
\__unravel_tex_primitive:nnn { globaldefs           } { assign_int } { 43 }
\__unravel_tex_primitive:nnn { fam                  } { assign_int } { 44 }
\__unravel_tex_primitive:nnn { escapechar           } { assign_int } { 45 }
\__unravel_tex_primitive:nnn { defaulthyphenchar    } { assign_int } { 46 }
\__unravel_tex_primitive:nnn { defaultskewchar      } { assign_int } { 47 }
\__unravel_tex_primitive:nnn { endlinechar          } { assign_int } { 48 }
\__unravel_tex_primitive:nnn { newlinechar          } { assign_int } { 49 }
\__unravel_tex_primitive:nnn { language             } { assign_int } { 50 }
\__unravel_tex_primitive:nnn { lefthyphenmin        } { assign_int } { 51 }
\__unravel_tex_primitive:nnn { righthyphenmin       } { assign_int } { 52 }
\__unravel_tex_primitive:nnn { holdinginserts       } { assign_int } { 53 }
\__unravel_tex_primitive:nnn { errorcontextlines    } { assign_int } { 54 }
\__unravel_tex_primitive:nnn { pdfoutput            } { assign_int } { 55 }
\__unravel_tex_primitive_pdf:nnn { compresslevel    } { assign_int } { 56 }
\__unravel_tex_primitive_pdf:nnn { decimaldigits    } { assign_int } { 57 }
\__unravel_tex_primitive_pdf:nnn { movechars        } { assign_int } { 58 }
```

```
\__unravel_tex_primitive_pdf:nnn { imageresolution   } { assign_int } { 59 }
\__unravel_tex_primitive_pdf:nnn { pkresolution      } { assign_int } { 60 }
\__unravel_tex_primitive_pdf:nnn { uniqueresname     } { assign_int } { 61 }
\__unravel_tex_primitive_pdf:nnn
  { optionalwaysusepdfpagebox    } { assign_int } { 62 }
\__unravel_tex_primitive_pdf:nnn
  { optionpdfinclusionerrorlevel } { assign_int } { 63 }
\__unravel_tex_primitive_pdf:nnn
  { optionpdfminorversion        } { assign_int } { 64 }
\__unravel_tex_primitive_pdf:nnn { minorversion      } { assign_int } { 64 }
\__unravel_tex_primitive_pdf:nnn { forcepagebox      } { assign_int } { 65 }
\__unravel_tex_primitive_pdf:nnn { pagebox           } { assign_int } { 66 }
\__unravel_tex_primitive_pdf:nnn
  { inclusionerrorlevel } { assign_int } { 67 }
\__unravel_tex_primitive_pdf:nnn { gamma             } { assign_int } { 68 }
\__unravel_tex_primitive_pdf:nnn { imagegamma        } { assign_int } { 69 }
\__unravel_tex_primitive_pdf:nnn { imagehicolor      } { assign_int } { 70 }
\__unravel_tex_primitive_pdf:nnn { imageapplygamma   } { assign_int } { 71 }
\__unravel_tex_primitive_pdf:nnn { adjustspacing     } { assign_int } { 72 }
\__unravel_tex_primitive_pdf:nnn { protrudechars     } { assign_int } { 73 }
\__unravel_tex_primitive_pdf:nnn { tracingfonts      } { assign_int } { 74 }
\__unravel_tex_primitive_pdf:nnn { objcompresslevel  } { assign_int } { 75 }
\__unravel_tex_primitive_pdf:nnn
  { adjustinterwordglue } { assign_int } { 76 }
\__unravel_tex_primitive_pdf:nnn { prependkern       } { assign_int } { 77 }
\__unravel_tex_primitive_pdf:nnn { appendkern        } { assign_int } { 78 }
\__unravel_tex_primitive_pdf:nnn { gentounicode      } { assign_int } { 79 }
\__unravel_tex_primitive_pdf:nnn { draftmode         } { assign_int } { 80 }
\__unravel_tex_primitive_pdf:nnn { inclusioncopyfonts } { assign_int } { 81 }
\__unravel_tex_primitive:nnn { tracingassigns        } { assign_int } { 82 }
\__unravel_tex_primitive:nnn { tracinggroups         } { assign_int } { 83 }
\__unravel_tex_primitive:nnn { tracingifs            } { assign_int } { 84 }
\__unravel_tex_primitive:nnn { tracingscantokens     } { assign_int } { 85 }
\__unravel_tex_primitive:nnn { tracingnesting        } { assign_int } { 86 }
\__unravel_tex_primitive:nnn { predisplaydirection   } { assign_int } { 87 }
\__unravel_tex_primitive:nnn { lastlinefit           } { assign_int } { 88 }
\__unravel_tex_primitive:nnn { savingvdiscards       } { assign_int } { 89 }
\__unravel_tex_primitive:nnn { savinghyphcodes       } { assign_int } { 90 }
\__unravel_tex_primitive:nnn { TeXXeTstate           } { assign_int } { 91 }
\__unravel_tex_primitive:nnn { parindent             } { assign_dimen } { 0 }
\__unravel_tex_primitive:nnn { mathsurround          } { assign_dimen } { 1 }
\__unravel_tex_primitive:nnn { lineskiplimit         } { assign_dimen } { 2 }
\__unravel_tex_primitive:nnn { hsize                 } { assign_dimen } { 3 }
\__unravel_tex_primitive:nnn { vsize                 } { assign_dimen } { 4 }
\__unravel_tex_primitive:nnn { maxdepth              } { assign_dimen } { 5 }
\__unravel_tex_primitive:nnn { splitmaxdepth         } { assign_dimen } { 6 }
\__unravel_tex_primitive:nnn { boxmaxdepth           } { assign_dimen } { 7 }
\__unravel_tex_primitive:nnn { hfuzz                 } { assign_dimen } { 8 }
\__unravel_tex_primitive:nnn { vfuzz                 } { assign_dimen } { 9 }
\__unravel_tex_primitive:nnn { delimitershortfall   } { assign_dimen } { 10 }
\__unravel_tex_primitive:nnn { nulldelimiterspace   } { assign_dimen } { 11 }
\__unravel_tex_primitive:nnn { scriptspace          } { assign_dimen } { 12 }
\__unravel_tex_primitive:nnn { predisplaysize       } { assign_dimen } { 13 }
\__unravel_tex_primitive:nnn { displaywidth         } { assign_dimen } { 14 }
```

```
\__unravel_tex_primitive:nnn { displayindent        } { assign_dimen } { 15 }
\__unravel_tex_primitive:nnn { overfullrule         } { assign_dimen } { 16 }
\__unravel_tex_primitive:nnn { hangindent           } { assign_dimen } { 17 }
\__unravel_tex_primitive:nnn { hoffset              } { assign_dimen } { 18 }
\__unravel_tex_primitive:nnn { voffset              } { assign_dimen } { 19 }
\__unravel_tex_primitive:nnn { emergencystretch     } { assign_dimen } { 20 }
\__unravel_tex_primitive_pdf:nnn { horigin          } { assign_dimen } { 21 }
\__unravel_tex_primitive_pdf:nnn { vorigin          } { assign_dimen } { 22 }
\__unravel_tex_primitive_pdf:nnn { pagewidth        } { assign_dimen } { 23 }
\__unravel_tex_primitive_pdf:nnn { pageheight       } { assign_dimen } { 24 }
\__unravel_tex_primitive_pdf:nnn { linkmargin       } { assign_dimen } { 25 }
\__unravel_tex_primitive_pdf:nnn { destmargin       } { assign_dimen } { 26 }
\__unravel_tex_primitive_pdf:nnn { threadmargin     } { assign_dimen } { 27 }
\__unravel_tex_primitive_pdf:nnn { firstlineheight  } { assign_dimen } { 28 }
\__unravel_tex_primitive_pdf:nnn { lastlinedepth    } { assign_dimen } { 29 }
\__unravel_tex_primitive_pdf:nnn { eachlineheight   } { assign_dimen } { 30 }
\__unravel_tex_primitive_pdf:nnn { eachlinedepth    } { assign_dimen } { 31 }
\__unravel_tex_primitive_pdf:nnn { ignoreddimen     } { assign_dimen } { 32 }
\__unravel_tex_primitive_pdf:nnn { pxdimen          } { assign_dimen } { 33 }
\__unravel_tex_primitive:nnn { lineskip              } { assign_glue } { 0 }
\__unravel_tex_primitive:nnn { baselineskip          } { assign_glue } { 1 }
\__unravel_tex_primitive:nnn { parskip               } { assign_glue } { 2 }
\__unravel_tex_primitive:nnn { abovedisplayskip      } { assign_glue } { 3 }
\__unravel_tex_primitive:nnn { belowdisplayskip      } { assign_glue } { 4 }
\__unravel_tex_primitive:nnn { abovedisplayshortskip } { assign_glue } { 5 }
\__unravel_tex_primitive:nnn { belowdisplayshortskip } { assign_glue } { 6 }
\__unravel_tex_primitive:nnn { leftskip              } { assign_glue } { 7 }
\__unravel_tex_primitive:nnn { rightskip             } { assign_glue } { 8 }
\__unravel_tex_primitive:nnn { topskip               } { assign_glue } { 9 }
\__unravel_tex_primitive:nnn { splittopskip          } { assign_glue } { 10 }
\__unravel_tex_primitive:nnn { tabskip               } { assign_glue } { 11 }
\__unravel_tex_primitive:nnn { spaceskip             } { assign_glue } { 12 }
\__unravel_tex_primitive:nnn { xspaceskip            } { assign_glue } { 13 }
\__unravel_tex_primitive:nnn { parfillskip           } { assign_glue } { 14 }
\__unravel_tex_primitive:nnn { thinmuskip       } { assign_mu_glue } { 15 }
\__unravel_tex_primitive:nnn { medmuskip        } { assign_mu_glue } { 16 }
\__unravel_tex_primitive:nnn { thickmuskip      } { assign_mu_glue } { 17 }
\__unravel_tex_primitive:nnn { fontdimen        } { assign_font_dimen } { 0 }
\__unravel_tex_primitive:nnn { hyphenchar       } { assign_font_int } { 0 }
\__unravel_tex_primitive:nnn { skewchar         } { assign_font_int } { 1 }
\__unravel_tex_primitive:nnn { lpcode           } { assign_font_int } { 2 }
\__unravel_tex_primitive:nnn { rpcode           } { assign_font_int } { 3 }
\__unravel_tex_primitive:nnn { efcode           } { assign_font_int } { 4 }
\__unravel_tex_primitive:nnn { tagcode          } { assign_font_int } { 5 }
\__unravel_tex_primitive_pdf:nnn { noligatures  } { assign_font_int } { 6 }
\__unravel_tex_primitive:nnn { knbscode         } { assign_font_int } { 7 }
\__unravel_tex_primitive:nnn { stbscode         } { assign_font_int } { 8 }
\__unravel_tex_primitive:nnn { shbscode         } { assign_font_int } { 9 }
\__unravel_tex_primitive:nnn { knbccode         } { assign_font_int } { 10 }
\__unravel_tex_primitive:nnn { knaccode         } { assign_font_int } { 11 }
\__unravel_tex_primitive:nnn { spacefactor      } { set_aux } { 102 }
\__unravel_tex_primitive:nnn { prevdepth        } { set_aux } { 1 }
\__unravel_tex_primitive:nnn { prevgraf         } { set_prev_graf } { 0 }
\__unravel_tex_primitive:nnn { pagegoal         } { set_page_dimen } { 0 }
```

```
\__unravel_tex_primitive:nnn { pagetotal        } { set_page_dimen } { 1 }
\__unravel_tex_primitive:nnn { pagestretch      } { set_page_dimen } { 2 }
\__unravel_tex_primitive:nnn { pagefilstretch   } { set_page_dimen } { 3 }
\__unravel_tex_primitive:nnn { pagefillstretch  } { set_page_dimen } { 4 }
\__unravel_tex_primitive:nnn { pagefilllstretch } { set_page_dimen } { 5 }
\__unravel_tex_primitive:nnn { pageshrink       } { set_page_dimen } { 6 }
\__unravel_tex_primitive:nnn { pagedepth        } { set_page_dimen } { 7 }
\__unravel_tex_primitive:nnn { deadcycles       } { set_page_int } { 0 }
\__unravel_tex_primitive:nnn { insertpenalties  } { set_page_int } { 1 }
\__unravel_tex_primitive:nnn { interactionmode  } { set_page_int } { 2 }
\__unravel_tex_primitive:nnn { wd               } { set_box_dimen } { 1 }
\__unravel_tex_primitive:nnn { dp               } { set_box_dimen } { 2 }
\__unravel_tex_primitive:nnn { ht               } { set_box_dimen } { 3 }
\__unravel_tex_primitive:nnn { parshape              } { set_shape } { 0 }
\__unravel_tex_primitive:nnn { interlinepenalties    } { set_shape } { 1 }
\__unravel_tex_primitive:nnn { clubpenalties         } { set_shape } { 2 }
\__unravel_tex_primitive:nnn { widowpenalties        } { set_shape } { 3 }
\__unravel_tex_primitive:nnn { displaywidowpenalties } { set_shape } { 4 }
\__unravel_tex_primitive:nnn { catcode               } { def_code } { 0 }
\__unravel_tex_primitive:nnn { lccode                } { def_code } { 256 }
\__unravel_tex_primitive:nnn { uccode                } { def_code } { 512 }
\__unravel_tex_primitive:nnn { sfcode                } { def_code } { 768 }
\__unravel_tex_primitive:nnn { mathcode              } { def_code } { 1024 }
\__unravel_tex_primitive:nnn { delcode               } { def_code } { 1591 }
\__unravel_tex_primitive:nnn { textfont              } { def_family } { -48 }
\__unravel_tex_primitive:nnn { scriptfont            } { def_family } { -32 }
\__unravel_tex_primitive:nnn { scriptscriptfont      } { def_family } { -16 }
\__unravel_tex_primitive:nnn { nullfont              } { set_font } { 0 }
\__unravel_tex_primitive:nnn { font                  } { def_font } { 0 }
\__unravel_tex_primitive:nnn { count           } { register } { 1 000 000 }
\__unravel_tex_primitive:nnn { dimen           } { register } { 2 000 000 }
\__unravel_tex_primitive:nnn { skip            } { register } { 3 000 000 }
\__unravel_tex_primitive:nnn { muskip          } { register } { 4 000 000 }
\__unravel_tex_primitive:nnn { advance         } { advance } { 0 }
\__unravel_tex_primitive:nnn { multiply        } { multiply } { 0 }
\__unravel_tex_primitive:nnn { divide          } { divide } { 0 }
\__unravel_tex_primitive:nnn { long            } { prefix } { 1 }
\__unravel_tex_primitive:nnn { outer           } { prefix } { 2 }
\__unravel_tex_primitive:nnn { global          } { prefix } { 4 }
\__unravel_tex_primitive:nnn { protected       } { prefix } { 8 }
\__unravel_tex_primitive:nnn { let             } { let } { 0 }
\__unravel_tex_primitive:nnn { futurelet       } { let } { 1 }
\__unravel_tex_primitive:nnn { chardef         } { shorthand_def } { 0 }
\__unravel_tex_primitive:nnn { mathchardef     } { shorthand_def } { 1 }
\__unravel_tex_primitive:nnn { countdef        } { shorthand_def } { 2 }
\__unravel_tex_primitive:nnn { dimendef        } { shorthand_def } { 3 }
\__unravel_tex_primitive:nnn { skipdef         } { shorthand_def } { 4 }
\__unravel_tex_primitive:nnn { muskipdef       } { shorthand_def } { 5 }
\__unravel_tex_primitive:nnn { toksdef         } { shorthand_def } { 6 }
\__unravel_tex_primitive:nnn { read            } { read_to_cs } { 0 }
\__unravel_tex_primitive:nnn { readline        } { read_to_cs } { 1 }
\__unravel_tex_primitive:nnn { def             } { def } { 0 }
\__unravel_tex_primitive:nnn { gdef            } { def } { 1 }
\__unravel_tex_primitive:nnn { edef            } { def } { 2 }
```

```
\__unravel_tex_primitive:nnn { xdef            } { def } { 3 }
\__unravel_tex_primitive:nnn { setbox          } { set_box } { 0 }
\__unravel_tex_primitive:nnn { hyphenation     } { hyph_data } { 0 }
\__unravel_tex_primitive:nnn { patterns        } { hyph_data } { 1 }
\__unravel_tex_primitive:nnn { batchmode       } { set_interaction } { 0 }
\__unravel_tex_primitive:nnn { nonstopmode     } { set_interaction } { 1 }
\__unravel_tex_primitive:nnn { scrollmode      } { set_interaction } { 2 }
\__unravel_tex_primitive:nnn { errorstopmode   } { set_interaction } { 3 }
\__unravel_tex_primitive:nnn { letterspacefont } { letterspace_font } { 0 }
\__unravel_tex_primitive_pdf:nnn { copyfont    } { pdf_copy_font } { 0 }
\__unravel_tex_primitive:nnn { undefined         } { undefined_cs } { 0 }
\__unravel_tex_primitive:nnn { ndefined          } { undefined_cs } { 0 }
\__unravel_tex_primitive:nnn { expandafter       } { expand_after } { 0 }
\__unravel_tex_primitive:nnn { unless            } { expand_after } { 1 }
\__unravel_tex_primitive_pdf:nnn { primitive     } { no_expand } { 1 }
\__unravel_tex_primitive:nnn { noexpand          } { no_expand } { 0 }
\__unravel_tex_primitive:nnn { input             } { input } { 0 }
\__unravel_tex_primitive:nnn { endinput          } { input } { 1 }
\__unravel_tex_primitive:nnn { scantokens        } { input } { 2 }
\__unravel_tex_primitive:nnn { if                } { if_test } { 0 }
\__unravel_tex_primitive:nnn { ifcat             } { if_test } { 1 }
\__unravel_tex_primitive:nnn { ifnum             } { if_test } { 2 }
\__unravel_tex_primitive:nnn { ifdim             } { if_test } { 3 }
\__unravel_tex_primitive:nnn { ifodd             } { if_test } { 4 }
\__unravel_tex_primitive:nnn { ifvmode           } { if_test } { 5 }
\__unravel_tex_primitive:nnn { ifhmode           } { if_test } { 5 }
\__unravel_tex_primitive:nnn { ifmmode           } { if_test } { 5 }
\__unravel_tex_primitive:nnn { ifinner           } { if_test } { 5 }
\__unravel_tex_primitive:nnn { ifvoid            } { if_test } { 9 }
\__unravel_tex_primitive:nnn { ifhbox            } { if_test } { 9 }
\__unravel_tex_primitive:nnn { ifvbox            } { if_test } { 9 }
\__unravel_tex_primitive:nnn { ifx               } { if_test } { 12 }
\__unravel_tex_primitive:nnn { ifeof             } { if_test } { 13 }
\__unravel_tex_primitive:nnn { iftrue            } { if_test } { 14 }
\__unravel_tex_primitive:nnn { iffalse           } { if_test } { 15 }
\__unravel_tex_primitive:nnn { ifcase            } { if_test } { 16 }
\__unravel_tex_primitive:nnn { ifdefined         } { if_test } { 17 }
\__unravel_tex_primitive:nnn { ifcsname          } { if_test } { 18 }
\__unravel_tex_primitive:nnn { iffontchar        } { if_test } { 19 }
\__unravel_tex_primitive:nnn { ifincsname        } { if_test } { 20 }
\__unravel_tex_primitive:nnn { ifprimitive       } { if_test } { 21 }
\__unravel_tex_primitive:nnn { ifpdfprimitive    } { if_test } { 21 }
\__unravel_tex_primitive:nnn { ifabsnum          } { if_test } { 22 }
\__unravel_tex_primitive:nnn { ifpdfabsnum       } { if_test } { 22 }
\__unravel_tex_primitive:nnn { ifabsdim          } { if_test } { 23 }
\__unravel_tex_primitive:nnn { ifpdfabsdim       } { if_test } { 23 }
\bool_if:nT { \sys_if_engine_ptex_p: || \sys_if_engine_uptex_p: }
  {
    \__unravel_tex_primitive:nnn { iftdir        } { if_test } { 5 }
    \__unravel_tex_primitive:nnn { ifydir        } { if_test } { 5 }
    \__unravel_tex_primitive:nnn { ifddir        } { if_test } { 5 }
    \__unravel_tex_primitive:nnn { ifmdir        } { if_test } { 5 }
    \__unravel_tex_primitive:nnn { iftbox        } { if_test } { 9 }
    \__unravel_tex_primitive:nnn { ifybox        } { if_test } { 9 }
    \__unravel_tex_primitive:nnn { ifdbox        } { if_test } { 9 }
    \__unravel_tex_primitive:nnn { ifmbox        } { if_test } { 9 }
    \__unravel_tex_primitive:nnn { ifjfont       } { if_test } { 24 }
    \__unravel_tex_primitive:nnn { iftfont       } { if_test } { 24 }
  }
\__unravel_tex_primitive:nnn { fi                } { fi_or_else } { 2 }
\__unravel_tex_primitive:nnn { else              } { fi_or_else } { 3 }
\__unravel_tex_primitive:nnn { or                } { fi_or_else } { 4 }
\__unravel_tex_primitive:nnn { csname            } { cs_name } { 0 }
\__unravel_tex_primitive:nnn { lastnamedcs       } { cs_name } { 1 }
\__unravel_tex_primitive:nnn { number            } { convert } { 0 }
\__unravel_tex_primitive:nnn { romannumeral      } { convert } { 1 }
\__unravel_tex_primitive:nnn { string            } { convert } { 2 }
\__unravel_tex_primitive:nnn { meaning           } { convert } { 3 }
\__unravel_tex_primitive:nnn { fontname          } { convert } { 4 }
\__unravel_tex_primitive:nnn { eTeXrevision      } { convert } { 5 }
\__unravel_tex_primitive_pdf:nnn { texrevision   } { convert } { 6 }
\__unravel_tex_primitive_pdf:nnn { texbanner     } { convert } { 7 }
\__unravel_tex_primitive:nnn { pdffontname       } { convert } { 8 }
\__unravel_tex_primitive_pdf:nnn { fontobjnum    } { convert } { 9 }
\__unravel_tex_primitive_pdf:nnn { fontsize      } { convert } { 10 }
\__unravel_tex_primitive_pdf:nnn { pageref       } { convert } { 11 }
\__unravel_tex_primitive_pdf:nnn { xformname     } { convert } { 12 }
\__unravel_tex_primitive_pdf:nnn { escapestring  } { convert } { 13 }
\__unravel_tex_primitive_pdf:nnn { escapename    } { convert } { 14 }
\__unravel_tex_primitive:nnn { leftmarginkern    } { convert } { 15 }
\__unravel_tex_primitive:nnn { rightmarginkern   } { convert } { 16 }
\__unravel_tex_primitive_pdf:nnn { strcmp        } { convert } { 17 }
\__unravel_tex_primitive_pdf:nnn { colorstackinit } { convert } { 18 }
\__unravel_tex_primitive_pdf:nnn { escapehex     } { convert } { 19 }
\__unravel_tex_primitive_pdf:nnn { unescapehex   } { convert } { 20 }
\__unravel_tex_primitive_pdf:nnn { creationdate  } { convert } { 21 }
\__unravel_tex_primitive_pdf:nnn { filemoddate   } { convert } { 22 }
\__unravel_tex_primitive_pdf:nnn { filesize      } { convert } { 23 }
\__unravel_tex_primitive_pdf:nnn { mdfivesum     } { convert } { 24 }
\__unravel_tex_primitive_pdf:nnn { filedump      } { convert } { 25 }
\__unravel_tex_primitive_pdf:nnn { match         } { convert } { 26 }
\__unravel_tex_primitive_pdf:nnn { lastmatch     } { convert } { 27 }
\__unravel_tex_primitive_pdf:nnn { uniformdeviate } { convert } { 28 }
\__unravel_tex_primitive_pdf:nnn { normaldeviate } { convert } { 29 }
\__unravel_tex_primitive_pdf:nnn { insertht      } { convert } { 30 }
\__unravel_tex_primitive_pdf:nnn { ximagebbox    } { convert } { 31 }
\__unravel_tex_primitive:nnn { jobname           } { convert } { 32 }
\sys_if_engine_luatex:T
  { \__unravel_tex_primitive:nnn { directlua     } { convert } { 33 } }
\__unravel_tex_primitive:nnn { expanded          } { convert } { 34 }
\sys_if_engine_luatex:T
  { \__unravel_tex_primitive:nnn { luaescapestring } { convert } { 35 } }
\bool_if:nT { \sys_if_engine_xetex_p: || \sys_if_engine_ptex_p: || \sys_if_engine_uptex_p: }
  {
    \__unravel_tex_primitive:nnn { Ucharcat        } { convert } { 40 }
  }
\__unravel_tex_primitive:nnn { the               } { the } { 0 }
\__unravel_tex_primitive:nnn { unexpanded        } { the } { 1 }
```

```
\__unravel_tex_primitive:nnn { detokenize        } { the } { 5 }
\__unravel_tex_primitive:nnn { topmark           } { top_bot_mark } { 0 }
\__unravel_tex_primitive:nnn { firstmark         } { top_bot_mark } { 1 }
\__unravel_tex_primitive:nnn { botmark           } { top_bot_mark } { 2 }
\__unravel_tex_primitive:nnn { splitfirstmark    } { top_bot_mark } { 3 }
\__unravel_tex_primitive:nnn { splitbotmark      } { top_bot_mark } { 4 }
\__unravel_tex_primitive:nnn { topmarks          } { top_bot_mark } { 5 }
\__unravel_tex_primitive:nnn { firstmarks        } { top_bot_mark } { 6 }
\__unravel_tex_primitive:nnn { botmarks          } { top_bot_mark } { 7 }
\__unravel_tex_primitive:nnn { splitfirstmarks   } { top_bot_mark } { 8 }
\__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.

```
\cs_new_protected:Npn \__unravel_get_next:
  {
    \__unravel_input_if_empty:TF
      {
        \__unravel_error:nnnnn { runaway-unravel } { } { } { } { }
        \__unravel_back_input_gtl:N \c__unravel_frozen_relax_gtl
      }
      { }
    \__unravel_input_gpop:N \l__unravel_head_gtl
    \gtl_head_do:NN \l__unravel_head_gtl \__unravel_get_next_aux:w
    \gtl_if_tl:NTF \l__unravel_head_gtl
      {
        \tl_set:Ne \l__unravel_head_tl
          { \gtl_head:N \l__unravel_head_gtl }
        \token_if_eq_meaning:NNT
          \l__unravel_head_token \__unravel_special_relax:
          \__unravel_get_next_notexpanded:
      }
      { \tl_clear:N \l__unravel_head_tl }
  }
\cs_new_protected:Npn \__unravel_get_next_aux:w
  { \cs_set_eq:NN \l__unravel_head_token }
```

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

__unravel_get_next_notexpanded:
__unravel_notexpanded_test:w
__unravel_notexpanded_expand:nN
__unravel_notexpanded_expand:NN

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 of that check the control sequence has the form `\notexpanded:...`. Since we don't know the escape character we must use `\cs_to_str:N`, but that function is not meant 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.

```
\cs_set_protected:Npn \__unravel_tmp:w #1
  {
    \cs_new_protected:Npn \__unravel_get_next_notexpanded:
      {
        \tl_if_eq:onTF { \l__unravel_head_tl } { \__unravel_unravel_marker: }
          { \__unravel_get_next_marker: }
          {
            \exp_args:NNe \use:nn \__unravel_notexpanded_test:w
              { \scan_stop: \exp_after:wN \cs_to_str:N \l__unravel_head_tl Z }
              \q_mark \__unravel_notexpanded_expand:n
              #1 Z \q_mark \use_none:n
              \q_stop
          }
      }
    \cs_new_protected:Npn \__unravel_notexpanded_test:w
        ##1 #1 ##2 Z \q_mark ##3##4 \q_stop
      { ##3 {##2} }
  }
\exp_args:Ne \__unravel_tmp:w { \scan_stop: \tl_to_str:n { notexpanded: } }
\group_begin:
  \char_set_catcode_active:n { 0 }
  \cs_new_protected:Npn \__unravel_notexpanded_expand:n #1
    {
      \exp_args:Ne \tl_if_empty:nTF { \str_tail:n {#1} }
        {
          \group_begin:
          \char_set_lccode:nn { 0 } { `#1 }
          \tex_lowercase:D
            {
              \group_end:
              \__unravel_notexpanded_expand:N ^^@
            }
        }
        {
          \group_begin: \exp_args:NNc \group_end:
          \__unravel_notexpanded_expand:N { \use_none:n #1 }
        }
    }
\group_end:
\cs_new_protected:Npn \__unravel_notexpanded_expand:N #1
```

```
  {
    \gtl_set:Nn \l__unravel_head_gtl {#1}
    \tl_set:Nn \l__unravel_head_tl {#1}
    \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax:
  }
```

(*End of definition for* `\__unravel_get_next_notexpanded:` *and others.*)

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

```
\cs_new_protected:Npn \__unravel_get_next_marker:
  {
    \__unravel_get_next:
    \tl_if_eq:onTF \l__unravel_head_tl { \__unravel:nn }
      { \__unravel_error:neeee { nested-unravel } { } { } { } { } }
      { \__unravel_error:neeee { internal } { marker~unknown } { } { } { } }
    \__unravel_input_gpop_item:NF \l__unravel_argi_tl
      { \__unravel_error:neeee { internal } { marker~1 } { } { } { } }
    \__unravel_input_gpop_item:NF \l__unravel_argii_tl
      { \__unravel_error:neeee { internal } { marker~2 } { } { } { } }
    \exp_args:Nno \keys_set:nn { unravel } \l__unravel_argi_tl
    \exp_args:Ne \__unravel_back_input:n
      { \exp_not:N \exp_not:n { \exp_not:o \l__unravel_argii_tl } }
    \__unravel_get_next:
  }
```

(*End of 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`.

```
\cs_new_protected:Npn \__unravel_get_token:
  {
    \__unravel_get_next:
    \__unravel_set_cmd:
  }
```

(*End of 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 XeTeX or LuaTeX primitive perhaps). Otherwise, it can be a control sequence or a character.

```
\cs_new_protected:Npn \__unravel_set_cmd:
  {
    \__unravel_set_cmd_aux_meaning:
    \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
      { }
      {
        \__unravel_token_if_expandable:NTF \l__unravel_head_token
          {
            \token_if_macro:NTF \l__unravel_head_token
```

```
              { \__unravel_set_cmd_aux_macro: }
              { \__unravel_set_cmd_aux_unknown: }
          }
          {
            \token_if_cs:NTF \l__unravel_head_token
              { \__unravel_set_cmd_aux_cs: }
              { \__unravel_set_cmd_aux_char: }
          }
      }
  }
```

(*End of definition for* `\__unravel_set_cmd:`*.*)

`\__unravel_set_cmd_aux_meaning:`
`\__unravel_set_cmd_aux_meaning:w`

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.

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_meaning:
  {
    \tl_set:Ne \l__unravel_head_meaning_tl
      {
        \exp_after:wN \__unravel_strip_escape:w
        \token_to_meaning:N \l__unravel_head_token
        \tl_to_str:n { : }
      }
    \tl_set:Ne \l__unravel_head_meaning_tl
      {
        \exp_after:wN \__unravel_set_cmd_aux_meaning:w
          \l__unravel_head_meaning_tl \q_stop
      }
  }
\use:e
  {
    \cs_new:Npn \exp_not:N \__unravel_set_cmd_aux_meaning:w
      #1 \token_to_str:N : #2 \exp_not:N \q_stop {#1}
  }
```

(*End of definition for* `\__unravel_set_cmd_aux_meaning:` *and* `\__unravel_set_cmd_aux_meaning:w`*.*)

`\__unravel_set_cmd_aux_primitive:nTF`
`\__unravel_set_cmd_aux_primitive:oTF`
`\__unravel_set_cmd_aux_primitive:nn`

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

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nTF #1#2
  {
    \cs_if_exist:cTF { c__unravel_tex_#1_tl }
      {
        \exp_last_unbraced:Nv \__unravel_set_cmd_aux_primitive:nn
          { c__unravel_tex_#1_tl }
        #2
      }
  }
\cs_generate_variant:Nn \__unravel_set_cmd_aux_primitive:nTF { o }
\cs_new_protected:Npn \__unravel_set_cmd_aux_primitive:nn #1#2
  {
    \int_set:Nn \l__unravel_head_cmd_int {#1}
    \int_set:Nn \l__unravel_head_char_int {#2}
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_macro:
  {
    \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n { call } }
    \int_zero:N \l__unravel_head_char_int
  }
```

(*End of 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`.

```
\sys_if_engine_luatex:TF
  {
    \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
      {
        \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
          \c__unravel_tex_relax_tl
        \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
          { \tl_to_str:n { xpandable~luacall } }
          { }
          {
            \__unravel_error:neeee { unknown-primitive }
              { \l__unravel_head_meaning_tl } { } { } { }
          }
      }
  }
  {
    \cs_new_protected:Npn \__unravel_set_cmd_aux_unknown:
      {
        \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
          \c__unravel_tex_relax_tl
        \__unravel_error:neeee { unknown-primitive }
          { \l__unravel_head_meaning_tl } { } { } { }
      }
  }
```

(*End of definition for* `\__unravel_set_cmd_aux_unknown:`.)

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

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_cs:
  {
    \__unravel_tl_if_in:ooTF \l__unravel_head_meaning_tl
      { \tl_to_str:n { elect~font } }
      {
        \exp_last_unbraced:NV \__unravel_set_cmd_aux_primitive:nn
          \c__unravel_tex_nullfont_tl
      }
      { \__unravel_set_cmd_aux_numeric: }
  }
```

(*End of definition for* `\__unravel_set_cmd_aux_cs:`*.*)

`\__unravel_set_cmd_aux_numeric:`
`\__unravel_set_cmd_aux_numeric:w`
`\__unravel_set_cmd_aux_given:n`
`\__unravel_set_cmd_aux_numeric:N`

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.

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:
  {
    \tl_set:Ne \l__unravel_tmpa_tl
      {
        \exp_after:wN \__unravel_set_cmd_aux_numeric:N
          \l__unravel_head_meaning_tl + 0
      }
    \exp_after:wN \__unravel_set_cmd_aux_numeric:w
      \l__unravel_tmpa_tl \q_stop
  }
\cs_new:Npn \__unravel_set_cmd_aux_numeric:N #1
  {
    \if_int_compare:w ‘#1 < ‘A \exp_stop_f:
      \exp_not:N \q_mark
      \exp_after:wN \use_i:nn
    \fi:
    #1 \__unravel_set_cmd_aux_numeric:N
  }
\cs_new_protected:Npn \__unravel_set_cmd_aux_numeric:w #1 \q_mark #2 \q_stop
  {
    \str_case:nnF {#1}
      {
        { char }     { \__unravel_set_cmd_aux_given:n { char_given } }
        { kchar }    { \__unravel_set_cmd_aux_given:n { char_given } }
        { mathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
        { omathchar } { \__unravel_set_cmd_aux_given:n { math_given } }
      }
      {
        \__unravel_set_cmd_aux_primitive:nTF {#1}
          { }
          { \__unravel_set_cmd_aux_unknown: }
        \int_add:Nn \l__unravel_head_char_int { 100 000 }
      }
    \int_add:Nn \l__unravel_head_char_int {#2}
  }
\cs_new_protected:Npn \__unravel_set_cmd_aux_given:n #1
  {
    \int_set:Nn \l__unravel_head_cmd_int { \__unravel_tex_use:n {#1} }
    \int_zero:N \l__unravel_head_char_int
  }
```

(*End of definition for* `\__unravel_set_cmd_aux_numeric:` *and others.*)

`\__unravel_set_cmd_aux_char:`
`\__unravel_set_cmd_aux_char:w`

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

```
\cs_new_protected:Npn \__unravel_set_cmd_aux_char:
  {
    \tl_set:Ne \l__unravel_head_meaning_tl
      { \token_to_meaning:N \l__unravel_head_token }
    \token_if_eq_catcode:NNT \l__unravel_head_token \c_catcode_other_token
      { \tl_set:Nn \l__unravel_head_meaning_tl { other~ } }
    \exp_after:wN \__unravel_set_cmd_aux_char:w
      \l__unravel_head_meaning_tl \q_stop
    \exp_args:NNe \int_set:Nn \l__unravel_head_char_int
      { ` \__unravel_token_to_char:N \l__unravel_head_token }
  }
\cs_new_protected:Npn \__unravel_set_cmd_aux_char:w #1 ~ #2 \q_stop
  {
    \int_set:Nn \l__unravel_head_cmd_int
      { \__unravel_tex_use:n { #1_char } }
  }
```

(*End of 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.

```
\cs_new:Npn \__unravel_input_to_str:
  {
    \int_step_function:nnnN \g__unravel_input_int { -1 } { 1 }
      \__unravel_input_to_str_aux:n
  }
\cs_new:Npn \__unravel_input_to_str_aux:n #1
  { \gtl_to_str:c { g__unravel_input_#1_gtl } }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_input_if_empty:TF
  {
    \int_compare:nNnTF \g__unravel_input_int = 0
      { \use_i:nn }
      {
        \gtl_if_empty:cTF
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
          {
            \int_gdecr:N \g__unravel_input_int
            \__unravel_input_if_empty:TF
          }
          {
            \__unravel_input_split:
            \use_ii:nn
          }
      }
  }
```

(*End of 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 occurence of that first token.

```
\cs_new_protected:Npn \__unravel_input_split:
  {
    \int_compare:nNnT \g__unravel_input_int = 1
      {
        \exp_args:Nc \__unravel_input_split_aux:N
          { g__unravel_input_1_gtl }
      }
  }
\cs_new_protected:Npn \__unravel_input_split_aux:N #1
  {
    \gtl_if_tl:NT #1
      {
        \gtl_if_head_is_N_type:NT #1
          {
            \tl_set:Ne \l__unravel_input_tmpa_tl { \gtl_left_tl:N #1 }
            \exp_args:NNe \use:nn
                \__unravel_input_split_auxii:N
                { \tl_head:N \l__unravel_input_tmpa_tl }
          }
      }
  }
\cs_new_protected:Npn \__unravel_input_split_auxii:N #1
  {
    \token_if_parameter:NF #1
      {
        \tl_replace_all:Nnn \l__unravel_input_tmpa_tl {#1}
          { \__unravel_input_split_end: \__unravel_input_split_auxiii:w #1 }
        \group_begin:
          \cs_set:Npn \__unravel_input_split_auxiii:w
           ##1 \__unravel_input_split_end: { + 1 }
          \int_gset:Nn \g__unravel_input_int
            { 0 \l__unravel_input_tmpa_tl \__unravel_input_split_end: }
        \group_end:
        \int_gset_eq:NN \g__unravel_input_tmpa_int \g__unravel_input_int
        \l__unravel_input_tmpa_tl \__unravel_input_split_end:
      }
  }
\cs_new:Npn \__unravel_input_split_end: { }
\cs_new_protected:Npn \__unravel_input_split_auxiii:w
    #1 \__unravel_input_split_end:
```

```
  {
    \gtl_gclear_new:c
      { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl }
    \gtl_gset:cn
      { g__unravel_input_ \int_use:N \g__unravel_input_tmpa_int _gtl } {#1}
    \int_gdecr:N \g__unravel_input_tmpa_int
  }
```

(*End of definition for* `\__unravel_input_split:`.)

`\__unravel_input_gset:n`

At first, all of the input is in the same `gtl`.

```
\cs_new_protected:Npn \__unravel_input_gset:n
  {
    \int_gzero:N \g__unravel_input_int
    \__unravel_back_input:n
  }
```

(*End of definition for* `\__unravel_input_gset:n`.)

`\__unravel_input_get:N`

```
\cs_new_protected:Npn \__unravel_input_get:N #1
  {
    \__unravel_input_if_empty:TF
      { \gtl_set:Nn #1 { \q_no_value } }
      {
        \gtl_get_left:cN
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
      }
  }
```

(*End of definition for* `\__unravel_input_get:N`.)

`\__unravel_input_get_left:N`
`\__unravel_input_get_left_aux:nN`
`\l__unravel_input_get_left_tl`

```
\tl_new:N \l__unravel_input_get_left_tl
\cs_new_protected:Npn \__unravel_input_get_left:N #1
  {
    \tl_clear:N #1
    \exp_args:NV \__unravel_input_get_left_aux:nN \g__unravel_input_int #1
  }
\cs_new_protected:Npn \__unravel_input_get_left_aux:nN #1#2
  {
    \int_compare:nNnF {#1} = 0
      {
        \tl_set:Ne \l__unravel_input_get_left_tl
          { \gtl_left_tl:c { g__unravel_input_#1_gtl } }
        \tl_concat:NNN #2 #2 \l__unravel_input_get_left_tl
        \gtl_if_tl:cT { g__unravel_input_#1_gtl }
          {
            \exp_args:Nf \__unravel_input_get_left_aux:nN
              { \int_eval:n { #1 - 1 } } #2
          }
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_input_gpop:N #1
  {
    \__unravel_input_if_empty:TF
      { \gtl_set:Nn #1 { \q_no_value } }
      {
        \gtl_gpop_left:cN
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_input_merge:
  {
    \int_gdecr:N \g__unravel_input_int
    \gtl_gconcat:ccc
      { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
      { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
      { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
    \gtl_gclear:c
      { g__unravel_input_ \int_eval:n { \g__unravel_input_int + 1 } _gtl }
  }
```

(*End of definition for* `\__unravel_input_merge:`*.*)

`__unravel_input_gpop_item:NTF`
`\__unravel_input_gpop_item_aux:NN`

If there is no input, we cannot pop an item. Othewise, 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.

```
\prg_new_protected_conditional:Npnn \__unravel_input_gpop_item:N #1 { F }
  {
    \int_compare:nNnTF \g__unravel_input_int = 0
      { \prg_return_false: }
      {
        \exp_args:Nc \__unravel_input_gpop_item_aux:NN
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
      }
  }
\cs_new_protected:Npn \__unravel_input_gpop_item_aux:NN #1#2
  {
    \gtl_gpop_left_item:NNTF #1#2
      { \prg_return_true: }
      {
        \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0
          { \prg_return_false: }
          {
            \int_compare:nNnTF \g__unravel_input_int = 1
              { \prg_return_false: }
              {
                \__unravel_input_merge:
                \exp_args:Nc \__unravel_input_gpop_item_aux:NN
                  {
                    g__unravel_input_
                    \int_use:N \g__unravel_input_int _gtl
                  }
                  #2
              }
          }
      }
  }
```

(*End of definition for* `\__unravel_input_gpop_item:NTF` *and* `\__unravel_input_gpop_item_aux:NN`.)

`\__unravel_input_gpop_tl:N`

```
\cs_new_protected:Npn \__unravel_input_gpop_tl:N #1
  { \tl_clear:N #1 \__unravel_input_gpop_tl_aux:N #1 }
\cs_new_protected:Npn \__unravel_input_gpop_tl_aux:N #1
  {
    \int_compare:nNnF \g__unravel_input_int = 0
      {
        \exp_args:Nc \__unravel_input_gpop_tl_aux:NN
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl } #1
      }
  }
\cs_new_protected:Npn \__unravel_input_gpop_tl_aux:NN #1#2
  {
    \gtl_if_tl:NTF #1
      {
        \tl_put_right:Ne #2 { \gtl_left_tl:N #1 }
        \gtl_gclear:N #1
        \int_gdecr:N \g__unravel_input_int
        \__unravel_input_gpop_tl_aux:N #2
      }
      {
        \int_compare:nNnTF \g__unravel_input_int > 1
          { \int_compare:nNnTF { \gtl_extra_end:N #1 } > 0 }
          { \use_i:nn }
          {
            \tl_put_right:Ne #2 { \gtl_left_tl:N #1 }
            \gtl_gpop_left_tl:N #1
          }
          {
            \__unravel_input_merge:
            \__unravel_input_gpop_tl_aux:N #2
          }
      }
  }
```

(*End of definition for* `\__unravel_input_gpop_tl:N`.)

`\__unravel_input_if_head_is_group_begin:TF`

Call `\__unravel_input_if_empty:TF` to remove empty levels from the input stack, then check if the left-most non-empty level starts with an explicit begin-group character token.

```
\prg_new_protected_conditional:Npnn \__unravel_input_if_head_is_group_begin: { T , F , TF }
```

```
  {
    \__unravel_input_if_empty:TF
      { \prg_return_false: }
      {
        \exp_args:Nc \gtl_if_head_is_group_begin:NTF
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
          { \prg_return_true: }
          { \prg_return_false: }
      }
  }
```

(*End of definition for* `\__unravel_input_if_head_is_group_begin:TF`.)

`\__unravel_back_input:n`
`\__unravel_back_input:V`
`\__unravel_back_input:o`

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.

```
\cs_new_protected:Npn \__unravel_back_input:n
  {
    \int_gincr:N \g__unravel_input_int
    \gtl_gclear_new:c { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
    \gtl_gset:cn { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
  }
\cs_generate_variant:Nn \__unravel_back_input:n { V , o }
```

(*End of definition for* `\__unravel_back_input:n`.)

`\__unravel_back_input_gtl:N`

Insert a generalized token list back into the input.

```
\cs_new_protected:Npn \__unravel_back_input_gtl:N #1
  {
    \gtl_if_tl:NTF #1
      { \exp_args:Ne \__unravel_back_input:n { \gtl_left_tl:N #1 } }
      {
        \gtl_gconcat:cNc
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
          #1
          { g__unravel_input_ \int_use:N \g__unravel_input_int _gtl }
      }
  }
```

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

`\__unravel_back_input:`

Insert the last token read back into the input stream.

```
\cs_new_protected:Npn \__unravel_back_input:
  { \__unravel_back_input_gtl:N \l__unravel_head_gtl }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_back_input_tl_o:
  {
    \tl_set:Ne \l__unravel_tmpa_tl
      { \exp_args:NV \exp_not:o \l__unravel_head_tl }
    \__unravel_back_input:V \l__unravel_tmpa_tl
    \__unravel_print_expansion:e
```

```
      { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \l__unravel_tmpa_tl }
  }
```

(*End of 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`.

```
\cs_new_protected:Npn \__unravel_insert_relax:
  {
    \__unravel_back_input:
    \gtl_set_eq:NN \l__unravel_head_gtl \c__unravel_frozen_relax_gtl
    \__unravel_back_input:
    \__unravel_print_action:
  }
```

(*End of definition for* `\__unravel_insert_relax:`*.*)

`\__unravel_insert_group_begin_error:`

```
\cs_new_protected:Npn \__unravel_insert_group_begin_error:
  {
    \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
    \__unravel_back_input:
    \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
    \__unravel_back_input:
    \__unravel_tex_error:nV { missing-lbrace } \l__unravel_tmpa_tl
    \__unravel_print_action:
  }
```

(*End of definition for* `\__unravel_insert_group_begin_error:`*.*)

`\__unravel_insert_dollar_error:`

```
\cs_new_protected:Npn \__unravel_insert_dollar_error:
  {
    \__unravel_back_input:
    \__unravel_back_input:n { $ } % $
    \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
    \__unravel_print_action:
  }
```

(*End of definition for* `\__unravel_insert_dollar_error:`*.*)

2.5.3 Macro calls

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

```
\use:e
  {
    \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:NN #1 }
      {
        \exp_not:n { \exp_after:wN \__unravel_macro_split_do:wN }
        \exp_not:n { \token_to_meaning:N #1 \q_mark { } }
        \tl_to_str:n { : } \exp_not:n { -> \q_mark \use_none:nnnn }
        \exp_not:N \q_stop
      }
    \exp_not:n { \cs_new:Npn \__unravel_macro_split_do:wN }
        \exp_not:n {#1} \tl_to_str:n { : } \exp_not:n { #2 -> }
        \exp_not:n { #3 \q_mark #4 #5 \q_stop #6 }
      { \exp_not:n { #4 #6 {#1} {#2} {#3} } }
  }
\cs_new:Npn \__unravel_macro_prefix:N #1
  { \__unravel_macro_split_do:NN #1 \use_i:nnn }
\cs_new:Npn \__unravel_macro_parameter:N #1
  { \__unravel_macro_split_do:NN #1 \use_ii:nnn }
\cs_new:Npn \__unravel_macro_replacement:N #1
  { \__unravel_macro_split_do:NN #1 \use_iii:nnn }
```

(*End of definition for* `\__unravel_macro_prefix:N`*,* `\__unravel_macro_parameter:N`*, and* `\__unravel_macro_replacement:N`*.*)

__unravel_macro_call:
__unravel_macro_call_safe:
__unravel_macro_call_quick:
__unravel_macro_call_quick_loop:NNN
__unravel_macro_call_quick_loop:NN
__unravel_macro_call_quick_runaway:Nw

Macros are simply expanded once. We cannot determine precisely which tokens a macro will need for its parameters, but we know that it must form a balanced token list. Thus we can be safe by extracting the longest balanced prefix in the input and working with that. We need to check if the argument was braced, to improve the error recovery for a non-`\long` macro receiving `\par`.

```
\cs_new_protected:Npn \__unravel_macro_call:
  {
    \bool_if:NTF \g__unravel_speedup_macros_bool
      {
        \tl_set:Ne \l__unravel_tmpa_tl
          { ^ \exp_after:wN \__unravel_macro_parameter:N \l__unravel_head_tl }
        \__unravel_tl_if_in:ooTF \c__unravel_parameters_tl \l__unravel_tmpa_tl
          { \__unravel_macro_call_quick: } { \__unravel_macro_call_safe: }
      }
      { \__unravel_macro_call_safe: }
    \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
    \__unravel_print_expansion:
  }
\cs_new_protected:Npn \__unravel_macro_call_safe:
  {
    \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
    \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
  }
\cs_new_protected:Npn \__unravel_macro_call_quick:
  {
    \exp_after:wN \__unravel_macro_call_quick_loop:NNN \l__unravel_tmpa_tl
      { ? \use_none_delimit_by_q_stop:w } \q_stop
  }
\cs_new_protected:Npn \__unravel_macro_call_quick_loop:NNN #1#2#3
  {
    \use_none:n #2
    \__unravel_input_if_head_is_group_begin:TF
      { \__unravel_macro_call_quick_loop:NN \prg_do_nothing: }
      { \__unravel_macro_call_quick_loop:NN \use:n }
    #3
  }
\cs_new_protected:Npn \__unravel_macro_call_quick_loop:NN #1#2
```

```
  {
    \__unravel_input_gpop_item:NF \l__unravel_tmpa_tl
      { \__unravel_macro_call_quick_runaway:Nw #2 }
    \tl_put_right:Ne \l__unravel_head_tl
      { #1 { \exp_not:V \l__unravel_tmpa_tl } }
    \__unravel_macro_call_quick_loop:NNN
    #2
  }
\cs_new_protected:Npn \__unravel_macro_call_quick_runaway:Nw #1#2 \q_stop
  {
    \__unravel_error:neeee { runaway-macro-parameter }
      { \tl_to_str:N \l__unravel_head_tl } { \tl_to_str:n {#1} } { } { }
  }
```

(*End of 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 TeX 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 mimick TeX's structure, distinguishing macros from other commands because we find macro arguments very differently from primitives.

```
\cs_new_protected:Npn \__unravel_expand_do:N
  {
    \__unravel_set_action_text:
    \bool_if:NT \g__unravel_internal_debug_bool
      {
        \__unravel_set_cmd:
        \exp_args:Ne \iow_term:n { Exp:~\int_to_arabic:n { \l__unravel_head_cmd_int } }
      }
    \token_if_macro:NTF \l__unravel_head_token
      { \__unravel_macro_call: }
      { \__unravel_expand_nonmacro: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_expand_nonmacro:
  {
    \__unravel_set_cmd_aux_meaning:
    \__unravel_set_cmd_aux_primitive:oTF { \l__unravel_head_meaning_tl }
      {
        \cs_if_exist_use:cF
          { __unravel_expandable_ \int_use:N \l__unravel_head_cmd_int : }
          { \__unravel_error:neeee { internal } { expandable } { } { } { } }
      }
      {
        \__unravel_set_cmd_aux_unknown:
        \__unravel_input_gpop_tl:N \l__unravel_tmpa_tl
        \tl_put_right:NV \l__unravel_head_tl \l__unravel_tmpa_tl
        \exp_args:NV \__unravel_back_input:o \l__unravel_head_tl
        \__unravel_print_expansion:
      }
  }
```

(*End of 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 LaTeX3 these are situations where TeX "`f`-expands"). It is also the basis of the `\__unravel_main_loop:`.

```
\cs_new_protected:Npn \__unravel_get_x_next:
  {
    \__unravel_get_next:
    \__unravel_token_if_expandable:NT \l__unravel_head_token
      { \__unravel_expand_do:N \__unravel_get_x_next: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_get_x_or_protected:
  {
    \__unravel_get_next:
    \__unravel_token_if_protected:NF \l__unravel_head_token
      { \__unravel_expand_do:N \__unravel_get_x_or_protected: }
  }
```

(*End of definition for* `\__unravel_get_x_or_protected:`*.*)

`\__unravel_get_token_xdef:`
`\__unravel_get_token_x:`

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).

```
\cs_new_protected:Npn \__unravel_get_token_xdef:
  {
    \__unravel_get_next:
    \__unravel_token_if_protected:NF \l__unravel_head_token
      { \__unravel_expand_do:N \__unravel_get_token_xdef: }
  }
\cs_new_protected:Npn \__unravel_get_token_x:
  {
    \__unravel_get_next:
    \__unravel_token_if_protected:NF \l__unravel_head_token
      { \__unravel_expand_do:N \__unravel_get_token_x: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_get_x_non_blank:
  {
    \__unravel_get_x_next:
    \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
      { \__unravel_get_x_non_blank: }
  }
```

(*End of definition for* `\__unravel_get_x_non_blank:`*.*)

`\__unravel_get_x_non_relax:`

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

```
\cs_new_protected:Npn \__unravel_get_x_non_relax:
  {
    \__unravel_get_x_next:
    \token_if_eq_meaning:NNTF \l__unravel_head_token \scan_stop:
      { \__unravel_get_x_non_relax: }
      {
        \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_special_relax:
          { \__unravel_get_x_non_relax: }
          {
            \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
              { \__unravel_get_x_non_relax: }
          }
      }
  }
```

(*End of definition for* `\__unravel_get_x_non_relax:`*.*)

`\__unravel_skip_optional_space:`

```
\cs_new_protected:Npn \__unravel_skip_optional_space:
  {
    \__unravel_get_x_next:
    \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
      { \__unravel_back_input: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_optional_equals:
  {
    \__unravel_get_x_non_blank:
    \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_eq_tl
      { \__unravel_prev_input:n { = } }
      {
        \__unravel_prev_input_silent:n { = }
        \__unravel_back_input:
      }
  }
```

(*End of definition for* `\__unravel_scan_optional_equals:`.)

`\__unravel_scan_left_brace:`

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

```
\cs_new_protected:Npn \__unravel_scan_left_brace:
  {
    \__unravel_get_x_non_relax:
    \token_if_eq_catcode:NNF \l__unravel_head_token \c_group_begin_token
      {
        \__unravel_insert_group_begin_error:
        \__unravel_get_next:
      }
  }
```

(*End of 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 TeX looks for keywords are quite tricky to get right, in particular with respect to expansion, case-insensitivity, and spaces. We get rid of the case issue by requiring the keyword to be given in both cases, intertwined: for instance, `\__unravel_scan_keyword:n { pPtT }`. Then loop through pairs of letters (which should be matching lowercase and uppercase letters). The looping auxiliary takes three arguments, the first of which is a boolean, `true` if spaces are allowed (no letter of the keyword has been found yet). At each iteration, get a token, with expansion, and test whether it is a non-active character equal (in character code) to either letter of the pair: this happens if the token is not “definable” (neither a control sequence nor an active character) and it has the right string representation… well, it could also be doubled (macro parameter character), hence we look at the first character only; spaces become an empty string, but this works out because no keyword contains a space. So, at each iteration, if the token is the correct non-active character, add it to 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 TeX’s skipping of leading spaces here must be intertwined with the search for keyword, as is shown by the (plain TeX) example

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

```
\cs_new_protected:Npn \__unravel_scan_keyword:n #1
  { \__unravel_scan_keyword:nTF {#1} { } { } }
\prg_new_protected_conditional:Npnn \__unravel_scan_keyword:n #1
  { T , F , TF }
  {
    \__unravel_prev_input_gpush_gtl:
    \__unravel_scan_keyword_loop:NNN \c_true_bool
      #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop
  }
\cs_new_protected:Npn \__unravel_scan_keyword_loop:NNN #1#2#3
  {
    \quark_if_recursion_tail_stop_do:nn {#2}
      { \__unravel_scan_keyword_true: }
    \quark_if_recursion_tail_stop_do:nn {#3}
      { \__unravel_error:neeee { internal } { odd-keyword-length } { } { } { } }
    \__unravel_get_x_next:
    \__unravel_scan_keyword_test:NNTF #2#3
      {
        \__unravel_prev_input_gtl:N \l__unravel_head_gtl
        \__unravel_scan_keyword_loop:NNN \c_false_bool
      }
      {
        \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
          { \__unravel_scan_keyword_false:w }
        \bool_if:NF #1
          { \__unravel_scan_keyword_false:w }
        \__unravel_scan_keyword_loop:NNN #1#2#3
      }
  }
\prg_new_protected_conditional:Npnn \__unravel_scan_keyword_test:NN #1#2
  { TF }
  {
    \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
      { \prg_return_false: }
      {
        \str_if_eq:eeTF
          { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#1}
          { \prg_return_true: }
          {
            \str_if_eq:eeTF
              { \str_head:f { \gtl_to_str:N \l__unravel_head_gtl } } {#2}
              { \prg_return_true: }
              { \prg_return_false: }
          }
      }
  }
\cs_new_protected:Npn \__unravel_scan_keyword_true:
  {
    \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
    \__unravel_prev_input:e { \gtl_to_str:N \l__unravel_tmpb_gtl }
    \prg_return_true:
  }
\cs_new_protected:Npn \__unravel_scan_keyword_false:w
    #1 \q_recursion_stop
  {
    \__unravel_back_input:
    \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
    \__unravel_back_input_gtl:N \l__unravel_tmpb_gtl
    \prg_return_false:
  }
```

(*End of definition for* `\__unravel_scan_keyword:n` *and others.*)

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

```
\cs_new_protected:Npn \__unravel_scan_to:
  {
    \__unravel_scan_keyword:nF { tToO }
      {
        \__unravel_error:nnnnn { missing-to } { } { } { } { }
        \__unravel_prev_input:n { to }
      }
  }
```

(*End of definition for* `\__unravel_scan_to:`*.*)

`\__unravel_scan_font_ident:` Find a font identifier.

```
\cs_new_protected:Npn \__unravel_scan_font_ident:
  {
    \__unravel_get_x_non_blank:
    \__unravel_set_cmd:
    \int_case:nnF \l__unravel_head_cmd_int
      {
        { \__unravel_tex_use:n { def_font } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { letterspace_font } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { pdf_copy_font } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { set_font } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { def_family } }
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_int:
          }
      }
      {
        \__unravel_error:nnnnn { missing-font-id } { } { } { } { }
        \__unravel_back_input:
        \__unravel_prev_input:n { \__unravel_nullfont: }
      }
  }
```

(*End of definition for* `\__unravel_scan_font_ident:`*.*)

`\__unravel_scan_font_int:` Find operands for one of `\hyphenchar`'s friends (command code `assign_font_int=78`).

```
\cs_new_protected:Npn \__unravel_scan_font_int:
  {
    \int_case:nnF \l__unravel_head_char_int
```

```
      {
        { 0 } { \__unravel_scan_font_ident: }
        { 1 } { \__unravel_scan_font_ident: }
        { 6 } { \__unravel_scan_font_ident: }
      }
      { \__unravel_scan_font_ident: \__unravel_scan_int: }
  }
```

(*End of definition for* `\__unravel_scan_font_int:`.)

`\__unravel_scan_font_dimen:`

Find operands for `\fontdimen`.

```
\cs_new_protected:Npn \__unravel_scan_font_dimen:
  {
    \__unravel_scan_int:
    \__unravel_scan_font_ident:
  }
```

(*End of definition for* `\__unravel_scan_font_dimen:`.)

`\__unravel_rescan_something_internal:n`
`\__unravel_scan_something_aux:nwn`

Receives an (explicit) "level" argument:

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

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

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

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

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

Finally, tell the user the tokens that have been found (if there was a single token, its meaning as well) and their value. Use `=>` rather than `=` because the value displayed is the value used, not the actual value (this matters in constructions such as

`\parindent=\parskip` where a skip or a dimen is downgraded to a dimen or an int, or when there was an error).

```
\cs_new_protected:Npn \__unravel_rescan_something_internal:n #1
  {
    \__unravel_set_cmd:
    \__unravel_set_action_text:
    \tl_set:Nf \l__unravel_tmpa_tl { \__unravel_thing_case: }
    \exp_after:wN \__unravel_scan_something_aux:nwn
      \l__unravel_tmpa_tl \q_stop {#1}
  }
\cs_new_protected:Npn \__unravel_scan_something_aux:nwn #1#2 \q_stop #3
  {
    \int_compare:nT { #3 < 4 <= #1 }
      {
        \__unravel_back_input:
        \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
        \__unravel_thing_use_get:nnNN { 1 } {#3} \c_zero_dim \l__unravel_tmpa_tl
        \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl { 1 }
        \__unravel_break:w
      }
    \int_compare:nNnT {#1} = { 8 }
      {
        \__unravel_tex_error:nV { the-cannot } \l__unravel_head_tl
        \__unravel_rescan_something_internal_auxii:nn 0 { 0 }
        \__unravel_break:w
      }
    \tl_if_empty:nF {#2}
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        #2
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
      }
    \int_compare:nNnTF {#3} < { 4 }
      { \__unravel_thing_use_get:nnNN {#1} {#3} \l__unravel_head_tl \l__unravel_tmpa_tl }
      { \tl_set:Ne \l__unravel_tmpa_tl { \__unravel_the:w \l__unravel_head_tl } }
    \__unravel_rescan_something_internal_auxii:Vn \l__unravel_tmpa_tl {#1}
    \__unravel_break_point:
    \int_compare:nNnT {#3} < { 4 } { \__unravel_print_action: }
  }
\cs_new_protected:Npn \__unravel_rescan_something_internal_auxii:nn #1#2
  {
    \__unravel_prev_input_silent:n {#1}
    \__unravel_set_action_text:
    \__unravel_set_action_text:e
      { \g__unravel_action_text_str \use:n { ~ => ~ } \tl_to_str:n {#1} }
    \int_gset:Nn \g__unravel_val_level_int {#2}
  }
\cs_generate_variant:Nn \__unravel_rescan_something_internal_auxii:nn { V }
```

(*End of definition for* `\__unravel_rescan_something_internal:n` *and* `\__unravel_scan_something_aux:nwn`*.*)

`\__unravel_thing_case:`
`\__unravel_thing_last_item:`
`\__unravel_thing_register:`

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`, `set_shape` and `register`, the level of the token, or what has to be scanned, 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.

```
\cs_new:Npn \__unravel_thing_case:
  {
    \int_case:nnF \l__unravel_head_cmd_int
      {
        { 68 } { 0                        } % char_given
        { 69 } { 0                        } % math_given
        { 70 } { \__unravel_thing_last_item:      } % last_item
        { 71 } { 5 \__unravel_scan_toks_register: } % toks_register
        { 72 } { 5                        } % assign_toks
        { 73 } { 0                        } % assign_int
        { 74 } { 1                        } % assign_dimen
        { 75 } { 2                        } % assign_glue
        { 76 } { 3                        } % assign_mu_glue
        { 77 } { 1 \__unravel_scan_font_dimen:    } % assign_font_dimen
        { 78 } { 0 \__unravel_scan_font_int:      } % assign_font_int
        { 79 } { \__unravel_thing_set_aux:        } % set_aux
        { 80 } { 0                        } % set_prev_graf
        { 81 } { 1                        } % set_page_dimen
        { 82 } { 0                        } % set_page_int
        { 83 } { 1 \__unravel_scan_int:           } % set_box_dimen
        { 84 } { \__unravel_thing_set_shape:      } % set_shape
        { 85 } { 0 \__unravel_scan_int:           } % def_code
        { 86 } { 4 \__unravel_scan_int:           } % def_family
        { 87 } { 4                        } % set_font
        { 88 } { 4                        } % def_font
        { 89 } { \__unravel_thing_register:       } % register
        {101 } { 4                        } % letterspace_font
        {102 } { 4                        } % pdf_copy_font
      }
      { 8 }
  }
\cs_new:Npn \__unravel_thing_set_aux:
  { \int_compare:nNnTF \l__unravel_head_char_int = { 1 } { 1 } { 0 } }
\cs_new:Npn \__unravel_thing_set_shape:
  { \int_compare:nNnTF \l__unravel_head_char_int = 0 { 0 } { 0 \__unravel_scan_int: } }
\cs_new:Npn \__unravel_thing_last_item:
  {
    \int_compare:nNnTF \l__unravel_head_char_int < { 26 }
      {
        \int_case:nnF \l__unravel_head_char_int
          {
            { 1 } { 1 } % lastkern
            { 2 } { 2 } % lastskip
          }
          { 0 } % other integer parameters
      }
      {
        \int_case:nnF \l__unravel_head_char_int
```

```
          {
            { 26 } { 0 \__unravel_scan_normal_glue: } % gluestretchorder
            { 27 } { 0 \__unravel_scan_normal_glue: } % glueshrinkorder
            { 28 } % fontcharwd
              { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
            { 29 } % fontcharht
              { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
            { 30 } % fontchardp
              { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
            { 31 } % fontcharic
              { 1 \__unravel_scan_font_ident: \__unravel_scan_int: }
            { 32 } { 1 \__unravel_scan_int: } % parshapelength
            { 33 } { 1 \__unravel_scan_int: } % parshapeindent
            { 34 } { 1 \__unravel_scan_int: } % parshapedimen
            { 35 } { 1 \__unravel_scan_normal_glue: } % gluestretch
            { 36 } { 1 \__unravel_scan_normal_glue: } % glueshrink
            { 37 } { 2 \__unravel_scan_mu_glue: } % mutoglue
            { 38 } { 3 \__unravel_scan_normal_glue: } % gluetomu
            { 39 } % numepr
              { 0 \__unravel_scan_expr:N \__unravel_scan_int: }
            { 40 } % dimexpr
              { 1 \__unravel_scan_expr:N \__unravel_scan_normal_dimen: }
            { 41 } % glueexpr
              { 2 \__unravel_scan_expr:N \__unravel_scan_normal_glue: }
            { 42 } % muexpr
              { 3 \__unravel_scan_expr:N \__unravel_scan_mu_glue: }
          }
          { }
      }
  }
\cs_new:Npn \__unravel_thing_register:
  {
    \int_eval:n { \l__unravel_head_char_int / 1 000 000 - 1 }
    \int_compare:nNnT { \tl_tail:V \l__unravel_head_char_int } = 0
      { \__unravel_scan_int: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_toks_register:
  {
    \int_compare:nNnT \l__unravel_head_char_int = 0
      { \__unravel_scan_int: }
  }
```

(*End of definition for* `\__unravel_scan_toks_register:`*.*)

`\__unravel_thing_use_get:nnNN`

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

```
\cs_new_protected:Npn \__unravel_thing_use_get:nnNN #1#2#3#4
  {
    \int_compare:nNnTF {#2} < { 3 }
      {
        \int_compare:nNnT {#1} = { 3 }
          { \__unravel_tex_error:nV { incompatible-units } #3 }
        \tl_set:Ne #4
          {
            \int_case:nn { \int_min:nn {#1} {#2} }
              {
                { 0 } \int_eval:n
                { 1 } \dim_eval:n
                { 2 } \skip_eval:n
              }
            { \int_compare:nNnT {#1} = { 3 } \tex_mutoglue:D #3 }
          }
      }
      {
        \int_case:nnF {#1}
          {
            { 0 } { \tl_set:Ne #4 { \int_eval:n {#3} } }
            { 3 } { \tl_set:Ne #4 { \muskip_eval:n {#3} } }
          }
          {
            \__unravel_tex_error:nV { incompatible-units } #3
            \tl_set:Ne #4 { \muskip_eval:n { \tex_gluetomu:D #3 } }
          }
      }
  }
```

(*End of definition for* `\__unravel_thing_use_get:nnNN`*.*)

`\__unravel_scan_expr:N`
`\__unravel_scan_expr_aux:NN`
`\__unravel_scan_factor:N`

```
\cs_new_protected:Npn \__unravel_scan_expr:N #1
  { \__unravel_scan_expr_aux:NN #1 \c_false_bool }
\cs_new_protected:Npn \__unravel_scan_expr_aux:NN #1#2
  {
    \__unravel_get_x_non_blank:
    \__unravel_scan_factor:N #1
    \__unravel_scan_expr_op:NN #1#2
  }
\cs_new_protected:Npn \__unravel_scan_expr_op:NN #1#2
  {
    \__unravel_get_x_non_blank:
    \token_case_meaning:NnF \l__unravel_head_tl
      {
        \c__unravel_plus_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_expr_aux:NN #1#2
          }
        \c__unravel_minus_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_expr_aux:NN #1#2
          }
        \c__unravel_times_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_get_x_non_blank:
            \__unravel_scan_factor:N \__unravel_scan_int:
            \__unravel_scan_expr_op:NN #1#2
          }
        \c__unravel_over_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_get_x_non_blank:
            \__unravel_scan_factor:N \__unravel_scan_int:
            \__unravel_scan_expr_op:NN #1#2
          }
        \c__unravel_rp_tl
          {
            \bool_if:NTF #2
              { \__unravel_prev_input:V \l__unravel_head_tl }
              { \__unravel_back_input: }
          }
      }
      {
        \bool_if:NTF #2
          {
            \__unravel_error:nnnnn { missing-rparen } { } { } { } { }
            \__unravel_back_input:
            \__unravel_prev_input:V \c__unravel_rp_tl
          }
          {
            \token_if_eq_meaning:NNF \l__unravel_head_token \scan_stop:
              { \__unravel_back_input: }
          }
      }
  }
\cs_new_protected:Npn \__unravel_scan_factor:N #1
  {
    \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_lp_tl
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \__unravel_scan_expr_aux:NN #1 \c_true_bool
      }
      {
        \__unravel_back_input:
        #1
      }
  }
```

(*End of 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`.

```
\cs_new_protected:Npn \__unravel_scan_signs:
  {
    \__unravel_get_x_non_blank:
    \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_plus_tl
```

```
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \__unravel_scan_signs:
      }
      {
        \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_minus_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_signs:
          }
      }
  }
```

(*End of definition for* `\__unravel_scan_signs:`*.*)

`\__unravel_scan_int:`
`\__unravel_scan_int_char:`
`\__unravel_scan_int_lq:`
`\__unravel_scan_int_explicit:n`

```
\cs_new_protected:Npn \__unravel_scan_int:
  {
    \__unravel_scan_signs:
    \__unravel_set_cmd:
    \__unravel_cmd_if_internal:TF
      { \__unravel_rescan_something_internal:n { 0 } }
      { \__unravel_scan_int_char: }
  }
\cs_new_protected:Npn \__unravel_scan_int_char:
  {
    \token_case_meaning:NnF \l__unravel_head_tl
      {
        \c__unravel_lq_tl { \__unravel_scan_int_lq: }
        \c__unravel_rq_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_get_x_next:
            \__unravel_scan_int_explicit:Nn \c_false_bool { ’ }
          }
        \c__unravel_dq_tl
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_get_x_next:
            \__unravel_scan_int_explicit:Nn \c_false_bool { " } % "
          }
      }
      { \__unravel_scan_int_explicit:Nn \c_false_bool { } }
  }
\cs_new_protected:Npn \__unravel_scan_int_lq:
  {
    \__unravel_get_next:
    \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
      {
        \tl_set:Ne \l__unravel_head_tl
          { \__unravel_token_to_char:N \l__unravel_head_token }
      }
    \tl_set:Ne \l__unravel_tmpa_tl
      { \int_eval:n { \exp_after:wN ‘ \l__unravel_head_tl } }
    \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
    \__unravel_print_action:e
      { ` \gtl_to_str:N \l__unravel_head_gtl = \l__unravel_tmpa_tl }
    \__unravel_skip_optional_space:
  }
\cs_new_protected:Npn \__unravel_scan_int_explicit:Nn #1#2
  {
    \if_int_compare:w 1
        < #2 1 \exp_after:wN \exp_not:N \l__unravel_head_tl \exp_stop_f:
      \exp_after:wN \use_i:nn
    \else:
      \exp_after:wN \use_ii:nn
    \fi:
    {
      \__unravel_prev_input:V \l__unravel_head_tl
      \__unravel_get_x_next:
      \__unravel_scan_int_explicit:Nn \c_true_bool {#2}
    }
    {
      \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
        { \__unravel_back_input: }
      \bool_if:NF #1
        {
          \__unravel_tex_error:nV { missing-number } \l__unravel_head_tl
          \__unravel_prev_input:n { 0 }
        }
    }
  }
```

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

`\__unravel_scan_normal_dimen:`

```
\cs_new_protected:Npn \__unravel_scan_normal_dimen:
  { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_dimen:nN #1#2
  {
    \__unravel_scan_signs:
    \__unravel_prev_input_gpush:
    \__unravel_set_cmd:
    \__unravel_cmd_if_internal:TF
      {
        \int_compare:nNnTF {#1} = { 3 }
          { \__unravel_rescan_something_internal:n { 3 } }
          { \__unravel_rescan_something_internal:n { 1 } }
        \int_compare:nNnT \g__unravel_val_level_int = { 0 }
          { \__unravel_scan_dim_unit:nN {#1} #2 }
      }
      { \__unravel_scan_dimen_char:nN {#1} #2 }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_prev_input_silent:V \l__unravel_head_tl
  }
\cs_new_protected:Npn \__unravel_scan_dimen_char:nN #1#2
    {
    \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
      { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
    \tl_if_eq:NNTF \l__unravel_head_tl \c__unravel_point_tl
      {
        \__unravel_prev_input:n { . }
        \__unravel_scan_decimal_loop:
      }
      {
        \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl
          {
            \__unravel_back_input:
            \__unravel_scan_int:
            \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_comma_tl
              { \tl_set_eq:NN \l__unravel_head_tl \c__unravel_point_tl }
            \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_point_tl
              {
                \__unravel_input_gpop:N \l__unravel_tmpb_gtl
                \__unravel_prev_input:n { . }
                \__unravel_scan_decimal_loop:
              }
          }
          {
            \__unravel_back_input:
            \__unravel_scan_int:
          }
      }
    \__unravel_scan_dim_unit:nN {#1} #2
    }
\cs_new_protected:Npn \__unravel_scan_dim_unit:nN #1#2
    {
    \bool_if:NT #2
      {
        \__unravel_scan_keyword:nT { fFiIlL }
          {
            \__unravel_scan_inf_unit_loop:
            \__unravel_break:w
          }
      }
    \__unravel_get_x_non_blank:
    \__unravel_set_cmd:
    \__unravel_cmd_if_internal:TF
      {
        \__unravel_prev_input_gpush:
        \__unravel_rescan_something_internal:n {#1}
        \int_compare:nNnTF \g__unravel_val_level_int = { 0 }
          { \__unravel_prev_input_join_get:nnN {#1} { sp } \l__unravel_tmpa_tl }
          { \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl }
        \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
        \exp_after:wN \use_none:n \__unravel_break:w
      }
      { }
    \__unravel_back_input:
    \int_compare:nNnT {#1} = { 3 }
      {
        \__unravel_scan_keyword:nT { mMuU } { \__unravel_break:w }
        \__unravel_tex_error:nV { missing-mu } \l__unravel_head_tl
        \__unravel_prev_input:n { mu }
        \__unravel_break:w
      }
    \__unravel_scan_keyword:nT { eEmM } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { eExX } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { pPxX } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { tTrRuUeE }
      { \__unravel_prepare_mag: }
    \__unravel_scan_keyword:nT { pPtT } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { iInN } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { pPcC } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { cCmM } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { mMmM } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { bBpP } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { dDdD } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { cCcC } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { nNdD } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { nNcC } { \__unravel_break:w }
    \__unravel_scan_keyword:nT { sSpP } { \__unravel_break:w }
    \__unravel_tex_error:nV { missing-pt } \l__unravel_head_tl
    \__unravel_prev_input:n { pt }
    \__unravel_break_point:
    \__unravel_skip_optional_space:
  }
\cs_new_protected:Npn \__unravel_scan_inf_unit_loop:
  { \__unravel_scan_keyword:nT { lL } { \__unravel_scan_inf_unit_loop: } }
\cs_new_protected:Npn \__unravel_scan_decimal_loop:
  {
    \__unravel_get_x_next:
    \tl_if_empty:NTF \l__unravel_head_tl
      { \use_ii:nn }
      { \__unravel_tl_if_in:ooTF { 0123456789 } \l__unravel_head_tl }
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \__unravel_scan_decimal_loop:
      }
      {
        \token_if_eq_catcode:NNF \l__unravel_head_token \c_space_token
          { \__unravel_back_input: }
        \__unravel_prev_input_silent:n { ~ }
      }
  }
```

(*End of definition for* `\__unravel_scan_dimen:nN`*.*)

`\__unravel_scan_normal_glue:`
`\__unravel_scan_mu_glue:`

```
\cs_new_protected:Npn \__unravel_scan_normal_glue:
  { \__unravel_scan_glue:n { 2 } }
\cs_new_protected:Npn \__unravel_scan_mu_glue:
  { \__unravel_scan_glue:n { 3 } }
```

(*End of definition for* `\__unravel_scan_normal_glue:` *and* `\__unravel_scan_mu_glue:`*.*)

`\__unravel_scan_glue:n`

```
\cs_new_protected:Npn \__unravel_scan_glue:n #1
  {
    \__unravel_prev_input_gpush:
    \__unravel_scan_signs:
    \__unravel_prev_input_gpush:
    \__unravel_set_cmd:
    \__unravel_cmd_if_internal:TF
      {
        \__unravel_rescan_something_internal:n {#1}
        \int_case:nnF \g__unravel_val_level_int
          {
            { 0 } { \__unravel_scan_dim_unit:nN {#1} \c_false_bool }
            { 1 } { }
          }
          { \__unravel_break:w }
      }
      { \__unravel_back_input: \__unravel_scan_dimen:nN {#1} \c_false_bool }
    \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
    \__unravel_prev_input_gpush:
    \__unravel_prev_input_gpush:N \l__unravel_tmpa_tl
    \__unravel_scan_keyword:nT { pPlLuUsS }
      { \__unravel_scan_dimen:nN {#1} \c_true_bool }
    \__unravel_scan_keyword:nT { mMiInNuUsS }
      { \__unravel_scan_dimen:nN {#1} \c_true_bool }
    \__unravel_break_point:
    \__unravel_prev_input_join_get:nnN {#1} { } \l__unravel_tmpa_tl
    \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
  }
```

(*End of definition for* `\__unravel_scan_glue:n`*.*)

`\__unravel_scan_file_name:`

```
\cs_new_protected:Npn \__unravel_scan_file_name:
  {
    \__unravel_get_x_non_relax:
    \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
      { \__unravel_scan_group_x:N \c_false_bool }
      {
        \__unravel_back_input:
        \bool_gset_true:N \g__unravel_name_in_progress_bool
        \bool_gset_false:N \g__unravel_quotes_bool
        \__unravel_get_x_non_blank:
        \__unravel_scan_file_name_loop:
```

```
        \bool_gset_false:N \g__unravel_name_in_progress_bool
        \__unravel_prev_input_silent:n { ~ }
      }
  }
\cs_new_protected:Npn \__unravel_scan_file_name_loop:
  {
    \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
      { \__unravel_back_input: }
      {
        \tl_set:Ne \l__unravel_tmpa_tl
          { \__unravel_token_to_char:N \l__unravel_head_token }
        \tl_if_eq:NNT \l__unravel_tmpa_tl \c__unravel_dq_tl
          {
            \bool_if:NTF \g__unravel_quotes_bool
              { \bool_set_false:N } { \bool_set_true:N } \g__unravel_quotes_bool
          }
        \bool_if:NTF \g__unravel_quotes_bool
          { \use:n } { \tl_if_eq:NNF \l__unravel_tmpa_tl \c_space_tl }
          {
            \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
            \__unravel_get_x_next:
            \__unravel_scan_file_name_loop:
          }
      }
  }
\bool_new:N \g__unravel_quotes_bool
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_r_token:
  {
    \bool_do_while:nn
      { \tl_if_eq_p:NN \l__unravel_head_tl \c_space_tl }
      { \__unravel_get_next: }
    \__unravel_gtl_if_head_is_definable:NF \l__unravel_head_gtl
      {
        \__unravel_error:nnnnn { missing-cs } { } { } { } { }
        \__unravel_back_input:
        \tl_set:Nn \l__unravel_head_tl { \__unravel_inaccessible:w }
      }
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
  }
```

(*End of definition for* `\__unravel_scan_r_token:`.)

`\__unravel_scan_toks_to_str:`

```
\cs_new_protected:Npn \__unravel_scan_toks_to_str:
  {
    \__unravel_prev_input_gpush:
    \__unravel_scan_toks:NN \c_false_bool \c_true_bool
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_prev_input_silent:e
      { { \exp_after:wN \tl_to_str:n \l__unravel_tmpa_tl } }
  }
```

(*End of definition for* `\__unravel_scan_toks_to_str:`*.*)

`\__unravel_scan_pdf_ext_toks:`

```
\cs_new_protected:Npn \__unravel_scan_pdf_ext_toks:
  {
    \__unravel_prev_input_gpush:
    \__unravel_scan_toks:NN \c_false_bool \c_true_bool
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_prev_input_silent:e
      { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
  }
```

(*End of 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`).

```
\cs_new_protected:Npn \__unravel_scan_toks:NN #1#2
  {
    \bool_if:NT #1 { \__unravel_scan_param: }
    \__unravel_scan_left_brace:
    \bool_if:NTF #2
      { \__unravel_scan_group_x:N #1 }
      { \__unravel_scan_group_n:N #1 }
  }
```

(*End of definition for* `\__unravel_scan_toks:NN`*.*)

`\__unravel_scan_param:`
`\__unravel_scan_param_aux:`

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`.

```
\cs_new_protected:Npn \__unravel_scan_param:
  {
    \tl_clear:N \l__unravel_tmpa_tl
    \__unravel_scan_param_aux:
    \tl_put_right:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
    \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
  }
\cs_new_protected:Npn \__unravel_scan_param_aux:
  {
    \__unravel_get_next:
    \tl_concat:NNN \l__unravel_tmpa_tl
      \l__unravel_tmpa_tl \l__unravel_head_tl
    \tl_if_empty:NTF \l__unravel_head_tl
      { \__unravel_back_input: } { \__unravel_scan_param_aux: }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_group_n:N #1
  {
    \gtl_set_eq:NN \l__unravel_head_gtl \c_group_begin_gtl
    \__unravel_back_input:
    \__unravel_input_gpop_item:NF \l__unravel_head_tl
      {
        \__unravel_error:nnnnn { runaway-text } { } { } { } { }
        \__unravel_exit_hard:w
      }
    \tl_set:Ne \l__unravel_head_tl { { \exp_not:V \l__unravel_head_tl } }
    \bool_if:NT #1
      { \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl }
    \__unravel_prev_input_silent:V \l__unravel_head_tl
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_group_x:N #1
  {
    \__unravel_input_gpop_tl:N \l__unravel_head_tl
    \__unravel_back_input:V \l__unravel_head_tl
    \bool_if:NTF #1
      {
        \__unravel_prev_input_silent:V \c_left_brace_str
        \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
        \__unravel_scan_group_xdef:n { 1 }
      }
      {
        \__unravel_prev_input_gpush_gtl:
        \__unravel_prev_input_gtl:N \l__unravel_head_gtl
        \__unravel_scan_group_x:n { 1 }
        \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
        \__unravel_prev_input_silent:e
          { \gtl_left_tl:N \l__unravel_tmpb_gtl }
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_scan_group_xdef:n #1
  {
    \__unravel_get_token_xdef:
    \tl_if_empty:NTF \l__unravel_head_tl
      {
        \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
          {
            \__unravel_prev_input_silent:V \c_left_brace_str
            \tl_put_right:Nn \l__unravel_defining_tl { { \if_false: } \fi: }
            \__unravel_scan_group_xdef:f { \int_eval:n { #1 + 1 } }
          }
          {
            \__unravel_prev_input_silent:V \c_right_brace_str
            \tl_put_right:Nn \l__unravel_defining_tl { \if_false: { \fi: } }
            \int_compare:nNnF {#1} = 1
              { \__unravel_scan_group_xdef:f { \int_eval:n { #1 - 1 } } }
          }
      }
      {
        \__unravel_prev_input_silent:V \l__unravel_head_tl
        \tl_put_right:Ne \l__unravel_defining_tl
          { \exp_not:N \exp_not:N \exp_not:V \l__unravel_head_tl }
        \__unravel_scan_group_xdef:n {#1}
      }
  }
\cs_generate_variant:Nn \__unravel_scan_group_xdef:n { f }
```

(*End of definition for* `\__unravel_scan_group_xdef:n`.)

`\__unravel_scan_group_x:n`

```
\cs_new_protected:Npn \__unravel_scan_group_x:n #1
  {
    \__unravel_get_token_x:
    \__unravel_prev_input_gtl:N \l__unravel_head_gtl
    \tl_if_empty:NTF \l__unravel_head_tl
      {
        \gtl_if_head_is_group_begin:NTF \l__unravel_head_gtl
          { \__unravel_scan_group_x:f { \int_eval:n { #1 + 1 } } }
          {
            \int_compare:nNnF {#1} = 1
              { \__unravel_scan_group_x:f { \int_eval:n { #1 - 1 } } }
          }
      }
      { \__unravel_scan_group_x:n {#1} }
  }
\cs_generate_variant:Nn \__unravel_scan_group_x:n { f }
```

(*End of definition for* `\__unravel_scan_group_x:n`.)

`\__unravel_scan_alt_rule:`

```
\cs_new_protected:Npn \__unravel_scan_alt_rule:
  {
    \__unravel_scan_keyword:nTF { wWiIdDtThH }
      {
        \__unravel_scan_normal_dimen:
        \__unravel_scan_alt_rule:
      }
      {
        \__unravel_scan_keyword:nTF { hHeEiIgGhHtT }
          {
            \__unravel_scan_normal_dimen:
            \__unravel_scan_alt_rule:
          }
          {
            \__unravel_scan_keyword:nT { dDeEpPtThH }
              {
                \__unravel_scan_normal_dimen:
                \__unravel_scan_alt_rule:
              }
          }
      }
  }
```

(*End of definition for* `\__unravel_scan_alt_rule:`.)

`\__unravel_scan_spec:`

Some TEX primitives accept the keywords `to` and `spread`, followed by a dimension.

```
\cs_new_protected:Npn \__unravel_scan_spec:
  {
    \__unravel_scan_keyword:nTF { tToO } { \__unravel_scan_normal_dimen: }
      {
        \__unravel_scan_keyword:nT { sSpPrReEaAdD }
          { \__unravel_scan_normal_dimen: }
      }
    \__unravel_scan_left_brace:
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_do_box:N #1
  {
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_compare:nNnTF
      \l__unravel_head_cmd_int = { \__unravel_tex_use:n { make_box } }
      { \__unravel_do_begin_box:N #1 }
      {
        \bool_if:NTF #1
          {
            \int_case:nnTF \l__unravel_head_cmd_int
              {
                { \__unravel_tex_use:n { hrule } } { }
                { \__unravel_tex_use:n { vrule } } { }
              }
              { \__unravel_do_leaders_rule: }
              { \__unravel_do_box_error: }
          }
          { \__unravel_do_box_error: }
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_do_box_error:
  {
    \__unravel_back_input:
    \__unravel_error:nnnnn { missing-box } { } { } { } { }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_do_begin_box:N #1
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \int_case:nnTF \l__unravel_head_char_int
      {
        { 0 } { \__unravel_scan_int: } % box
        { 1 } { \__unravel_scan_int: } % copy
        { 2 } { } % lastbox
        { 3 } % vsplit
          {
            \__unravel_scan_int:
            \__unravel_scan_to:
            \__unravel_scan_normal_dimen:
          }
      }
      { \__unravel_do_simple_box:N #1 }
      { \__unravel_do_box_explicit:N #1 }
  }
```

(*End of definition for* `\__unravel_do_begin_box:N`.)

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

```
\cs_new_protected:Npn \__unravel_do_simple_box:N #1
  {
    \bool_if:NTF #1 { \__unravel_do_leaders_fetch_skip: }
```

```
      {
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \tl_if_head_eq_meaning:VNT \l__unravel_head_tl \tex_shipout:D
          { \__unravel_prepare_mag: }
        \tl_use:N \l__unravel_head_tl \scan_stop:
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
      }
  }
```

(*End of definition for* `\__unravel_do_simple_box:N`.)

`\__unravel_do_leaders_fetch_skip:`

```
\cs_new_protected:Npn \__unravel_do_leaders_fetch_skip:
  {
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_compare:nNnTF \l__unravel_head_cmd_int
      = { \__unravel_tex_use:n { \mode_if_vertical:TF { vskip } { hskip } } }
      {
        \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
        \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
        \__unravel_do_append_glue:
      }
      {
        \__unravel_back_input:
        \__unravel_error:nnnnn { improper-leaders } { } { } { } { }
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
      }
  }
```

(*End of 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 TEX 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 TEX 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). TEX 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.

```
\cs_new_protected:Npn \__unravel_do_box_explicit:N #1
  {
    \token_if_eq_meaning:NNTF \l__unravel_head_token \__unravel_hbox:w
      { \__unravel_box_hook:N \tex_everyhbox:D }
      { \__unravel_box_hook:N \tex_everyvbox:D }
    \__unravel_scan_spec:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_set_action_text:e
      { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
    \seq_push:Nf \l__unravel_leaders_box_seq
```

```
      { \bool_if:NTF #1 { \mode_if_vertical:TF { v } { h } } { Z } }
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \gtl_gconcat:NNN \g__unravel_output_gtl
      \g__unravel_output_gtl \c_group_begin_gtl
    \tl_use:N \l__unravel_head_tl
      \c_group_begin_token \__unravel_box_hook_end:
  }
```

(*End of definition for* `\__unravel_do_box_explicit:N`*.*)

`\__unravel_box_hook:N`
`\__unravel_box_hook:w`
`\__unravel_box_hook_end:`

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

```
\cs_new_protected:Npn \__unravel_box_hook:N #1
  {
    \tl_set:NV \l__unravel_tmpa_tl #1
    \str_if_eq:eeF
      { \tl_head:N \l__unravel_tmpa_tl } { \exp_not:N \__unravel_box_hook:w }
      {
        \exp_args:Ne #1
          {
            \exp_not:n { \__unravel_box_hook:w \prg_do_nothing: }
            \exp_not:V #1
          }
      }
    \cs_gset_protected:Npn \__unravel_box_hook:w ##1 \__unravel_box_hook_end:
      {
        \exp_args:No #1 {##1}
        \cs_gset_eq:NN \__unravel_box_hook:w \prg_do_nothing:
        \gtl_clear:N \l__unravel_after_group_gtl
        \__unravel_print_action:
        \__unravel_back_input:o {##1}
        \__unravel_set_action_text:e
          { \token_to_meaning:N #1 = \tl_to_str:o {##1} }
        \tl_if_empty:oF {##1} { \__unravel_print_action: }
      }
  }
\cs_new_eq:NN \__unravel_box_hook:w \prg_do_nothing:
\cs_new_eq:NN \__unravel_box_hook_end: \prg_do_nothing:
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_do_leaders_rule:
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_scan_alt_rule:
    \__unravel_do_leaders_fetch_skip:
  }
```

(*End of definition for* `\__unravel_do_leaders_rule:`*.*)

2.9 Paragraphs

__unravel_charcode_if_safe:nTF

```
\prg_new_protected_conditional:Npnn \__unravel_charcode_if_safe:n #1 { TF }
  {
    \bool_if:nTF
      {
        \int_compare_p:n { #1 = `! }
        || \int_compare_p:n { `' <= #1 <= `[ }
        || \int_compare_p:n { #1 = `] }
        || \int_compare_p:n { ` ` <= #1 <= `z }
      }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

(*End of definition for* `\__unravel_charcode_if_safe:nTF`.)

__unravel_char:n
__unravel_char:V
__unravel_char:e

```
\cs_new_protected:Npn \__unravel_char:n #1
  {
    \tex_char:D #1 \scan_stop:
    \__unravel_charcode_if_safe:nTF {#1}
      {
        \tl_set:Ne \l__unravel_tmpa_tl { \char_generate:nn {#1} { 12 } }
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
      }
      {
        \tl_set:Ne \l__unravel_tmpa_tl
          { \exp_not:N \char \int_eval:n {#1} ~ }
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
        \__unravel_print_action:e
          { " \char_generate:nn {#1} { 12 } " = \tl_to_str:N \l__unravel_tmpa_tl }
      }
  }
\cs_generate_variant:Nn \__unravel_char:n { V , e }
```

(*End of definition for* `\__unravel_char:n`.)

__unravel_char_in_mmode:n
__unravel_char_in_mmode:V
__unravel_char_in_mmode:e

```
\cs_new_protected:Npn \__unravel_char_in_mmode:n #1
  {
    \int_compare:nNnTF { \tex_mathcode:D #1 }
      = { \sys_if_engine_luatex:TF { "1000000 } { "8000 } }
      { % math active
        \__unravel_active_do:nn {#1} { \gtl_set:Nn \l__unravel_head_gtl }
        \__unravel_back_input:
        \__unravel_print_action:e
          { \char_generate:nn {#1} { 12 } ~ active }
      }
      { \__unravel_char:n {#1} }
  }
\cs_generate_variant:Nn \__unravel_char_in_mmode:n { V , e }
```

(*End of definition for* `\__unravel_char_in_mmode:n`.)

`\__unravel_mathchar:n`
`\__unravel_mathchar:e`

```
\cs_new_protected:Npn \__unravel_mathchar:n #1
  {
    \tex_mathchar:D #1 \scan_stop:
    \tl_set:Ne \l__unravel_tmpa_tl
      { \exp_not:N \mathchar " \int_to_hex:n {#1} ~ } % "
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_tmpa_tl
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
  }
\cs_generate_variant:Nn \__unravel_mathchar:n { e }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_new_graf:N #1
  {
    \tl_set:NV \l__unravel_tmpa_tl \__unravel_everypar:w
    \__unravel_everypar:w { }
    \bool_if:NTF #1 { \tex_indent:D } { \tex_noindent:D }
    \exp_args:NV \__unravel_everypar:w \l__unravel_tmpa_tl
    \__unravel_back_input:V \l__unravel_tmpa_tl
    \__unravel_print_action:e
      {
        \g__unravel_action_text_str \c_space_tl : ~
        \token_to_str:N \everypar = { \tl_to_str:N \l__unravel_tmpa_tl }
      }
  }
```

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

`\__unravel_par_if_hmode:`

This is like the `end_graf` procedure in TEX.

```
\cs_new_protected:Npn \__unravel_par_if_hmode:
  { \mode_if_horizontal:T { \__unravel_par: } }
```

(*End of definition for* `\__unravel_par_if_hmode:`.)

`\__unravel_par:`

```
\cs_new_protected:Npn \__unravel_par:
  {
    \tex_par:D
    \gtl_gput_right:Nn \g__unravel_output_gtl { \par }
    \__unravel_print_action:e { Paragraph~end. }
  }
```

(*End of definition for* `\__unravel_par:`.)

`\__unravel_build_page:`

```
\cs_new_protected:Npn \__unravel_build_page:
  {
  }
```

(*End of 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.

```
\int_new:N \l__unravel_choice_int
```

(*End of 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`.

```
\cs_new_protected:Npn \__unravel_handle_right_brace:
  {
    \int_compare:nTF { 1 <= \__unravel_currentgrouptype: <= 13 }
      {
        \gtl_gconcat:NNN \g__unravel_output_gtl
          \g__unravel_output_gtl \c_group_end_gtl
        \int_case:nnF \__unravel_currentgrouptype:
          {
            { 10 } { } % disc
            { 13 } { } % math_choice
          }
          { \__unravel_back_input_gtl:N \l__unravel_after_group_gtl }
        \int_case:nn \__unravel_currentgrouptype:
          {
            { 1 } { \__unravel_end_simple_group: } % simple
            { 2 } { \__unravel_end_box_group: } % hbox
            { 3 } { \__unravel_end_box_group: } % adjusted_hbox
            { 4 } { \__unravel_par_if_hmode: \__unravel_end_box_group: } % vbox
            { 5 } { \__unravel_par_if_hmode: \__unravel_end_box_group: } % vtop
            { 6 } { \__unravel_end_align_group: } % align
            { 7 } { \__unravel_end_no_align_group: } % no_align
            { 8 } { \__unravel_end_output_group: } % output
            { 9 } { \__unravel_end_simple_group: } % math
            { 10 } { \__unravel_end_choice_group:NN 2 \discretionary  } % disc
            { 11 } { \__unravel_par_if_hmode: \__unravel_end_simple_group: } % insert
            { 12 } { \__unravel_par_if_hmode: \__unravel_end_simple_group: } % vcenter
            { 13 } { \__unravel_end_choice_group:NN 3 \mathchoice } % math_choice
          }
      }
      { % bottom_level, semi_simple, math_shift, math_left
        \l__unravel_head_token
        \__unravel_print_action:
      }
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_end_simple_group:
  {
    \l__unravel_head_token
    \__unravel_print_action:
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_end_box_group:
  {
    \seq_pop:NN \l__unravel_leaders_box_seq \l__unravel_tmpa_tl
    \exp_args:No \__unravel_end_box_group_aux:n { \l__unravel_tmpa_tl }
  }
\cs_new_protected:Npn \__unravel_end_box_group_aux:n #1
  {
    \str_if_eq:eeTF {#1} { Z }
      { \__unravel_end_simple_group: }
      {
        \__unravel_get_x_non_relax:
        \__unravel_set_cmd:
        \int_compare:nNnTF \l__unravel_head_cmd_int
          = { \__unravel_tex_use:n { #1 skip } }
          {
            \tl_put_left:Nn \l__unravel_head_tl { \c_group_end_token }
            \__unravel_do_append_glue:
          }
          {
            \__unravel_back_input:
            \c_group_end_token \group_begin: \group_end:
            \__unravel_print_action:
          }
      }
  }
```

(*End of definition for* `\__unravel_end_box_group:`.)

`\__unravel_end_align_group:`

```
\cs_new_protected:Npn \__unravel_end_align_group:
  {
    \__unravel_not_implemented:n { end_align_group }
    \__unravel_end_simple_group:
  }
```

(*End of definition for* `\__unravel_end_align_group:`.)

`\__unravel_end_no_align_group:`

```
\cs_new_protected:Npn \__unravel_end_no_align_group:
  {
    \__unravel_not_implemented:n { end_no_align_group }
    \__unravel_end_simple_group:
  }
```

(*End of definition for* `\__unravel_end_no_align_group:`.)

`\__unravel_end_output_group:`

```
\cs_new_protected:Npn \__unravel_end_output_group:
  {
    \__unravel_not_implemented:n { end_output_group }
    \__unravel_end_simple_group:
  }
```

(*End of definition for* `\__unravel_end_output_group:`*.*)

__unravel_end_choice_group:NN
__unravel_end_choice_group:nN

```
\cs_new_protected:Npn \__unravel_end_choice_group:NN #1#2
  {
    \int_compare:nNnTF \l__unravel_choice_int > {#1}
      {
        \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
        \c_group_end_token
        \__unravel_print_action:e
          { \token_to_str:N #2 \prg_replicate:nn { #1 + 1 } { {...} } }
      }
      { \exp_args:NV \__unravel_end_choice_group:nN \l__unravel_choice_int #2 }
  }
\cs_new_protected:Npn \__unravel_end_choice_group:nN #1#2
  {
    \__unravel_scan_left_brace:
    \gtl_gconcat:NNN \g__unravel_output_gtl
      \g__unravel_output_gtl \c_group_begin_gtl
    \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
    \use:n \c_group_end_token
    \use:n \c_group_begin_token
    \int_set:Nn \l__unravel_choice_int { #1 + 1 }
    \gtl_clear:N \l__unravel_after_group_gtl
    \__unravel_print_action:e
      {
        \token_to_str:N #2
        \prg_replicate:nn {#1} { { ... } }
        \iow_char:N \{
      }
  }
```

(*End of definition for* `\__unravel_end_choice_group:NN` *and* `\__unravel_end_choice_group:nN`*.*)

__unravel_off_save:

```
\cs_new_protected:Npn \__unravel_off_save:
  {
    \int_compare:nNnTF \__unravel_currentgrouptype: = { 0 }
      { % bottom-level
        \__unravel_error:neeee { extra-close }
          { \token_to_meaning:N \l__unravel_head_token } { } { } { }
      }
      {
        \__unravel_back_input:
        \int_case:nnF \__unravel_currentgrouptype:
          {
            { 14 } % semi_simple_group
              { \gtl_set:Nn \l__unravel_head_gtl { \group_end: } }
            { 15 } % math_shift_group
              { \gtl_set:Nn \l__unravel_head_gtl { $ } } % $
            { 16 } % math_left_group
              { \gtl_set:Nn \l__unravel_head_gtl { \tex_right:D . } }
          }
          { \gtl_set_eq:NN \l__unravel_head_gtl \c_group_end_gtl }
        \__unravel_back_input:
        \__unravel_error:neeee { off-save }
          { \gtl_to_str:N \l__unravel_head_gtl } { } { } { }
      }
  }
```

(*End of definition for* `\__unravel_off_save:`.)

2.11 Modes

`\__unravel_mode_math:n`
`\__unravel_mode_non_math:n`
`\__unravel_mode_vertical:n`

```
\cs_new_protected:Npn \__unravel_mode_math:n #1
  { \mode_if_math:TF {#1} { \__unravel_insert_dollar_error: } }
\cs_new_protected:Npn \__unravel_mode_non_math:n #1
  { \mode_if_math:TF { \__unravel_insert_dollar_error: } {#1} }
\cs_new_protected:Npn \__unravel_mode_vertical:n #1
  {
    \mode_if_math:TF
      { \__unravel_insert_dollar_error: }
      { \mode_if_horizontal:TF { \__unravel_head_for_vmode: } {#1} }
  }
\cs_new_protected:Npn \__unravel_mode_non_vertical:n #1
  {
    \mode_if_vertical:TF
      { \__unravel_back_input: \__unravel_new_graf:N \c_true_bool }
      {#1}
  }
```

(*End of definition for* `\__unravel_mode_math:n`*,* `\__unravel_mode_non_math:n`*, and* `\__unravel_mode_vertical:n`.)

`\__unravel_head_for_vmode:`

See TeX's `head_for_vmode`.

```
\cs_new_protected:Npn \__unravel_head_for_vmode:
  {
    \mode_if_inner:TF
      {
        \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_hrule:D
          {
            \__unravel_error:nnnnn { hrule-bad-mode } { } { } { } { }
            \__unravel_print_action:
          }
          { \__unravel_off_save: }
      }
      {
        \__unravel_back_input:
        \gtl_set:Nn \l__unravel_head_gtl { \par }
        \__unravel_back_input:
      }
  }
```

(*End of definition for* `\__unravel_head_for_vmode:`.)

`\__unravel_goto_inner_math:`

```
\cs_new_protected:Npn \__unravel_goto_inner_math:
  {
    \__unravel_box_hook:N \tex_everymath:D
    $ % $
    \__unravel_box_hook_end:
  }
```

(*End of definition for* `\__unravel_goto_inner_math:`.)

`\__unravel_goto_display_math:`

```
\cs_new_protected:Npn \__unravel_goto_display_math:
  {
    \__unravel_box_hook:N \tex_everydisplay:D
    $ $
    \__unravel_box_hook_end:
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_after_math:
  {
    \mode_if_inner:TF
      { \int_compare:nNnTF \l__unravel_choice_int > 0 }
      { \use_i:nn }
      {
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \__unravel_get_x_next:
        \token_if_eq_catcode:NNF
          \l__unravel_head_token \c_math_toggle_token
          {
            \__unravel_back_input:
            \tl_set:Nn \l__unravel_head_tl { $ } % $
            \__unravel_error:nnnnn { missing-dollar } { } { } { } { }
          }
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
        $ $
      }
      {
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
        $ % $
      }
    \__unravel_print_action:
  }
```

(*End of 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:

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

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

Not implemented at all: `endv`.

`\relax` does nothing.

```
\__unravel_new_tex_cmd:nn { relax }                                % 0
  {
    \token_if_eq_meaning:NNT \l__unravel_head_token \__unravel_special_relax:
      {
        \exp_after:wN \__unravel_token_if_expandable:NTF \l__unravel_head_tl
          {
            \__unravel_set_action_text:e
              { \iow_char:N \\notexpanded: \g__unravel_action_text_str }
          }
          { }
      }
    \__unravel_print_action:
  }
```

Begin-group characters are sent to the output, as their grouping behaviour may affect the scope of font changes, for instance. They are also performed.

```
\__unravel_new_tex_cmd:nn { begin-group_char }                   % 1
  {
    \gtl_gconcat:NNN \g__unravel_output_gtl
      \g__unravel_output_gtl \c_group_begin_gtl
    \__unravel_print_action:
    \l__unravel_head_token
    \gtl_clear:N \l__unravel_after_group_gtl
  }
```

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

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

```
\__unravel_new_tex_cmd:nn { math_char }                          % 3
  {
    \__unravel_mode_non_vertical:n
      {
        \mode_if_math:TF
          {
            \int_compare:nNnTF
              \__unravel_currentgrouptype: = { 15 } % math_shift_group
              { \__unravel_after_math: }
              { \__unravel_off_save: }
          }
          {
            \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
            \__unravel_get_next:
            \token_if_eq_catcode:NNTF
              \l__unravel_head_token \c_math_toggle_token
              {
                \mode_if_inner:TF
                  { \__unravel_back_input: \__unravel_goto_inner_math: }
                  {
                    \gtl_gput_right:NV
                      \g__unravel_output_gtl \l__unravel_head_tl
                    \__unravel_goto_display_math:
                  }
              }
              { \__unravel_back_input: \__unravel_goto_inner_math: }
          }
      }
  }
```

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

```
\__unravel_new_tex_cmd:nn { alignment_char }                     % 4
  { \l__unravel_head_token \__unravel_print_action: }
\__unravel_new_tex_cmd:nn { car_ret }                            % 5
  { \l__unravel_head_token \__unravel_print_action: }
\__unravel_new_tex_cmd:nn { macro_char }                         % 6
  { \l__unravel_head_token \__unravel_print_action: }
```

```
\__unravel_new_tex_cmd:nn { superscript_char }                     % 7
  { \__unravel_mode_math:n { \__unravel_sub_sup: } }
\__unravel_new_tex_cmd:nn { subscript_char }                       % 8
  { \__unravel_mode_math:n { \__unravel_sub_sup: } }
\cs_new_protected:Npn \__unravel_sub_sup:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_do_one_atom:
  }
\cs_new_protected:Npn \__unravel_do_one_atom:
  {
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_case:nnTF \l__unravel_head_cmd_int
      {
        { \__unravel_tex_use:n { the_char } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { other_char } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { char_given } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { char_num } }
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_int:
          }
        { \__unravel_tex_use:n { math_char_num } }
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_int:
          }
        { \__unravel_tex_use:n { math_given } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { delim_num } }
          { \__unravel_prev_input:V \l__unravel_head_tl \__unravel_scan_int: }
      }
      {
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \tl_use:N \l__unravel_head_tl \scan_stop:
      }
      {
        \__unravel_back_input:
        \__unravel_scan_left_brace:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
        \gtl_gconcat:NNN \g__unravel_output_gtl
          \g__unravel_output_gtl \c_group_begin_gtl
        \tl_use:N \l__unravel_head_tl \c_group_begin_token
      }
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

```
\__unravel_new_tex_cmd:nn { endv }                                 % 9
```

```
  {
    \__unravel_mode_non_math:n
      {
        \__unravel_not_implemented:n { alignments }
      }
  }
```

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

```
\__unravel_new_tex_cmd:nn { blank_char }                          % 10
  {
    \mode_if_horizontal:T
      {
        \gtl_gput_right:Nn \g__unravel_output_gtl { ~ }
        \l__unravel_head_token
      }
    \__unravel_print_action:
  }
```

Letters and other characters leave vertical mode.

```
\__unravel_new_tex_cmd:nn { the_char }                            % 11
  {
    \__unravel_mode_non_vertical:n
      {
        \tl_set:Ne \l__unravel_tmpa_tl
          { ‘ \__unravel_token_to_char:N \l__unravel_head_token }
        \mode_if_math:TF
          { \__unravel_char_in_mmode:V \l__unravel_tmpa_tl }
          { \__unravel_char:V \l__unravel_tmpa_tl }
      }
  }
\__unravel_new_eq_tex_cmd:nn { other_char } { the_char }          % 12
```

```
\__unravel_new_tex_cmd:nn { par_end }                             % 13
  {
    \__unravel_mode_non_math:n
      {
        \mode_if_vertical:TF
          { \__unravel_par: }
          {
            % if align_state<0 then off_save;
            \__unravel_par_if_hmode:
            \mode_if_vertical:T
              { \mode_if_inner:F { \__unravel_build_page: } }
          }
      }
  }
```

```
\__unravel_new_tex_cmd:nn { stop }                                % 14
  {
    \__unravel_mode_vertical:n
      {
        \mode_if_inner:TF
          { \__unravel_forbidden_case: }
          {
            % ^^A todo: unless its_all_over
            \int_gdecr:N \g__unravel_ends_int
            \int_compare:nNnTF \g__unravel_ends_int > 0
              {
                \__unravel_back_input:
                \__unravel_back_input:n
                  {
                    \__unravel_hbox:w to \tex_hsize:D { }
                    \tex_vfill:D
                    \tex_penalty:D - '10000000000 ~
                  }
                \__unravel_build_page:
                \__unravel_print_action:e { End~everything! }
              }
              {
                \__unravel_print_outcome:
                \l__unravel_head_token
              }
          }
      }
  }
\__unravel_new_tex_cmd:nn { delim_num }                            % 15
  {
    \__unravel_mode_math:n
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_int:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \tl_use:N \l__unravel_head_tl \scan_stop:
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
      }
  }
```

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`.

```
\__unravel_new_tex_cmd:nn { char_num }                           % 16
  {
    \__unravel_mode_non_vertical:n
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_int:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \mode_if_math:TF
          { \__unravel_char_in_mmode:e { \tl_tail:N \l__unravel_head_tl } }
          { \__unravel_char:e { \tl_tail:N \l__unravel_head_tl } }
      }
  }
```

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.

```
\__unravel_new_tex_cmd:nn { math_char_num }                      % 17
  {
    \__unravel_mode_math:n
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_int:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_mathchar:e { \tl_tail:N \l__unravel_head_tl }
      }
  }
```

```
\__unravel_new_tex_cmd:nn { mark }                               % 18
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \int_compare:nNnF \l__unravel_head_char_int = 0
      { \__unravel_scan_int: }
    \__unravel_prev_input_gpush:
    \__unravel_scan_toks:NN \c_false_bool \c_true_bool
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_print_action:e
```

```
      { \tl_to_str:N \l__unravel_head_tl \tl_to_str:N \l__unravel_tmpa_tl }
    \tl_put_right:Ne \l__unravel_head_tl
      { { \exp_not:N \exp_not:n \exp_not:V \l__unravel_tmpa_tl } }
    \tl_use:N \l__unravel_head_tl
  }
```

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.

```
\__unravel_new_tex_cmd:nn { xray }                                % 19
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \int_case:nnF \l__unravel_head_char_int
      {
        { 0 }
          { % show
            \__unravel_get_next:
            \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
            \token_if_eq_meaning:NNTF
              \l__unravel_head_token \__unravel_special_relax:
              {
                \exp_after:wN \exp_after:wN \exp_after:wN \l__unravel_tmpa_tl
                \exp_after:wN \exp_not:N \l__unravel_head_tl
              }
              { \gtl_head_do:NN \l__unravel_head_gtl \l__unravel_tmpa_tl }
          }
        { 2 }
          { % showthe
            \__unravel_get_x_next:
            \__unravel_rescan_something_internal:n { 5 }
            \__unravel_prev_input_gpop:N \l__unravel_head_tl
            \exp_args:Ne \use:n % better display than \use:e
              { \tex_showtokens:D { \tl_tail:N \l__unravel_head_tl } }
          }
      }
      { % no operand for showlists, showgroups, showifs
        \int_compare:nNnT \l__unravel_head_char_int = 1 % showbox
          { \__unravel_scan_int: }
        \int_compare:nNnT \l__unravel_head_char_int = 5 % showtokens
          { \__unravel_scan_toks:NN \c_false_bool \c_false_bool }
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \tl_use:N \l__unravel_head_tl \scan_stop:
      }
  }
```

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

```
\__unravel_new_tex_cmd:nn { make_box }                            % 20
  {
    \__unravel_prev_input_gpush:
    \__unravel_back_input:
```

```
    \__unravel_do_box:N \c_false_bool
  }
```

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

```
\cs_new_protected:Npn \__unravel_do_move:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_normal_dimen:
    \__unravel_do_box:N \c_false_bool
  }
```

(*End of definition for* `\__unravel_do_move:`.)

`hmove=21` for `\moveright` and `\moveleft`.

```
\__unravel_new_tex_cmd:nn { hmove }                                 % 21
  {
    \mode_if_vertical:TF
      { \__unravel_do_move: } { \__unravel_forbidden_case: }
  }
```

`vmove=22` for `\lower` and `\raise`.

```
\__unravel_new_tex_cmd:nn { vmove }                                 % 22
  {
    \mode_if_vertical:TF
      { \__unravel_forbidden_case: } { \__unravel_do_move: }
  }
```

`\__unravel_do_unpackage:`

```
\cs_new_protected:Npn \__unravel_do_unpackage:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_int:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

(*End of definition for* `\__unravel_do_unpackage:`.)

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

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

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

```
\__unravel_new_tex_cmd:nn { un_vbox }                               % 24
  {
    \__unravel_mode_vertical:n
      {
        \int_compare:nNnTF \l__unravel_head_char_int > { 1 }
          { \l__unravel_head_token \__unravel_print_action: }
          { \__unravel_do_unpackage: }
      }
  }
```

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

```
\__unravel_new_tex_cmd:nn { remove_item }                         % 25
  { \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.

```
\cs_new_protected:Npn \__unravel_do_append_glue:
  {
    \int_compare:nNnTF \l__unravel_head_char_int < { 4 }
      { \tl_use:N \l__unravel_head_tl \__unravel_print_action: }
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \exp_args:Nf \__unravel_scan_glue:n
          { \int_eval:n { \l__unravel_head_char_int - 2 } }
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \tl_use:N \l__unravel_head_tl \scan_stop:
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
      }
  }
```

(*End of definition for* `\__unravel_do_append_glue:`.)

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

```
\__unravel_new_tex_cmd:nn { hskip }                               % 26
  { \__unravel_mode_non_vertical:n { \__unravel_do_append_glue: } }
```

`vskip=27` for `\vfil`, `\vfill`, `\vss`, `\vfilneg`, `\vskip`.

```
\__unravel_new_tex_cmd:nn { vskip }                               % 27
  { \__unravel_mode_vertical:n { \__unravel_do_append_glue: } }
```

`mskip=28` for `\mskip` (5).

```
\__unravel_new_tex_cmd:nn { mskip }                               % 28
  { \__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.

```
\cs_new_protected:Npn \__unravel_do_append_kern:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_kern:D
      { \__unravel_scan_dimen:nN { 2 } \c_false_bool }
      { \__unravel_scan_dimen:nN { 3 } \c_false_bool }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

(*End of definition for* `\__unravel_do_append_kern:`.)

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

```
\__unravel_new_tex_cmd:nn { kern }                                  % 29
  { \__unravel_do_append_kern: }
```

`mkern`=30 for `\mkern` (99).

```
\__unravel_new_tex_cmd:nn { mkern }                                 % 30
  { \__unravel_mode_math:n { \__unravel_do_append_kern: } }
```

`leader_ship`=31 for `\shipout` (99), `\leaders` (100), `\cleaders` (101), `\xleaders` (102).

```
\__unravel_new_tex_cmd:nn { leader_ship }                           % 31
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \tl_if_head_eq_meaning:VNTF \l__unravel_head_tl \tex_shipout:D
      { \__unravel_do_box:N \c_false_bool }
      { \__unravel_do_box:N \c_true_bool }
  }
```

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

```
\__unravel_new_tex_cmd:nn { halign }                                % 32
  { \__unravel_not_implemented:n { halign } }
\__unravel_new_tex_cmd:nn { valign }                                % 33
  { \__unravel_not_implemented:n { valign } }
\__unravel_new_tex_cmd:nn { no_align }                              % 34
  { \l__unravel_head_token \__unravel_print_action: }
```

```
\__unravel_new_tex_cmd:nn { vrule }                                % 35
  { \__unravel_mode_non_vertical:n { \__unravel_do_rule: } }
\__unravel_new_tex_cmd:nn { hrule }                                % 36
  { \__unravel_mode_vertical:n { \__unravel_do_rule: } }
\cs_new_protected:Npn \__unravel_do_rule:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_alt_rule:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
\__unravel_new_tex_cmd:nn { insert }                               % 37
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_int:
    \__unravel_begin_insert_or_adjust:
  }
\__unravel_new_tex_cmd:nn { vadjust }                              % 38
  {
    \mode_if_vertical:TF
      { \__unravel_forbidden_case: }
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_keyword:nTF { pPrReE }
        \__unravel_begin_insert_or_adjust:
      }
  }
\cs_new_protected:Npn \__unravel_begin_insert_or_adjust:
  {
    \__unravel_scan_left_brace:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \gtl_gconcat:NNN \g__unravel_output_gtl
      \g__unravel_output_gtl \c_group_begin_gtl
    \tl_use:N \l__unravel_head_tl \c_group_begin_token
    \__unravel_print_action:e
      { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
  }
\__unravel_new_tex_cmd:nn { ignore_spaces }                        % 39
  {
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ignorespaces:D
      {
        \__unravel_print_action:
        \__unravel_get_x_non_blank:
        \__unravel_set_cmd:
        \__unravel_do_step:
      }
      { \__unravel_not_implemented:n { pdfprimitive } }
  }
```

```
\__unravel_new_tex_cmd:nn { after_assignment }                     % 40
  {
    \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
    \__unravel_get_next:
    \gtl_gset_eq:NN \g__unravel_after_assignment_gtl \l__unravel_head_gtl
    \__unravel_print_action:e
      {
        Afterassignment:~\tl_to_str:N \l__unravel_tmpa_tl
        \gtl_to_str:N \l__unravel_head_gtl
      }
  }
```

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.

```
\__unravel_new_tex_cmd:nn { after_group }                          % 41
  {
    \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_head_tl
    \__unravel_get_next:
    \int_compare:nNnTF \__unravel_currentgrouptype: = 0
      {
        \__unravel_print_action:e
          {
            Aftergroup~(level~0~=>~dropped):~
            \tl_to_str:N \l__unravel_tmpa_tl
            \gtl_to_str:N \l__unravel_head_gtl
          }
      }
      {
        \gtl_concat:NNN \l__unravel_after_group_gtl
          \l__unravel_after_group_gtl \l__unravel_head_gtl
        \__unravel_print_action:e
          {
            Aftergroup:~\tl_to_str:N \l__unravel_tmpa_tl
            \gtl_to_str:N \l__unravel_head_gtl
          }
      }
  }
```

See `\__unravel_do_append_glue:`.

```
\__unravel_new_tex_cmd:nn { break_penalty }                        % 42
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_int:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

```
\__unravel_new_tex_cmd:nn { start_par }                            % 43
  {
    \mode_if_vertical:TF
      {
        \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noindent:D
          { \__unravel_new_graf:N \c_false_bool }
          { \__unravel_new_graf:N \c_true_bool }
      }
      {
        \int_compare:nNnT \l__unravel_head_char_int = { 1 } % indent
          {
            \__unravel_hbox:w width \tex_parindent:D { }
            \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
          }
        \__unravel_print_action:
      }
  }
```

```
\__unravel_new_tex_cmd:nn { ital_corr }                            % 44
  {
    \mode_if_vertical:TF { \__unravel_forbidden_case: }
      { \l__unravel_head_token \__unravel_print_action: }
  }
```

`\__unravel_do_accent:`

```
\cs_new_protected:Npn \__unravel_do_accent:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_int:
    \__unravel_do_assignments:
    \bool_if:nTF
      {
        \token_if_eq_catcode_p:NN
          \l__unravel_head_token \c_catcode_letter_token
        ||
        \token_if_eq_catcode_p:NN
          \l__unravel_head_token \c_catcode_other_token
        ||
        \int_compare_p:nNn
          \l__unravel_head_cmd_int = { \__unravel_tex_use:n { char_given } }
      }
      { \__unravel_prev_input:V \l__unravel_head_tl }
      {
        \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_char:D
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_int:
          }
          { \__unravel_break:w }
      }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_break_point:
  }
```

(*End of 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).

```
\cs_new_protected:Npn \__unravel_do_math_accent:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_int:
    \__unravel_do_one_atom:
  }
```

(*End of definition for* `\__unravel_do_math_accent:`*.*)

```
\__unravel_new_tex_cmd:nn { accent }                                    % 45
  {
    \__unravel_mode_non_vertical:n
      {
        \mode_if_math:TF
          { \__unravel_do_math_accent: } { \__unravel_do_accent: }
      }
  }
\__unravel_new_tex_cmd:nn { math_accent }                               % 46
  { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
\__unravel_new_tex_cmd:nn { discretionary }                             % 47
  {
    \__unravel_mode_non_vertical:n
      {
        \int_compare:nNnTF \l__unravel_head_char_int = { 1 }
          { \__unravel_output_head_token: }
          { \__unravel_do_choice: }
      }
  }
```

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`

```
\__unravel_new_tex_cmd:nn { eq_no }                               % 48
  {
    \mode_if_math:TF
      {
        \mode_if_inner:TF
          { \__unravel_off_save: }
          {
            \int_compare:nNnTF \tex_currentgrouptype:D = { 15 }
              {
                \__unravel_box_hook:N \tex_everymath:D
                \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
                \l__unravel_head_token
                \__unravel_box_hook_end:
                \int_set:Nn \l__unravel_choice_int { 1 }
              }
              { \__unravel_off_save: }
          }
      }
      { \__unravel_forbidden_case: }
  }
\__unravel_new_tex_cmd:nn { left_right }                          % 49
  {
    \__unravel_mode_math:n
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_delimiter:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_left:D
          {
            \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
            \tl_use:N \l__unravel_head_tl \scan_stop:
            \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
          }
          {
            \int_case:nnF \tex_currentgrouptype:D
              {
                { 16 }
                  {
                    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
                    \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
                    \tl_if_head_eq_meaning:nNTF \l__unravel_head_tl \tex_middle:D
                      {
                        \tl_use:N \l__unravel_head_tl \scan_stop:
                        \gtl_clear:N \l__unravel_after_group_gtl
                      }
                      { \tl_use:N \l__unravel_head_tl \scan_stop: }
                    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
                  }
                { 15 }
                  { % todo: this is a TeX error
                    \tl_use:N \l__unravel_head_tl \scan_stop:
                  }
              }
            { \__unravel_off_save: }
        }
    }
  }
\cs_new_protected:Npn \__unravel_scan_delimiter:
  {
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_case:nnF \l__unravel_head_cmd_int
      {
        { \__unravel_tex_use:n { the_char } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { other_char } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
        { \__unravel_tex_use:n { delim_num } }
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \__unravel_scan_int:
          }
      }
      {
        \__unravel_back_input:
        \__unravel_tex_error:nV { missing-delim } \l__unravel_head_tl
        \__unravel_prev_input:n { . }
      }
  }
```

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

```
\__unravel_new_tex_cmd:nn { limit_switch }                          % 51
  { \__unravel_mode_math:n { \__unravel_output_head_token: } }
\cs_new_protected:Npn \__unravel_output_head_token:
  {
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \l__unravel_head_token
    \__unravel_print_action:
  }
```

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

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

```
\__unravel_new_tex_cmd:nn { math_choice }                           % 54
  { \__unravel_mode_math:n { \__unravel_do_choice: } }
\cs_new_protected:Npn \__unravel_do_choice:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_left_brace:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \gtl_gconcat:NNN \g__unravel_output_gtl
      \g__unravel_output_gtl \c_group_begin_gtl
    \tl_use:N \l__unravel_head_tl \c_group_begin_token
    \gtl_clear:N \l__unravel_after_group_gtl
    \int_set:Nn \l__unravel_choice_int { 1 }
    \__unravel_print_action:e
      { \tl_to_str:N \l__unravel_head_tl \iow_char:N \{ }
  }
```

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

```
\__unravel_new_tex_cmd:nn { vcenter }                                % 56
  { \__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`

```
\__unravel_new_tex_cmd:nn { case_shift }                             % 57
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_scan_toks:NN \c_false_bool \c_false_bool
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \exp_after:wN \__unravel_case_shift:Nn \l__unravel_tmpa_tl
  }
\cs_new_protected:Npn \__unravel_case_shift:Nn #1#2
  {
    #1 { \__unravel_back_input:n {#2} }
    \__unravel_print_action:e
      { \token_to_meaning:N #1 ~ \tl_to_str:n { {#2} } }
  }
```

```
\__unravel_new_tex_cmd:nn { message }                              % 58
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_toks_to_str:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

Extensions are implemented in a later section.

```
\__unravel_new_tex_cmd:nn { extension }                            % 59
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_scan_extension_operands:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
\__unravel_new_tex_cmd:nn { in_stream }                            % 60
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_action:
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_openin:D
      {
        \__unravel_scan_int:
        \__unravel_scan_optional_equals:
        \__unravel_scan_file_name:
      }
      { \__unravel_scan_int: }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
  }
\__unravel_new_tex_cmd:nn { begin_group }                          % 61
  {
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \l__unravel_head_token
    \gtl_clear:N \l__unravel_after_group_gtl
    \__unravel_print_action:
  }
\__unravel_new_tex_cmd:nn { end_group }                            % 62
  {
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \__unravel_back_input_gtl:N \l__unravel_after_group_gtl
    \l__unravel_head_token
    \__unravel_print_action:
  }
\__unravel_new_tex_cmd:nn { omit }                                 % 63
  { \l__unravel_head_token \__unravel_print_action: }
\__unravel_new_tex_cmd:nn { ex_space }                             % 64
  {
    \__unravel_mode_non_vertical:n
      { \l__unravel_head_token \__unravel_print_action: }
  }
\__unravel_new_tex_cmd:nn { no_boundary }                          % 65
  {
    \__unravel_mode_non_vertical:n
      { \l__unravel_head_token \__unravel_print_action: }
  }
\__unravel_new_tex_cmd:nn { radical }                              % 66
  { \__unravel_mode_math:n { \__unravel_do_math_accent: } }
\__unravel_new_tex_cmd:nn { end_cs_name }                          % 67
  {
    \__unravel_tex_error:nV { extra-endcsname } \l__unravel_head_tl
    \__unravel_print_action:
  }
```

See `the_char` and `other_char`.

```
\__unravel_new_tex_cmd:nn { char_given }                           % 68
  {
    \__unravel_mode_non_vertical:n
      {
        \mode_if_math:TF
          { \__unravel_char_in_mmode:V \l__unravel_head_char_int }
          { \__unravel_char:V \l__unravel_head_char_int }
      }
  }
```

See `math_char_num`.

```
\__unravel_new_tex_cmd:nn { math_given }                           % 69
  {
    \__unravel_mode_math:n
      { \__unravel_mathchar:e { \int_use:N \l__unravel_head_char_int } }
  }
\__unravel_new_tex_cmd:nn { last_item }                            % 70
  { \__unravel_forbidden_case: }
```

2.12.6 Extensions

__unravel_scan_extension_operands:

```
\cs_new_protected:Npn \__unravel_scan_extension_operands:
  {
    \int_case:nnF \l__unravel_head_char_int
      {
        { 0 } % openout
          {
            \__unravel_scan_int:
            \__unravel_scan_optional_equals:
            \__unravel_scan_file_name:
          }
        { 1 } % write
          {
            \__unravel_scan_int:
            \__unravel_scan_toks:NN \c_false_bool \c_false_bool
          }
        { 2 } % closeout
          { \__unravel_scan_int: }
        { 3 } % special
          { \__unravel_scan_toks_to_str: }
        { 4 } % immediate
          { \__unravel_scan_immediate_operands: }
        { 5 } % setlanguage
          {
            \mode_if_horizontal:TF
              { \__unravel_scan_int: }
              { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
          }
        { 6 } % pdfliteral
          {
            \__unravel_scan_keyword:nF { dDiIrReEcCtT }
              { \__unravel_scan_keyword:n { pPaAgGeE } }
            \__unravel_scan_pdf_ext_toks:
          }
        { 7 } % pdfobj
          {
            \__unravel_scan_keyword:nTF
              { rReEsSeErRvVeEoObBjJnNuUmM }
              { \__unravel_skip_optional_space: }
              {
                \__unravel_scan_keyword:nF { uUsSeEoObBjJnNuUmM }
                  { \__unravel_scan_int: }
                \__unravel_scan_keyword:nT { sStTrReEaAmM }
                  {
                    \__unravel_scan_keyword:nT { aAtTtTrR }
                      { \__unravel_scan_pdf_ext_toks: }
                  }
                \__unravel_scan_keyword:n { fFiIlLeE }
                \__unravel_scan_pdf_ext_toks:
              }
          }
        { 8 } % pdfrefobj
          { \__unravel_scan_int: }
        { 9 } % pdfxform
          {
            \__unravel_scan_keyword:nT { aAtTtTrR }
              { \__unravel_scan_pdf_ext_toks: }
            \__unravel_scan_keyword:nTF { rReEsSoOuUrRcCeEsS }
              { \__unravel_scan_pdf_ext_toks: }
            \__unravel_scan_int:
          }
        { 10 } % pdfrefxform
          { \__unravel_scan_int: }
        { 11 } % pdfximage
          { \__unravel_scan_image: }
        { 12 } % pdfrefximage
          { \__unravel_scan_int: }
        { 13 } % pdfannot
          {
            \__unravel_scan_keyword:nTF
              { rReEsSeErRvVeEoObBjJnNuUmM }
              { \__unravel_scan_optional_space: }
              {
                \__unravel_scan_keyword:nT { uUsSeEoObBjJnNuUmM }
                  { \__unravel_scan_int: }
                \__unravel_scan_alt_rule:
                \__unravel_scan_pdf_ext_toks:
              }
          }
        { 14 } % pdfstartlink
          {
            \mode_if_vertical:TF
              { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
              {
                \__unravel_scan_rule_attr:
                \__unravel_scan_action:
              }
          }
        { 15 } % pdfendlink
          {
            \mode_if_vertical:T
              { \__unravel_error:nnnnn { invalid-mode } { } { } { } { } }
          }
        { 16 } % pdfoutline
          {
            \__unravel_scan_keyword:nT { aAtTtTrR }
              { \__unravel_scan_pdf_ext_toks: }
            \__unravel_scan_action:
            \__unravel_scan_keyword:nT { cCoOuUnNtT }
              { \__unravel_scan_int: }
            \__unravel_scan_pdf_ext_toks:
          }
        { 17 } % pdfdest
          { \__unravel_scan_pdfdest_operands: }
        { 18 } % pdfthread
          { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
        { 19 } % pdfstartthread
          { \__unravel_scan_rule_attr: \__unravel_scan_thread_id: }
        { 20 } % pdfendthread
          { }
        { 21 } % pdfsavepos
          { }
        { 22 } % pdfinfo
          { \__unravel_scan_pdf_ext_toks: }
        { 23 } % pdfcatalog
          {
            \__unravel_scan_pdf_ext_toks:
            \__unravel_scan_keyword:n { oOpPeEnNaAcCtTiIoOnN }
              { \__unravel_scan_action: }
          }
        { 24 } % pdfnames
          { \__unravel_scan_pdf_ext_toks: }
        { 25 } % pdffontattr
```

```
          {
            \__unravel_scan_font_ident:
            \__unravel_scan_pdf_ext_toks:
          }
        { 26 } % pdfincludechars
          {
            \__unravel_scan_font_ident:
            \__unravel_scan_pdf_ext_toks:
          }
        { 27 } % pdfmapfile
          { \__unravel_scan_pdf_ext_toks: }
        { 28 } % pdfmapline
          { \__unravel_scan_pdf_ext_toks: }
        { 29 } % pdftrailer
          { \__unravel_scan_pdf_ext_toks: }
        { 30 } % pdfresettimer
          { }
        { 31 } % pdffontexpand
          {
            \__unravel_scan_font_ident:
            \__unravel_scan_optional_equals:
            \__unravel_scan_int:
            \__unravel_scan_int:
            \__unravel_scan_int:
            \__unravel_scan_keyword:nT { aAuUtToOeExXpPaAnNdD }
              { \__unravel_skip_optional_space: }
          }
        { 32 } % pdfsetrandomseed
          { \__unravel_scan_int: }
        { 33 } % pdfsnaprefpoint
          { }
        { 34 } % pdfsnapy
          { \__unravel_scan_normal_glue: }
        { 35 } % pdfsnapycomp
          { \__unravel_scan_int: }
        { 36 } % pdfglyphtounicode
          {
            \__unravel_scan_pdf_ext_toks:
            \__unravel_scan_pdf_ext_toks:
          }
        { 37 } % pdfcolorstack
          { \__unravel_scan_pdfcolorstack_operands: }
        { 38 } % pdfsetmatrix
          { \__unravel_scan_pdf_ext_toks: }
        { 39 } % pdfsave
          { }
        { 40 } % pdfrestore
          { }
        { 41 } % pdfnobuiltintounicode
          { \__unravel_scan_font_ident: }
      }
      { } % no other cases.
  }
```

(*End of definition for* `\__unravel_scan_extension_operands:`.)

__unravel_scan_pdfcolorstack_operands:

```
\cs_new_protected:Npn \__unravel_scan_pdfcolorstack_operands:
  {
    \__unravel_scan_int:
    \__unravel_scan_keyword:nF { sSeEtT }
      {
        \__unravel_scan_keyword:nF { pPuUsShH }
          {
            \__unravel_scan_keyword:nF { pPoOpP }
              {
                \__unravel_scan_keyword:nF { cCuUrRrReEnNtT }
                  {
                    \__unravel_error:nnnnn { color-stack-action-missing }
                      { } { } { } { }
                  }
              }
          }
      }
  }
```

(*End of definition for* `\__unravel_scan_pdfcolorstack_operands:`.)

__unravel_scan_rule_attr:

```
\cs_new_protected:Npn \__unravel_scan_rule_attr:
  {
    \__unravel_scan_alt_rule:
    \__unravel_scan_keyword:nT { aAtTtTrR }
      { \__unravel_scan_pdf_ext_toks: }
  }
```

(*End of definition for* `\__unravel_scan_rule_attr:`.)

__unravel_scan_action:

```
\cs_new_protected:Npn \__unravel_scan_action:
  {
    \__unravel_scan_keyword:nTF { uUsSeErR }
      { \__unravel_scan_pdf_ext_toks: }
      {
        \__unravel_scan_keyword:nF { gGoOtToO }
          {
            \__unravel_scan_keyword:nF { tThHrReEaAdD }
              { \__unravel_error:nnnnn { action-type-missing } { } { } { } { } }
          }
      }
    \__unravel_scan_keyword:nT { fFiIlLeE }
      { \__unravel_scan_pdf_ext_toks: }
    \__unravel_scan_keyword:nTF { pPaAgGeE }
      {
        \__unravel_scan_int:
        \__unravel_scan_pdf_ext_toks:
      }
      {
        \__unravel_scan_keyword:nTF { nNaAmMeE }
          { \__unravel_scan_pdf_ext_toks: }
          {
            \__unravel_scan_keyword:nTF { nNuUmM }
              { \__unravel_scan_int: }
              { \__unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
          }
      }
    \__unravel_scan_keyword:nTF { nNeEwWwWiInNdDoOwW }
      { \__unravel_skip_optional_space: }
      {
        \__unravel_scan_keyword:nT { nNoOnNeEwWwWiInNdDoOwW }
          { \__unravel_skip_optional_space: }
      }
  }
```

(*End of definition for* `\__unravel_scan_action:`*.*)

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

```
\cs_new_protected:Npn \__unravel_scan_image:
  {
    \__unravel_scan_rule_attr:
    \__unravel_scan_keyword:nTF { nNaAmMeEdD }
      { \__unravel_scan_pdf_ext_toks: }
      {
        \__unravel_scan_keyword:nT { pPaAgGeE }
          { \__unravel_scan_int: }
      }
    \__unravel_scan_keyword:nT { cCoOlLoOrRsSpPaAcCeE }
      { \__unravel_scan_int: }
    \__unravel_scan_pdf_ext_toks:
  }
```

(*End of definition for* `\__unravel_scan_image:`*.*)

`\__unravel_scan_immediate_operands:`

```
\cs_new_protected:Npn \__unravel_scan_immediate_operands:
  {
    \__unravel_get_x_next:
    \__unravel_set_cmd:
    \int_compare:nNnTF
      \l__unravel_head_cmd_int = { \__unravel_tex_use:n { extension } }
      {
        \int_compare:nNnTF
          \l__unravel_head_char_int < { 3 } % openout, write, closeout
          { \__unravel_scan_immediate_operands_aux: }
          {
            \int_case:nnF \l__unravel_head_char_int
              {
                { 7 } { \__unravel_scan_extension_operands_aux: } % pdfobj
                { 9 }
                  {
                    \__unravel_prepare_mag:
                    \__unravel_scan_extension_operands_aux:
                  } % pdfxform
                { 11 } { \__unravel_scan_extension_operands_aux: } %pdfximage
              }
              { \__unravel_scan_immediate_operands_bad: }
          }
      }
      { \__unravel_scan_immediate_operands_bad: }
  }
\cs_new_protected:Npn \__unravel_scan_immediate_operands_aux:
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_scan_extension_operands:
  }
\cs_new_protected:Npn \__unravel_scan_immediate_operands_bad:
  {
    \__unravel_back_input:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl ignored }
    \__unravel_prev_input_gpush:
  }

```

(*End of definition for* `\__unravel_scan_immediate_operands:`.)

__unravel_scan_pdfdest_operands:

```
\cs_new_protected:Npn \__unravel_scan_pdfdest_operands:
  {
    \__unravel_scan_keyword:nTF { nNuUmM }
      { \__unravel_scan_int: }
      {
        \__unravel_scan_keyword:nTF { nNaAmMeE }
          { \__unravel_scan_pdf_ext_toks: }
          { \__unravel_error:nnnnn { identifier-type-missing } { } { } { } { } }
      }
    \__unravel_scan_keyword:nTF { xXyYzZ }
      {
        \__unravel_scan_keyword:nT { zZoOoOmM }
          { \__unravel_scan_int: }
      }
      {
        \__unravel_scan_keyword:nF { fFiItTbBhH }
          {
            \__unravel_scan_keyword:nF { fFiItTbBvV }
              {
                \__unravel_scan_keyword:nF { fFiItTbB }
                  {
                    \__unravel_scan_keyword:nF { fFiItThHhH }
                      {
                        \__unravel_scan_keyword:nF { fFiItTvV }
                          {
                            \__unravel_scan_keyword:nTF
                              { fFiItTrR }
                              {
                                \__unravel_skip_optional_space:
                                \__unravel_scan_alt_rule:
                                \use_none:n
                              }
                              {
                                \__unravel_scan_keyword:nF
                                  { fFiItT }
                                  {
                                    \__unravel_error:nnnnn { destination-type-missing }
                                      { } { } { } { }
                                  }
                              }
                          }
                      }
                  }
              }
          }
      }
    \__unravel_skip_optional_space:
  }
```

(*End of 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`.

```
\cs_set_protected:Npn \__unravel_tmp:w
  {
    \__unravel_prev_input_gpush:
    \__unravel_prefixed_command:
  }
\int_step_inline:nnnn
  { \__unravel_tex_use:n { max_non_prefixed_command } + 1 }
  { 1 }
  { \__unravel_tex_use:n { max_command } }
  { \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.

```
\cs_new_protected:Npn \__unravel_prefixed_command:
  {
    \int_while_do:nNnn
      \l__unravel_head_cmd_int = { \__unravel_tex_use:n { prefix } }
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \__unravel_get_x_non_relax:
        \__unravel_set_cmd:
        \int_compare:nNnF \l__unravel_head_cmd_int
          > { \__unravel_tex_use:n { max_non_prefixed_command } }
          {
            \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
            \__unravel_error:neeee { erroneous-prefixes }
              { \tl_to_str:N \l__unravel_tmpa_tl }
              { \tl_to_str:N l__unravel_head_tl }
              { } { }
            \__unravel_back_input:
            \__unravel_omit_after_assignment:w
          }
      }
    % ^^A todo: Discard non-\global prefixes if they are irrelevant
    % ^^A todo: Adjust for the setting of \globaldefs
    \cs_if_exist_use:cF
      { __unravel_prefixed_ \int_use:N \l__unravel_head_cmd_int : }
      {
        \__unravel_error:nnnnn { internal } { prefixed } { } { } { }
        \__unravel_omit_after_assignment:w
      }
    \__unravel_after_assignment:
  }
```

(*End of 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`

```
\cs_new_protected:Npn \__unravel_after_assignment:
  {
    \__unravel_back_input_gtl:N \g__unravel_after_assignment_gtl
    \gtl_gclear:N \g__unravel_after_assignment_gtl
  }
\cs_new_protected:Npn \__unravel_omit_after_assignment:w
    #1 \__unravel_after_assignment: { }
```

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

`\__unravel_prefixed_new:nn`

```
\cs_new_protected:Npn \__unravel_prefixed_new:nn #1#2
  {
    \cs_new_protected:cpn
      { __unravel_prefixed_ \__unravel_tex_use:n {#1} : } {#2}
  }
```

(*End of definition for* `\__unravel_prefixed_new:nn`*.*)

`\__unravel_assign_token:n`

```
\cs_new_protected:Npn \__unravel_assign_token:n #1
  {
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    #1
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_assigned_token:
  }
```

(*End of definition for* `\__unravel_assign_token:n`*.*)

`\__unravel_assign_register:`

```
\cs_new_protected:Npn \__unravel_assign_register:
  {
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_assigned_register:
  }
```

(*End of definition for* `\__unravel_assign_register:`*.*)

`\__unravel_assign_value:nn`

```
\cs_new_protected:Npn \__unravel_assign_value:nn #1#2
  {
    \tl_if_empty:nF {#1}
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
        #1
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
      }
    \__unravel_prev_input:V \l__unravel_head_tl
    \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
    \__unravel_scan_optional_equals:
    #2
    \__unravel_assign_register:
  }
```

(*End of definition for* `\__unravel_assign_value:nn`*.*)

`\__unravel_assign_toks:`

```
\__unravel_prefixed_new:nn { toks_register }                      % 71
  {
    \int_compare:nNnT \l__unravel_head_char_int = 0
      { % \toks
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_action:
        \__unravel_scan_int:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
      }
    \__unravel_assign_toks:
  }
\__unravel_prefixed_new:nn { assign_toks }                        % 72
  { \__unravel_assign_toks: }
\cs_new_protected:Npn \__unravel_assign_toks:
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_print_action:
    \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
    \__unravel_scan_optional_equals:
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_compare:nNnTF
      \l__unravel_head_cmd_int = { \__unravel_tex_use:n { toks_register } }
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \int_compare:nNnT \l__unravel_head_char_int = 0
          { \__unravel_scan_int: }
      }
      {
        \int_compare:nNnTF
          \l__unravel_head_cmd_int = { \__unravel_tex_use:n { assign_toks } }
          { \__unravel_prev_input:V \l__unravel_head_tl }
          {
            \__unravel_back_input:
            \__unravel_scan_toks:NN \c_false_bool \c_false_bool
          }
      }
    \__unravel_assign_register:
  }
```

(*End of definition for* `\__unravel_assign_toks:`*.*)

```
\__unravel_prefixed_new:nn { assign_int }                           % 73
  { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
\__unravel_prefixed_new:nn { assign_dimen }                         % 74
  { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
\__unravel_prefixed_new:nn { assign_glue }                          % 75
  { \__unravel_assign_value:nn { } { \__unravel_scan_normal_glue: } }
\__unravel_prefixed_new:nn { assign_mu_glue }                       % 76
  { \__unravel_assign_value:nn { } { \__unravel_scan_mu_glue: } }
\__unravel_prefixed_new:nn { assign_font_dimen }                    % 77
  {
    \__unravel_assign_value:nn
      { \__unravel_scan_int: \__unravel_scan_font_ident: }
      { \__unravel_scan_normal_dimen: }
  }
\__unravel_prefixed_new:nn { assign_font_int }                      % 78
  {
    \__unravel_assign_value:nn
      { \__unravel_scan_font_int: } { \__unravel_scan_int: }
  }
\__unravel_prefixed_new:nn { set_aux }                              % 79
  { % prevdepth = 1, spacefactor = 102
    \int_compare:nNnTF \l__unravel_head_char_int = 1
      { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
      { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
  }
\__unravel_prefixed_new:nn { set_prev_graf }                        % 80
  { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
\__unravel_prefixed_new:nn { set_page_dimen }                       % 81
  { \__unravel_assign_value:nn { } { \__unravel_scan_normal_dimen: } }
\__unravel_prefixed_new:nn { set_page_int }                         % 82
  { \__unravel_assign_value:nn { } { \__unravel_scan_int: } }
\__unravel_prefixed_new:nn { set_box_dimen }                        % 83
  {
    \__unravel_assign_value:nn
      { \__unravel_scan_int: } { \__unravel_scan_normal_dimen: }
  }
```

This is a variant of `\__unravel_assign_value:nn`, with a bit more complication because the syntax of `\parshape` and of ε-TeX primitives such as `\interlinepenalties` is a bit different.

```
\__unravel_prefixed_new:nn { set_shape }                            % 84
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
    \tl_if_head_eq_meaning:VNTF \l__unravel_defined_tl \tex_parshape:D
      {
        \__unravel_set_shape:NN 2 \__unravel_scan_normal_dimen:
        \__unravel_print_assigned_parshape:
      }
      {
        \__unravel_set_shape:NN 1 \__unravel_scan_int:
        \__unravel_print_assigned_set_shape:
      }
  }
\cs_new_protected:Npn \__unravel_set_shape:NN #1#2
  {
    \__unravel_scan_optional_equals:
    \__unravel_prev_input_gpush:
    \__unravel_scan_int:
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_prev_input_silent:V \l__unravel_tmpa_tl
    \prg_replicate:nn
      { \int_max:nn { 0 } { #1 * \l__unravel_tmpa_tl } }
      { \__unravel_prev_input_silent:n { ~ } #2 }
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \tl_use:N \l__unravel_tmpa_tl \scan_stop:
  }
\__unravel_prefixed_new:nn { def_code }                             % 85
  {
    \__unravel_assign_value:nn
      { \__unravel_scan_int: } { \__unravel_scan_int: }
  }
\__unravel_prefixed_new:nn { def_family }                           % 86
  {
    \__unravel_assign_value:nn
      { \__unravel_scan_int: } { \__unravel_scan_font_ident: }
  }
\__unravel_prefixed_new:nn { set_font }                             % 87
  {
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \gtl_gput_right:NV \g__unravel_output_gtl \l__unravel_head_tl
    \__unravel_print_action:
  }
\__unravel_prefixed_new:nn { def_font }                             % 88
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_scan_r_token:
    \__unravel_print_action:e
      { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
    \__unravel_scan_optional_equals:
    \__unravel_scan_file_name:
    \bool_gset_true:N \g__unravel_name_in_progress_bool
    \__unravel_scan_keyword:nTF { aAtT }
```

```
      { \__unravel_scan_normal_dimen: }
      {
        \__unravel_scan_keyword:nT { sScCaAlLeEdD }
          { \__unravel_scan_int: }
      }
    \bool_gset_false:N \g__unravel_name_in_progress_bool
    \__unravel_assign_token:n { }
  }
```

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

`let`, `futurelet`

```
\__unravel_prefixed_new:nn { let }                                 % 94
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_let:D
      { % |let|
        \__unravel_scan_r_token:
        \__unravel_prev_input_get:N \l__unravel_tmpa_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
        \__unravel_get_next:
        \bool_while_do:nn
          { \token_if_eq_catcode_p:NN \l__unravel_head_token \c_space_token }
          { \__unravel_get_next: }
        \tl_if_eq:NNT \l__unravel_head_tl \c__unravel_eq_tl
          { \__unravel_get_next: }
        \token_if_eq_catcode:NNT \l__unravel_head_token \c_space_token
          { \__unravel_get_next: }
      }
      { % |futurelet|
        \__unravel_scan_r_token:
        \__unravel_prev_input_get:N \l__unravel_tmpa_tl
        \__unravel_print_action:e { \tl_to_str:N \l__unravel_tmpa_tl }
        \__unravel_get_next:
        \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
        \__unravel_get_next:
        \__unravel_back_input:
        \gtl_set_eq:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
        \__unravel_back_input:
      }
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \tl_put_right:Nn \l__unravel_tmpa_tl { = ~ \l__unravel_head_token }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \use:e
      {
        \exp_not:V \l__unravel_head_tl
        \tex_let:D \tl_tail:N \l__unravel_tmpa_tl
      }
    \__unravel_print_assigned_token:
  }
```

```
\__unravel_prefixed_new:nn { shorthand_def }                       % 95
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \tl_set:Ne \l__unravel_prev_action_tl
```

```
      { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_scan_r_token:
    \__unravel_print_action:e
      { \l__unravel_prev_action_tl \tl_to_str:N \l__unravel_defined_tl }
    \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \scan_stop:
    \__unravel_just_print_assigned_token:
    \__unravel_scan_optional_equals:
    \__unravel_scan_int:
    \__unravel_assign_token:n { }
  }
```

`\__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.

```
\__unravel_prefixed_new:nn { read_to_cs }                         % 96
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_print_action:e { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_scan_int:
    \__unravel_scan_to:
    \__unravel_scan_r_token:
    \__unravel_prev_input_get:N \l__unravel_tmpa_tl
    \__unravel_read_to_cs_safe:fTF
      { \__unravel_tl_first_int:N \l__unravel_tmpa_tl }
      { \__unravel_assign_token:n { } }
      {
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_tex_fatal_error:nV { cannot-read } \l__unravel_head_tl
      }
  }
\prg_new_conditional:Npnn \__unravel_read_to_cs_safe:n #1 { TF }
  {
    \int_compare:nNnTF { \tex_interactionmode:D } > { 1 }
      { \prg_return_true: }
      {
        \int_compare:nNnTF {#1} < { 0 }
          { \prg_return_false: }
          {
            \int_compare:nNnTF {#1} > { 15 }
              { \prg_return_false: }
              {
                \tex_ifeof:D #1 \exp_stop_f:
                  \prg_return_false:
                \else:
                  \prg_return_true:
                \fi:
              }
          }
      }
  }
\cs_generate_variant:Nn \__unravel_read_to_cs_safe:nTF { f }
```

(*End of definition for* `\__unravel_read_to_cs_safe:nTF`*.*)

```
\__unravel_prefixed_new:nn { def }                                   % 97
  {
    \__unravel_prev_input_get:N \l__unravel_tmpa_tl
    \tl_set:NV \l__unravel_defining_tl \l__unravel_tmpa_tl
    \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \int_compare:nNnTF \l__unravel_head_char_int < 2
      { % def/gdef
        \__unravel_scan_r_token:
        \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
        \__unravel_scan_toks:NN \c_true_bool \c_false_bool
      }
      { % edef/xdef
        \__unravel_scan_r_token:
        \tl_put_right:NV \l__unravel_defining_tl \l__unravel_defined_tl
        \__unravel_scan_toks:NN \c_true_bool \c_true_bool
      }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_assign_token:n
      { \tl_set_eq:NN \l__unravel_head_tl \l__unravel_defining_tl }
  }
```

`\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.

```
\__unravel_prefixed_new:nn { set_box }                               % 98
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_scan_int:
    \__unravel_scan_optional_equals:
    \bool_if:NTF \g__unravel_set_box_allowed_bool
      { \__unravel_do_box:N \c_false_bool }
      {
        \__unravel_error:nnnnn { improper-setbox } { } { } { } { }
        \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
        \__unravel_omit_after_assignment:w
      }
  }
```

`\hyphenation` and `\patterns`

```
\__unravel_prefixed_new:nn { hyph_data }                             % 99
  {
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_scan_toks:NN \c_false_bool \c_false_bool
    \__unravel_assign_token:n { }
  }
```

```
\__unravel_prefixed_new:nn { set_interaction }                       % 100
  {
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \tl_put_left:NV \l__unravel_head_tl \l__unravel_tmpa_tl
    \tl_use:N \l__unravel_head_tl \scan_stop:
    \__unravel_print_assignment:e { \tl_to_str:N \l__unravel_head_tl }
  }
\__unravel_prefixed_new:nn { letterspace_font }                    % 101
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_scan_r_token:
    \__unravel_print_action:e
      { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
    \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
    \__unravel_just_print_assigned_token:
    \__unravel_scan_optional_equals:
    \__unravel_scan_font_ident:
    \__unravel_scan_int:
    \__unravel_assign_token:n { }
  }
\__unravel_prefixed_new:nn { pdf_copy_font }                       % 102
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
    \__unravel_scan_r_token:
    \__unravel_print_action:e
      { \g__unravel_action_text_str \tl_to_str:N \l__unravel_defined_tl }
    \exp_after:wN \cs_set_eq:NN \l__unravel_defined_tl \__unravel_nullfont:
    \__unravel_just_print_assigned_token:
    \__unravel_scan_optional_equals:
    \__unravel_scan_font_ident:
    \__unravel_assign_token:n { }
  }
```

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

```
\__unravel_prefixed_new:nn { register }                            % 89
  { \__unravel_do_register:N 0 }
\__unravel_prefixed_new:nn { advance }                             % 90
  { \__unravel_do_operation:N 1 }
\__unravel_prefixed_new:nn { multiply }                            % 91
  { \__unravel_do_operation:N 2 }
\__unravel_prefixed_new:nn { divide }                              % 92
  { \__unravel_do_operation:N 3 }
```

`\__unravel_do_operation:N`
`\__unravel_do_operation_fail:w`

```
\cs_new_protected:Npn \__unravel_do_operation:N #1
  {
    \__unravel_prev_input_silent:V \l__unravel_head_tl
    \__unravel_print_action:
    \__unravel_get_x_next:
    \__unravel_set_cmd:
    \int_compare:nNnTF
      \l__unravel_head_cmd_int > { \__unravel_tex_use:n { assign_mu_glue } }
      {
        \int_compare:nNnTF
          \l__unravel_head_cmd_int = { \__unravel_tex_use:n { register } }
          { \__unravel_do_register:N #1 }
          { \__unravel_do_operation_fail:w }
      }
      {
        \int_compare:nNnTF
          \l__unravel_head_cmd_int < { \__unravel_tex_use:n { assign_int } }
          { \__unravel_do_operation_fail:w }
          {
            \__unravel_prev_input:V \l__unravel_head_tl
            \exp_args:NNf \__unravel_do_register_set:Nn #1
              {
                \int_eval:n
                  {
                    \l__unravel_head_cmd_int
                    - \__unravel_tex_use:n { assign_toks }
                  }
              }
          }
      }
  }
\cs_new_protected:Npn \__unravel_do_operation_fail:w
  {
    \__unravel_error:nnnnn { after-advance } { } { } { } { }
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_omit_after_assignment:w
  }
```

(*End of definition for* `\__unravel_do_operation:N` *and* `\__unravel_do_operation_fail:w`.)

`\__unravel_do_register:N`
`\__unravel_do_register_aux:Nn`

```
\cs_new_protected:Npn \__unravel_do_register:N #1
  {
    \exp_args:NNV \__unravel_do_register_aux:Nn #1
      \l__unravel_head_char_int
  }
\cs_new_protected:Npn \__unravel_do_register_aux:Nn #1#2
  {
    \int_compare:nNnTF { \tl_tail:n {#2} } = 0
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_assignment:
        \__unravel_scan_int:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_prev_input_silent:V \l__unravel_head_tl
      }
      {
        \__unravel_prev_input_silent:V \l__unravel_head_tl
        \__unravel_print_assignment:
      }
    \tl_set_eq:NN \l__unravel_defined_tl \l__unravel_head_tl
    \exp_args:NNf \__unravel_do_register_set:Nn #1
      { \int_eval:n { #2 / 1 000 000 } }
  }
```

(*End of definition for* `\__unravel_do_register:N` *and* `\__unravel_do_register_aux:Nn`.)

__unravel_do_register_set:Nn

```
\cs_new_protected:Npn \__unravel_do_register_set:Nn #1#2
  {
    \int_compare:nNnTF {#1} = 0
      { % truly register command
        \__unravel_scan_optional_equals:
      }
      { % \advance, \multiply, \divide
        \__unravel_scan_keyword:nF { bByY }
          { \__unravel_prev_input_silent:n { by } }
      }
    \int_compare:nNnTF {#1} < 2
      {
        \int_case:nnF {#2}
          {
            { 1 } { \__unravel_scan_int:          } % count
            { 2 } { \__unravel_scan_normal_dimen: } % dim
            { 3 } { \__unravel_scan_normal_glue:  } % glue
            { 4 } { \__unravel_scan_mu_glue:      } % muglue
          }
          { \__unravel_error:neeee { internal } { do-reg=#2 } { } { } { } }
      }
      { \__unravel_scan_int: }
    \__unravel_assign_register:
  }
```

(*End of definition for* `\__unravel_do_register_set:Nn`*.*)

The following is used for instance when making accents.

```
\cs_new_protected:Npn \__unravel_do_assignments:
  {
    \__unravel_get_x_non_relax:
    \__unravel_set_cmd:
    \int_compare:nNnT
      \l__unravel_head_cmd_int
      > { \__unravel_tex_use:n { max_non_prefixed_command } }
      {
        \bool_gset_false:N \g__unravel_set_box_allowed_bool
        \__unravel_prev_input_gpush:
        \__unravel_prefixed_command:
        \bool_gset_true:N \g__unravel_set_box_allowed_bool
        \__unravel_do_assignments:
      }
  }
```

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 LuaTeX \directlua, \luaescapestring, and in XeTeX/(u)pTeX \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.

```
\__unravel_new_tex_expandable:nn { undefined_cs }                  % 103
  { \tl_use:N \l__unravel_head_tl \__unravel_print_expansion: }
```

__unravel_expandafter:
__unravel_unless:
__unravel_unless_bad:

```
\__unravel_new_tex_expandable:nn { expand_after }                  % 104
  {
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_expandafter:D
      { \__unravel_expandafter: } { \__unravel_unless: }
  }
\cs_new_protected:Npn \__unravel_expandafter:
  {
    \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
    \__unravel_get_next:
    \gtl_concat:NNN \l__unravel_head_gtl
      \l__unravel_tmpb_gtl \l__unravel_head_gtl
    \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
    \__unravel_print_expansion:e { \gtl_to_str:N \l__unravel_head_gtl }
    \__unravel_get_next:
    \__unravel_token_if_expandable:NTF \l__unravel_head_token
      { \__unravel_expand_do:N \prg_do_nothing: }
      { \__unravel_back_input: }
    \__unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
    \__unravel_set_action_text:e
      { back_input: ~ \gtl_to_str:N \l__unravel_head_gtl }
    \gtl_pop_left:N \l__unravel_head_gtl
    \__unravel_back_input:
```

```
    \__unravel_print_expansion:
  }
\cs_new_protected:Npn \__unravel_unless:
  {
    \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
    \__unravel_get_token:
    \int_compare:nNnTF
      \l__unravel_head_cmd_int = { \__unravel_tex_use:n { if_test } }
      {
        \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_ifcase:D
          { \__unravel_unless_bad: }
          {
            \tl_put_left:Ne \l__unravel_head_tl
              { \gtl_head_do:NN \l__unravel_tmpb_gtl \exp_not:N }
            \__unravel_expand_nonmacro:
          }
      }
      { \__unravel_unless_bad: }
  }
\cs_new_protected:Npn \__unravel_unless_bad:
  {
    \__unravel_error:nnnnn { bad-unless } { } { } { } { }
    \__unravel_back_input:
  }
```

(*End of definition for* `\__unravel_expandafter:`*,* `\__unravel_unless:`*, and* `\__unravel_unless_bad:`*.*)

`\__unravel_noexpand:N`
`\__unravel_noexpand_after:`
`\__unravel_pdfprimitive:`

Currently not fully implemented.

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.

```
\__unravel_new_tex_expandable:nn { no_expand }                    % 105
  {
    \token_if_eq_meaning:NNTF \l__unravel_head_token \tex_noexpand:D
      { \__unravel_noexpand:N }
      { \__unravel_pdfprimitive: }
  }
\cs_new_protected:Npn \__unravel_noexpand:N #1
  {
    \__unravel_get_token:
    \cs_if_eq:NNTF #1 \prg_do_nothing:
      {
        \tl_if_empty:NTF \l__unravel_head_tl
          { \__unravel_back_input: }
          {
            \exp_after:wN \__unravel_token_if_definable:NTF \l__unravel_head_tl
              { \__unravel_noexpand_after: }
              { \__unravel_back_input: }
          }
      }
      {
        \__unravel_back_input:
        \__unravel_get_next:
        \__unravel_token_if_expandable:NT \l__unravel_head_token
          { \cs_set_eq:NN \l__unravel_head_token \__unravel_special_relax: }
      }
  }
\cs_new_protected:Npn \__unravel_noexpand_after:
  {
    \group_begin:
      \__unravel_set_escapechar:n { 92 }
      \exp_args:NNc
    \group_end:
    \__unravel_noexpand_after:N
      { notexpanded: \exp_after:wN \token_to_str:N \l__unravel_head_tl }
  }
\cs_new_protected:Npn \__unravel_noexpand_after:N #1
  {
    \cs_gset_eq:NN #1 \__unravel_special_relax:
    \__unravel_back_input:n {#1}
  }
\cs_new_protected:Npn \__unravel_pdfprimitive:
  { \__unravel_not_implemented:n { pdfprimitive } }
```

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

`\__unravel_endinput:`
`\__unravel_scantokens:`
`\__unravel_input:`

```
\__unravel_new_tex_expandable:nn { input }                        % 106
  {
    \int_case:nnF \l__unravel_head_char_int
      {
        { 1 } { \__unravel_endinput: } % \endinput
        { 2 } { \__unravel_scantokens: } % \scantokens
      }
      { % 0=\input
        \bool_if:NTF \g__unravel_name_in_progress_bool
          { \__unravel_insert_relax: } { \__unravel_input: }
      }
  }
\cs_new_protected:Npn \__unravel_endinput:
  {
    \group_begin:
      \msg_warning:nn { unravel } { endinput-ignored }
    \group_end:
    \__unravel_print_expansion:
  }
\cs_new_protected:Npn \__unravel_scantokens:
  {
    \__unravel_prev_input_gpush:
    \__unravel_scan_toks:NN \c_false_bool \c_false_bool
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \exp_last_unbraced:NNNo \tl_set_rescan:Nnn
      \l__unravel_head_tl \prg_do_nothing: \l__unravel_tmpa_tl
    \__unravel_back_input:V \__unravel_everyeof:w
    \__unravel_back_input:V \l__unravel_head_tl
    \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_tmpa_tl }
  }
\cs_new_protected:Npn \__unravel_input:
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_scan_file_name:
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \tl_set:Ne \l__unravel_tmpa_tl { \tl_tail:N \l__unravel_head_tl }
    \__unravel_file_get:nN \l__unravel_tmpa_tl \l__unravel_tmpa_tl
    \__unravel_back_input:V \l__unravel_tmpa_tl
    \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_head_tl }
  }
```

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

`\__unravel_csname_loop:`

```
\__unravel_new_tex_expandable:nn { cs_name }                         % 109
  {
    \int_compare:nNnTF \l__unravel_head_char_int = 0
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_expansion:
        \__unravel_csname_loop:
        \__unravel_prev_input_silent:V \l__unravel_head_tl
        \__unravel_get_lastnamedcs:
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_back_input_tl_o:
      }
      {
        \__unravel_back_input:V \g__unravel_lastnamedcs_tl
        \__unravel_print_expansion:e
          { \tl_to_str:N \l__unravel_head_tl = \tl_to_str:N \g__unravel_lastnamedcs_tl }
      }
  }
\cs_new_protected:Npn \__unravel_csname_loop:
  {
    \__unravel_get_x_next:
    \__unravel_gtl_if_head_is_definable:NTF \l__unravel_head_gtl
      {
        \cs_if_eq:NNF \l__unravel_head_token \tex_endcsname:D
          {
            \__unravel_back_input:
            \__unravel_tex_error:nV { missing-endcsname } \l__unravel_head_tl
            \tl_set:Nn \l__unravel_head_tl { \tex_endcsname:D }
          }
      }
      {
        \__unravel_prev_input_silent:e
          { \__unravel_token_to_char:N \l__unravel_head_token }
        \__unravel_csname_loop:
      }
  }
\cs_new_protected:Npn \__unravel_get_lastnamedcs:
  {
    \group_begin:
    \__unravel_prev_input_get:N \l__unravel_head_tl
    \tl_gset:No \g__unravel_lastnamedcs_tl
      { \cs:w \exp_after:wN \__unravel_get_lastnamedcs_check:N \l__unravel_head_tl }
    \group_end:
  }
\cs_new:Npn \__unravel_get_lastnamedcs_check:N #1
  { \if_meaning:w \reverse_if:N #1 \use_i:nn \fi: }
```

(*End of definition for* `\__unravel_csname_loop:`*.*)

```
\__unravel_new_tex_expandable:nn { convert }                          % 110
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_expansion:
    \int_case:nn \l__unravel_head_char_int
      {
        0      \__unravel_scan_int:
        1      \__unravel_scan_int:
        2      \__unravel_convert_string:
        3      \__unravel_convert_meaning:w
        4      \__unravel_scan_font_ident:
        8      \__unravel_scan_font_ident:
        9      \__unravel_scan_font_ident:
        { 10 } \__unravel_scan_font_ident:
        { 11 } \__unravel_scan_int:
        { 12 } \__unravel_scan_int:
        { 13 } \__unravel_scan_pdf_ext_toks:
        { 14 } \__unravel_scan_pdf_ext_toks:
        { 15 } \__unravel_scan_int:
        { 16 } \__unravel_scan_int:
        { 17 } \__unravel_scan_pdfstrcmp:
        { 18 } \__unravel_scan_pdfcolorstackinit:
        { 19 } \__unravel_scan_pdf_ext_toks:
        { 20 } \__unravel_scan_pdf_ext_toks:
        { 22 } \__unravel_scan_pdf_ext_toks:
        { 23 } \__unravel_scan_pdf_ext_toks:
        { 24 }
          {
            \__unravel_scan_keyword:n { fFiIlLeE }
            \__unravel_scan_pdf_ext_toks:
          }
        { 25 } \__unravel_scan_pdffiledump:
        { 26 } \__unravel_scan_pdfmatch:
        { 27 } \__unravel_scan_int:
```

```
        { 28 } \__unravel_scan_int:
        { 30 } \__unravel_scan_int:
        { 31 } \__unravel_scan_pdfximagebbox:
        { 33 } \__unravel_scan_directlua:
        { 34 } \__unravel_scan_pdf_ext_toks:
        { 35 } \__unravel_scan_pdf_ext_toks:
        { 40 }
          {
            \__unravel_scan_int:
            \__unravel_prev_input_silent:n { ~ }
            \__unravel_scan_int:
          }
      }
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_back_input_tl_o:
  }
\cs_new_protected:Npn \__unravel_convert_string:
  {
    \__unravel_get_next:
    \tl_if_empty:NTF \l__unravel_head_tl
      { \__unravel_prev_input:e { \gtl_to_str:N \l__unravel_head_gtl } }
      { \__unravel_prev_input:V \l__unravel_head_tl }
  }
\cs_new_protected:Npn \__unravel_convert_meaning:w
    \__unravel_prev_input_gpop:N \l__unravel_head_tl \__unravel_back_input_tl_o:
  {
    \__unravel_get_next:
    \tl_if_empty:NTF \l__unravel_head_tl
      {
        \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
        \__unravel_prev_input_gpop:N \l__unravel_prev_input_tl
        \exp_args:NNV \gtl_put_left:Nn \l__unravel_tmpb_gtl \l__unravel_prev_input_tl
        \__unravel_prev_input_gpush_gtl:N \l__unravel_tmpb_gtl
        \__unravel_print_action:e { \gtl_to_str:N \l__unravel_tmpb_gtl }
        \__unravel_prev_input_gpop_gtl:N \l__unravel_tmpb_gtl
        \tl_set:Ne \l__unravel_tmpa_tl { \gtl_head_do:NN \l__unravel_head_gtl \tex_meaning:D
        \__unravel_back_input:V \l__unravel_tmpa_tl
        \__unravel_print_expansion:e
          { \gtl_to_str:N \l__unravel_tmpb_gtl = \tl_to_str:N \l__unravel_tmpa_tl }
      }
      {
        \__unravel_prev_input:V \l__unravel_head_tl
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_back_input_tl_o:
      }
  }
\cs_new_protected:Npn \__unravel_scan_pdfstrcmp:
  {
    \__unravel_scan_toks_to_str:
    \__unravel_scan_toks_to_str:
  }
\cs_new_protected:Npn \__unravel_scan_pdfximagebbox:
  { \__unravel_scan_int: \__unravel_scan_int: }
\cs_new_protected:Npn \__unravel_scan_pdfcolorstackinit:
  {
    \__unravel_scan_keyword:nTF { pPaAgGeE }
      { \bool_set_true:N \l__unravel_tmpa_bool }
      { \bool_set_false:N \l__unravel_tmpb_bool }
    \__unravel_scan_keyword:nF { dDiIrReEcCtT }
      { \__unravel_scan_keyword:n { pPaAgGeE } }
    \__unravel_scan_toks_to_str:
  }
\cs_new_protected:Npn \__unravel_scan_pdffiledump:
  {
    \__unravel_scan_keyword:nT { oOfFfFsSeEtT } \__unravel_scan_int:
    \__unravel_scan_keyword:nT { lLeEnNgGtThH } \__unravel_scan_int:
    \__unravel_scan_pdf_ext_toks:
  }
\cs_new_protected:Npn \__unravel_scan_pdfmatch:
  {
    \__unravel_scan_keyword:n { iIcCaAsSeE }
    \__unravel_scan_keyword:nT { sSuUbBcCoOuUnNtT }
      { \__unravel_scan_int: }
    \__unravel_scan_pdf_ext_toks:
    \__unravel_scan_pdf_ext_toks:
  }
\sys_if_engine_luatex:T
  {
    \cs_new_protected:Npn \__unravel_scan_directlua:
      {
        \__unravel_get_x_non_relax:
        \token_if_eq_catcode:NNTF \l__unravel_head_token \c_group_begin_token
          { \__unravel_back_input: }
          {
            \__unravel_scan_int:
            \__unravel_get_x_non_relax:
          }
        \__unravel_scan_pdf_ext_toks:
     }
  }
```

`\__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.

```
\__unravel_new_tex_expandable:nn { the }                          % 111
  { \__unravel_get_the:N }
\cs_new_protected:Npn \__unravel_get_the:N #1
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_expansion:
    \int_if_odd:nTF \l__unravel_head_char_int
      { % \unexpanded, \detokenize
        \__unravel_scan_toks:NN \c_false_bool \c_false_bool
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
      }
      { % \the
        \__unravel_get_x_next:
        \__unravel_rescan_something_internal:n { 5 }
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_set_action_text:e
          {
            \tl_head:N \l__unravel_head_tl
            => \tl_tail:N \l__unravel_head_tl
          }
        \tl_set:Ne \l__unravel_head_tl
          { \exp_not:N \exp_not:n { \tl_tail:N \l__unravel_head_tl } }
      }
    \cs_if_eq:NNTF #1 \__unravel_get_token_xdef:
      {
        \tl_put_right:NV \l__unravel_defining_tl \l__unravel_head_tl
        \__unravel_prev_input_silent:e { \l__unravel_head_tl }
        \__unravel_print_action:
      }
      {
        \cs_if_eq:NNTF #1 \__unravel_get_token_x:
          {
            \exp_args:NNe \gtl_set:Nn \l__unravel_tmpb_gtl { \l__unravel_head_tl }
            \__unravel_prev_input_gtl:N \l__unravel_tmpb_gtl
          }
          {
            \tl_set:Ne \l__unravel_tmpa_tl { \exp_args:NV \exp_not:o \l__unravel_head_tl }
            \__unravel_back_input:V \l__unravel_tmpa_tl
          }
        \__unravel_print_expansion:
      }
    #1
  }
```

(*End of definition for* `\__unravel_get_the:N`.)

```
\__unravel_new_tex_expandable:nn { top_bot_mark }                  % 112
  { \__unravel_back_input_tl_o: }
```

```
\__unravel_new_tex_expandable:nn { end_template }                  % 117
  { \__unravel_back_input_tl_o: }
```

2.13.1 Conditionals

`\__unravel_pass_text:`
`\__unravel_pass_text_done:w`

```
\cs_new_protected:Npn \__unravel_pass_text:
  {
    \__unravel_input_if_empty:TF
      { \__unravel_pass_text_empty: }
      {
        \__unravel_input_get:N \l__unravel_tmpb_gtl
        \if_true:
          \if_case:w \gtl_head_do:NN \l__unravel_tmpb_gtl \c_one_int
            \exp_after:wN \__unravel_pass_text_done:w
          \fi:
          \__unravel_input_gpop:N \l__unravel_tmpb_gtl
          \exp_after:wN \__unravel_pass_text:
        \else:
          \use:c { fi: }
          \int_set:Nn \l__unravel_if_nesting_int { 1 }
          \__unravel_input_gpop:N \l__unravel_tmpb_gtl
          \exp_after:wN \__unravel_pass_text_nested:
        \fi:
      }
  }
\cs_new_protected:Npn \__unravel_pass_text_done:w
  {
    \__unravel_get_next:
    \token_if_eq_meaning:NNT \l__unravel_head_token \fi: { \if_true: }
    \else:
  }
```

(*End of 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:`.

```
\cs_new_protected:Npn \__unravel_pass_text_nested:
  {
    \__unravel_input_if_empty:TF
      { \__unravel_pass_text_empty: }
      {
        \__unravel_input_get:N \l__unravel_tmpb_gtl
        \if_true:
          \if_true:
            \gtl_head_do:NN \l__unravel_tmpb_gtl \else:
            \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:
        \__unravel_input_gpop:N \l__unravel_unused_gtl
        \int_compare:nNnTF \l__unravel_if_nesting_int = 0
          { \__unravel_pass_text: }
          { \__unravel_pass_text_nested: }
      }
  }
```

(*End of definition for* `\__unravel_pass_text_nested:`*.*)

__unravel_pass_text_empty:

```
\cs_new_protected:Npn \__unravel_pass_text_empty:
  {
    \__unravel_error:nnnnn { runaway-if } { } { } { } { }
    \__unravel_exit_hard:w
  }
```

(*End of definition for* `\__unravel_pass_text_empty:`.)

__unravel_cond_push:
__unravel_cond_pop:

```
\cs_new_protected:Npn \__unravel_cond_push:
  {
    \tl_gput_left:Ne \g__unravel_if_limit_tl
      { { \int_use:N \g__unravel_if_limit_int } }
    \int_gincr:N \g__unravel_if_depth_int
    \int_gzero:N \g__unravel_if_limit_int
  }
\cs_new_protected:Npn \__unravel_cond_pop:
  {
    \fi:
    \int_gset:Nn \g__unravel_if_limit_int
      { \tl_head:N \g__unravel_if_limit_tl }
    \tl_gset:Ne \g__unravel_if_limit_tl
      { \tl_tail:N \g__unravel_if_limit_tl }
    \int_gdecr:N \g__unravel_if_depth_int
  }
```

(*End of definition for* `\__unravel_cond_push:` *and* `\__unravel_cond_pop:`.)

__unravel_change_if_limit:nn

```
\cs_new_protected:Npn \__unravel_change_if_limit:nn #1#2
  {
    \int_compare:nNnTF {#2} = \g__unravel_if_depth_int
      { \int_gset:Nn \g__unravel_if_limit_int {#1} }
      {
        \tl_clear:N \l__unravel_tmpa_tl
        \prg_replicate:nn { \g__unravel_if_depth_int - #2 - 1 }
          {
            \tl_put_right:Ne \l__unravel_tmpa_tl
              { { \tl_head:N \g__unravel_if_limit_tl } }
            \tl_gset:Ne \g__unravel_if_limit_tl
              { \tl_tail:N \g__unravel_if_limit_tl }
          }
        \tl_gset:Ne \g__unravel_if_limit_tl
          { \l__unravel_tmpa_tl {#1} \tl_tail:N \g__unravel_if_limit_tl }
      }
  }
```

(*End of definition for* `\__unravel_change_if_limit:nn`.)

```
\__unravel_new_tex_expandable:nn { if_test }                        % 107
  {
    \__unravel_cond_push:
    \exp_args:NV \__unravel_cond_aux:n \g__unravel_if_depth_int
  }
```

__unravel_cond_aux:nn

```
\cs_new_protected:Npn \__unravel_cond_aux:n #1
  {
    \int_case:nnF \l__unravel_head_char_int
      {
        {  0 } { \__unravel_test_two_chars:nn { 0 } {#1} } % if
        {  1 } { \__unravel_test_two_chars:nn { 1 } {#1} } % ifcat
        { 12 } { \__unravel_test_ifx:n {#1} }
        { 16 } { \__unravel_test_case:n {#1} }
        { 20 } { \if_true: \__unravel_test_incsname:n {#1} }
        { 21 } { \if_true: \__unravel_test_pdfprimitive:n {#1} }
      }
      {
        \__unravel_prev_input_gpush:N \l__unravel_head_tl
        \__unravel_print_expansion:
        \int_case:nn \l__unravel_head_char_int
          {
            {  2 } % ifnum
              { \__unravel_test_two_vals:N \__unravel_scan_int: }
            {  3 } % ifdim
              { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
            {  4 } { \__unravel_scan_int: } % ifodd
            % {  5 } { } % if[hvm]mode, ifinner, if[tydm]dir
            {  9 } { \__unravel_scan_int: } % ifvoid, ifhbox, ifvbox etc
            { 13 } { \__unravel_scan_int: } % ifeof
            % { 14 } { } % iftrue
            % { 15 } { } % iffalse
            { 17 } { \__unravel_test_ifdefined: } % ifdefined
            { 18 } { \__unravel_test_ifcsname: } % ifcsname
            { 19 } % iffontchar
              { \__unravel_scan_font_ident: \__unravel_scan_int: }
            { 22 } % ifabsnum
              { \__unravel_test_two_vals:N \__unravel_scan_int: }
            { 23 } % ifabsdim
              { \__unravel_test_two_vals:N \__unravel_scan_normal_dimen: }
            { 24 } { \__unravel_scan_font_ident: } % ifjfont, iftfont
          }
        \__unravel_prev_input_gpop:N \l__unravel_head_tl
        \__unravel_set_action_text:e { \tl_to_str:N \l__unravel_head_tl }
        \l__unravel_head_tl \scan_stop:
          \__unravel_cond_true:NNNn
        \else:
          \__unravel_cond_false:Nn
        \fi:
        {#1}
      }
  }
```

(*End of definition for* `\__unravel_cond_aux:nn`.)

__unravel_cond_true:NNNn

```
\cs_new_protected:Npn \__unravel_cond_true:NNNn #1#2#3#4
  {
    \__unravel_change_if_limit:nn { 3 } {#4} % wait for else/fi
    \__unravel_print_expansion:e { \g__unravel_action_text_str = true }
  }
```

(*End of definition for* `\__unravel_cond_true:NNNn`.)

`\__unravel_cond_false:Nn`
`\__unravel_cond_false_loop:n`
`\__unravel_cond_false_common:`

```
\cs_new_protected:Npn \__unravel_cond_false:Nn #1#2
  {
    \__unravel_cond_false_loop:n {#2}
    \__unravel_cond_false_common:
    \__unravel_print_expansion:e
      {
        \g__unravel_action_text_str = false ~
        => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
      }
  }
\cs_new_protected:Npn \__unravel_cond_false_loop:n #1
  {
    \__unravel_pass_text:
    \int_compare:nNnTF \g__unravel_if_depth_int = {#1}
      {
        \token_if_eq_meaning:NNT \l__unravel_head_token \or:
          {
            \__unravel_error:nnnnn { extra-or } { } { } { } { }
            \__unravel_cond_false_loop:n {#1}
          }
      }
      {
        \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
          { \__unravel_cond_pop: }
        \__unravel_cond_false_loop:n {#1}
      }
  }
\cs_new_protected:Npn \__unravel_cond_false_common:
  {
    \token_if_eq_meaning:NNTF \l__unravel_head_token \fi:
      { \__unravel_cond_pop: }
      { \int_gset:Nn \g__unravel_if_limit_int { 2 } } % wait for fi
  }
```

(*End of definition for* `\__unravel_cond_false:Nn`*,* `\__unravel_cond_false_loop:n`*, and* `\__unravel_cond_false_common:`.)

`\__unravel_test_two_vals:N`

```
\cs_new_protected:Npn \__unravel_test_two_vals:N #1
  {
    #1
    \__unravel_get_x_non_blank:
    \__unravel_tl_if_in:ooTF { < = > } \l__unravel_head_tl { }
      {
        \__unravel_error:nnnnn { missing-equals } { } { } { } { }
        \__unravel_back_input:
        \tl_set:Nn \l__unravel_head_tl { = }
      }
    \__unravel_prev_input:V \l__unravel_head_tl
    #1
  }
```

(*End of definition for* `\__unravel_test_two_vals:N`.)

__unravel_test_two_chars:nn
__unravel_test_two_chars_get:n
__unravel_test_two_chars_gtl:N

```
\cs_new_protected:Npn \__unravel_test_two_chars:nn #1
  {
    \exp_args:NNo \gtl_set:Nn \l__unravel_head_gtl { \l__unravel_head_tl }
    \__unravel_prev_input_gpush_gtl:N \l__unravel_head_gtl
    \__unravel_print_expansion:
    \__unravel_test_two_chars_get:n {#1}
    \__unravel_test_two_chars_get:n {#1}
    \__unravel_prev_input_gpop_gtl:N \l__unravel_head_gtl
    \__unravel_set_action_text:e { \gtl_to_str:N \l__unravel_head_gtl }
    \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
    \exp_args:No \tl_if_head_eq_meaning:nNT \l__unravel_head_tl \reverse_if:N
      {
        \gtl_pop_left_item:NNTF \l__unravel_head_gtl \l__unravel_head_tl { } { }
        \tl_put_left:Nn \l__unravel_head_tl { \reverse_if:N }
      }
    \gtl_pop_left:NN \l__unravel_head_gtl \l__unravel_tmpb_gtl
    \__unravel_test_two_chars_gtl:N \l__unravel_tmpb_gtl
    \__unravel_test_two_chars_gtl:N \l__unravel_head_gtl
    \l__unravel_head_tl \scan_stop:
      \__unravel_cond_true:NNNn
    \else:
      \__unravel_cond_false:Nn
    \fi:
  }
\cs_new_protected:Npn \__unravel_test_two_chars_get:n #1
  {
    \__unravel_get_x_next:
    \int_compare:nNnT {#1} = 0
      {
        \gtl_if_head_is_N_type:NF \l__unravel_head_gtl
          { \gtl_set:Ne \l__unravel_head_gtl { \gtl_to_str:N \l__unravel_head_gtl } }
      }
    \__unravel_prev_input_gtl:N \l__unravel_head_gtl
    \__unravel_print_action:e { \gtl_to_str:N \l__unravel_head_gtl }
  }
\cs_new_protected:Npn \__unravel_test_two_chars_gtl:N #1
  {
    \tl_put_right:Ne \l__unravel_head_tl
      {
        \gtl_if_head_is_group_begin:NTF #1 { \c_group_begin_token }
          {
            \gtl_if_head_is_group_end:NTF #1 { \c_group_end_token }
              {
                \exp_not:N \exp_not:N
                \exp_not:f { \gtl_head_do:NN #1 \exp_stop_f: }
              }
          }
      }
  }
```

(*End of 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`.

```
\cs_new_protected:Npn \__unravel_test_ifx:n #1
  {
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_expansion:
    \__unravel_get_next:
    \gtl_set_eq:NN \l__unravel_tmpb_gtl \l__unravel_head_gtl
    \cs_set_eq:NN \l__unravel_tmpb_token \l__unravel_head_token
    \__unravel_get_next:
    \__unravel_prev_input_gpop:N \l__unravel_tmpa_tl
    \__unravel_set_action_text:e
      {
        Compare:~ \tl_to_str:N \l__unravel_tmpa_tl
        \__unravel_test_ifx_str:NN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
        \__unravel_test_ifx_str:NN \l__unravel_head_token \l__unravel_head_gtl
      }
    \__unravel_test_ifx_aux:NNN \l__unravel_tmpb_token \l__unravel_tmpb_gtl
      \__unravel_test_ifx_aux:w
      \__unravel_cond_true:NNNn
    \else:
      \__unravel_cond_false:Nn
    \fi:
    {#1}
  }
\cs_new:Npn \__unravel_test_ifx_str:NN #1#2
  {
    \token_if_eq_meaning:NNT #1 \__unravel_special_relax:
      { \iow_char:N \\notexpanded: }
    \gtl_to_str:N #2
  }
\cs_new_protected:Npn \__unravel_test_ifx_aux:NNN #1#2#3
  {
    \token_if_eq_meaning:NNTF #1 \__unravel_special_relax:
      {
        \gtl_head_do:NN #2 \__unravel_token_if_expandable:NTF
          { #3 #1 } { \gtl_head_do:NN #2 #3 }
      }
      { \gtl_head_do:NN #2 #3 }
  }
\cs_new:Npn \__unravel_test_ifx_aux:w
  {
    \__unravel_test_ifx_aux:NNN \l__unravel_head_token \l__unravel_head_gtl
```

```
      \l__unravel_tmpa_tl
  }
```

(*End of definition for* `\__unravel_test_ifx:n` *and others.*)

`\__unravel_test_case:n`
`\__unravel_test_case_aux:nn`

```
\cs_new_protected:Npn \__unravel_test_case:n #1
  {
    \if_case:w 0 ~
    \__unravel_prev_input_gpush:N \l__unravel_head_tl
    \__unravel_print_expansion:
    \bool_if:NT \g__unravel_internal_debug_bool { \iow_term:n { {\ifcase level~#1} } }
    \__unravel_scan_int:
    \__unravel_prev_input_get:N \l__unravel_head_tl
    \tl_set:Ne \l__unravel_head_tl { \tl_tail:N \l__unravel_head_tl }
    % ^^A does text_case_aux use prev_input_seq?
    \int_compare:nNnF { \l__unravel_head_tl } = 0
      {
        \int_compare:nNnTF { \l__unravel_head_tl } > 0
          { \fi: \if_case:w 1 ~ \or: }
          { \fi: \if_case:w -1 ~ \else: }
      }
    \exp_args:No \__unravel_test_case_aux:nn { \l__unravel_head_tl } {#1}
    \__unravel_prev_input_gpop:N \l__unravel_head_tl
    \__unravel_print_expansion:e { \tl_to_str:N \l__unravel_head_tl }
  }
\cs_new_protected:Npn \__unravel_test_case_aux:nn #1#2
  {
    \int_compare:nNnTF {#1} = 0
      { \__unravel_change_if_limit:nn { 4 } {#2} }
      {
        \__unravel_pass_text:
        \int_compare:nNnTF \g__unravel_if_depth_int = {#2}
          {
            \token_if_eq_meaning:NNTF \l__unravel_head_token \or:
              {
                \exp_args:Nf \__unravel_test_case_aux:nn
                  { \int_eval:n { #1 - 1 } } {#2}
              }
              { \__unravel_cond_false_common: }
          }
          {
            \token_if_eq_meaning:NNT \l__unravel_head_token \fi:
              { \__unravel_cond_pop: }
            \__unravel_test_case_aux:nn {#1} {#2}
          }
      }
  }
```

(*End of definition for* `\__unravel_test_case:n` *and* `\__unravel_test_case_aux:nn`*.*)

`\__unravel_test_incsname:n`

```
\cs_new_protected:Npn \__unravel_test_incsname:n #1
  { \__unravel_not_implemented:n { ifincsname } }
```

(*End of definition for* `\__unravel_test_incsname:n`*.*)

`\__unravel_test_pdfprimitive:n`

```
\cs_new_protected:Npn \__unravel_test_pdfprimitive:n #1
  { \__unravel_not_implemented:n { ifpdfprimitive } }
```

(*End of definition for* `\__unravel_test_pdfprimitive:n`*.*)

`\__unravel_test_ifdefined:`

```
\cs_new_protected:Npn \__unravel_test_ifdefined:
  {
    \__unravel_input_if_empty:TF
      { \__unravel_pass_text_empty: }
      {
        \__unravel_input_gpop:N \l__unravel_tmpb_gtl
        \__unravel_set_action_text:e
          {
            Conditional:~ \tl_to_str:N \l__unravel_head_tl
            \gtl_to_str:N \l__unravel_tmpb_gtl
          }
        \__unravel_prev_input:e
          {
            \gtl_if_tl:NTF \l__unravel_tmpb_gtl
              { \gtl_head:N \l__unravel_tmpb_gtl }
              { \gtl_to_str:N \l__unravel_tmpb_gtl }
          }
      }
  }
```

(*End of definition for* `\__unravel_test_ifdefined:`*.*)

`\__unravel_test_ifcsname:`

```
\cs_new_protected:Npn \__unravel_test_ifcsname:
  {
    \__unravel_csname_loop:
    \__unravel_prev_input:V \l__unravel_head_tl
    \__unravel_get_lastnamedcs:
  }
```

(*End of definition for* `\__unravel_test_ifcsname:`*.*)

```
\__unravel_new_tex_expandable:nn { fi_or_else }                    % 108
  {
    \int_compare:nNnTF \l__unravel_head_char_int > \g__unravel_if_limit_int
      {
        \int_compare:nNnTF \g__unravel_if_limit_int = 0
          {
            \int_compare:nNnTF \g__unravel_if_depth_int = 0
              { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
              { \__unravel_insert_relax: }
          }
          { \__unravel_error:nnnnn { extra-fi-or-else } { } { } { } { } }
      }
      {
        \__unravel_set_action_text:
```

```
        \int_compare:nNnF \l__unravel_head_char_int = 2
          {
            \__unravel_fi_or_else_loop:
            \__unravel_set_action_text:e
              {
                \g__unravel_action_text_str \c_space_tl
                => ~ skip ~ to ~ \tl_to_str:N \l__unravel_head_tl
              }
          }
        \__unravel_print_expansion:
        \__unravel_cond_pop:
      }
  }
\cs_new_protected:Npn \__unravel_fi_or_else_loop:
  {
    \int_compare:nNnF \l__unravel_head_char_int = 2
      {
        \__unravel_pass_text:
        \__unravel_set_cmd:
        \__unravel_fi_or_else_loop:
      }
  }
```

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:`
`\l__unravel_print_tl`

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

```
\tl_new:N \l__unravel_print_tl
\sys_if_engine_luatex:TF
  {
    \cs_new_protected:Npe \__unravel_print_normalize_null:
      {
        \tl_replace_all:Nnn \exp_not:N \l__unravel_print_tl
          { \char_generate:nn { 0 } { 12 } }
          { \tl_to_str:n { ^ ^ @ } }
      }
  }
  { \cs_new_protected:Npn \__unravel_print_normalize_null: { } }
```

(*End of definition for* `\__unravel_print_normalize_null:` *and* `\l__unravel_print_tl`*.*)

`\__unravel_print:n`
`\__unravel_print:e`
`\__unravel_log:n`

```
\cs_new_protected:Npn \__unravel_print:n #1
  {
    \tl_set:Nn \l__unravel_print_tl {#1}
    \__unravel_print_normalize_null:
    \iow_term:e { \l__unravel_print_tl }
    \tl_if_empty:NF \g__unravel_output_file_tl
      { \iow_now:Ne \g__unravel_iow { \l__unravel_print_tl } }
  }
\cs_generate_variant:Nn \__unravel_print:n { e }
\cs_new_protected:Npn \__unravel_log:n #1
  {
    \tl_set:Nn \l__unravel_print_tl {#1}
    \__unravel_print_normalize_null:
    \tl_if_empty:NTF \g__unravel_output_file_tl
      { \iow_log:e { \l__unravel_print_tl } }
      { \iow_now:Ne \g__unravel_iow { \l__unravel_print_tl } }
  }
```

(*End of definition for* `\__unravel_print:n` *and* `\__unravel_log:n`*.*)

`\__unravel_print_step:n` Steps are printed or not according to the option value, unaffected by a prompt such as `s10`.

```
\cs_new_protected:Npn \__unravel_print_step:n #1
  {
    \int_compare:nNnF \g__unravel_online_int < 0
      {
        \int_compare:nNnTF \g__unravel_online_int = 0
          { \__unravel_log:n {#1} }
          { \__unravel_print:n {#1} }
      }
  }
```

(*End of definition for* `\__unravel_print_step: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_current_online_int` so as to be sensitive to the prompt.

```
\cs_new_protected:Npn \__unravel_print_message:nn #1 #2
  {
    \int_compare:nNnF \g__unravel_current_online_int < 0
      {
        \int_compare:nNnTF \g__unravel_current_online_int = 0
          { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_log:n }
          { \iow_wrap:nnnN { #1 #2 } { #1 } { } \__unravel_print:n }
      }
  }
```

(*End of definition for* `\__unravel_print_message:nn`*.*)

`\__unravel_set_action_text:e`

```
\cs_new_protected:Npn \__unravel_set_action_text:e #1
  {
    \group_begin:
      \__unravel_set_escapechar:n { 92 }
      \str_gset:Ne \g__unravel_action_text_str {#1}
    \group_end:
  }
```

(*End of definition for* `\__unravel_set_action_text:e`*.*)

`\__unravel_set_action_text:`

```
\cs_new_protected:Npn \__unravel_set_action_text:
  {
    \__unravel_set_action_text:e
      {
        \tl_to_str:N \l__unravel_head_tl
        \tl_if_single_token:VT \l__unravel_head_tl
          { = ~ \token_to_meaning:N \l__unravel_head_token }
      }
  }
```

(*End of definition for* `\__unravel_set_action_text:`*.*)

`\__unravel_print_state:`

```
\cs_new_protected:Npn \__unravel_print_state:
  {
    \group_begin:
      \__unravel_set_escapechar:n { 92 }
      \tl_use:N \g__unravel_before_print_state_tl
      \int_compare:nNnF \g__unravel_current_online_int < 0
        {
          \__unravel_print_state_output:
          \__unravel_print_state_prev:
          \__unravel_print_state_input:
        }
    \group_end:
  }
```

(*End of definition for* `\__unravel_print_state:`*.*)

`\__unravel_print_state_output:`
`\__unravel_print_state_output:n`

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

```
\cs_new_protected:Npn \__unravel_print_state_output:
  {
    \exp_args:Ne \__unravel_print_state_output:n
      { \gtl_to_str:N \g__unravel_output_gtl }
  }
\cs_new_protected:Npn \__unravel_print_state_output:n #1
  {
    \tl_if_empty:nF {#1}
      {
        \__unravel_print_message:nn { <| ~ } % |
          { \__unravel_str_truncate_left:nn {#1} { \g__unravel_max_output_int } }
      }
  }
```

(*End of definition for* `\__unravel_print_state_output:` *and* `\__unravel_print_state_output:n`*.*)

`\__unravel_print_state_prev:`

Never trim `##1`.

```
\cs_new_protected:Npn \__unravel_print_state_prev:
  {
    \seq_set_map_e:NNn \l__unravel_tmpa_seq \g__unravel_prev_input_seq
      { \__unravel_to_str:Nn ##1 }
    \seq_remove_all:Nn \l__unravel_tmpa_seq { }
    \seq_if_empty:NTF \l__unravel_tmpa_seq
      { \__unravel_print_message:nn { || ~ } { } }
      {
        \seq_map_inline:Nn \l__unravel_tmpa_seq
          {
            \__unravel_print_message:nn { || ~ } {##1}
          }
      }
  }
```

(*End of definition for* `\__unravel_print_state_prev:`*.*)

`\__unravel_print_state_input:`
`\__unravel_print_state_input:n`

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

```
\cs_new_protected:Npn \__unravel_print_state_input:
  {
    \exp_args:Ne \__unravel_print_state_input:n
      { \__unravel_input_to_str: }
  }
\cs_new_protected:Npn \__unravel_print_state_input:n #1
  {
    \__unravel_print_message:nn { |> ~ } % |
      { \__unravel_str_truncate_right:nn {#1} { \g__unravel_max_input_int } }
  }
```

(*End of definition for* `\__unravel_print_state_input:` *and* `\__unravel_print_state_input:n`*.*)

`\__unravel_print_meaning:`

```
\cs_new_protected:Npn \__unravel_print_meaning:
  {
    \__unravel_input_if_empty:TF
      { \__unravel_print_message:nn { } { Empty~input! } }
      {
        \__unravel_input_get:N \l__unravel_tmpb_gtl
        \__unravel_print_message:nn { }
          {
            \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_str:N
            = \gtl_head_do:NN \l__unravel_tmpb_gtl \token_to_meaning:N
          }
      }
  }
```

(*End of definition for* `\__unravel_print_meaning:`*.*)

`\__unravel_print_action:`
`\__unravel_print_action:e`
`\__unravel_print_assignment:`
`\__unravel_print_assignment:e`
`\__unravel_print_expansion:`
`\__unravel_print_expansion:e`
`\__unravel_print_action_aux:N`

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

```
\cs_new_protected:Npn \__unravel_print_action:
  { \__unravel_print_action_aux:N \g__unravel_trace_other_bool }
\cs_new_protected:Npn \__unravel_print_action:e #1
  {
    \__unravel_set_action_text:e {#1}
    \__unravel_print_action:
  }
\cs_new_protected:Npn \__unravel_print_assignment:
  { \__unravel_print_action_aux:N \g__unravel_trace_assigns_bool }
\cs_new_protected:Npn \__unravel_print_assignment:e #1
  {
    \__unravel_set_action_text:e {#1}
    \__unravel_print_assignment:
  }
\cs_new_protected:Npn \__unravel_print_expansion:
  { \__unravel_print_action_aux:N \g__unravel_trace_expansion_bool }
\cs_new_protected:Npn \__unravel_print_expansion:e #1
  {
    \__unravel_set_action_text:e {#1}
    \__unravel_print_expansion:
  }
\cs_new_protected:Npn \__unravel_print_action_aux:N #1
  {
    \int_gdecr:N \g__unravel_nonstop_int
    \int_gincr:N \g__unravel_step_int
    \bool_if:NT #1
      {
        \exp_args:Ne \__unravel_print_step:n
          {
            [=====
            \bool_if:NT \g__unravel_number_steps_bool
              { ~ Step ~ \int_to_arabic:n { \g__unravel_step_int } ~ }
            =====]~
            \int_compare:nNnTF
              { \str_count:N \g__unravel_action_text_str }
              > { \g__unravel_max_action_int }
              {
                \str_range:Nnn \g__unravel_action_text_str
                  { 1 } { \g__unravel_max_action_int - 3 } ...
              }
              { \g__unravel_action_text_str }
          }
        \__unravel_print_state:
        \__unravel_prompt:
      }
  }
```

(*End of definition for* `\__unravel_print_action:` *and others.*)

`\__unravel_just_print_assigned_token:`
`\__unravel_print_assigned_token:`
`\__unravel_print_assigned_register:`
`\__unravel_print_assigned_parshape:`
`\__unravel_print_assigned_set_shape:`
`\__unravel_print_assigned_set_shape_aux:n`

```
\cs_new_protected:Npn \__unravel_just_print_assigned_token:
  {
    \__unravel_print_assignment:e
      {
        Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
        = \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl
      }
  }
\cs_new_protected:Npn \__unravel_print_assigned_token:
  {
    \__unravel_after_assignment:
    \__unravel_just_print_assigned_token:
    \__unravel_omit_after_assignment:w
  }
\cs_new:Npn \__unravel_print_assigned_aux_name:
  {
    Set~ \exp_after:wN \token_to_str:N \l__unravel_defined_tl
    \tl_if_single:NT \l__unravel_defined_tl
      { ( \exp_after:wN \token_to_meaning:N \l__unravel_defined_tl ) }
  }
\cs_new_protected:Npn \__unravel_print_assigned_register:
  {
    \__unravel_after_assignment:
    \exp_args:Ne \__unravel_print_assignment:e % needed to stringify a \toks
      {
        \exp_not:N \__unravel_print_assigned_aux_name:
        = \exp_not:N \tl_to_str:n { \__unravel_the:w \l__unravel_defined_tl }
      }
    \__unravel_omit_after_assignment:w
  }
\cs_new_protected:Npn \__unravel_print_assigned_parshape:
  {
    \__unravel_after_assignment:
    \tl_set:Nn \l__unravel_tmpa_tl { \tex_parshapedimen:D }
    \__unravel_print_assignment:e
      {
        \__unravel_print_assigned_aux_name: = \__unravel_the:w \l__unravel_defined_tl
        \int_step_function:nN { 2 * \l__unravel_defined_tl }
          \__unravel_print_assigned_set_shape_aux:n
      }
    \__unravel_omit_after_assignment:w
  }
\cs_new_protected:Npn \__unravel_print_assigned_set_shape:
  {
    \__unravel_after_assignment:
    \tl_set_eq:NN \l__unravel_tmpa_tl \l__unravel_defined_tl
    \__unravel_print_assignment:e
      {
        \__unravel_print_assigned_aux_name:
        = \__unravel_the:w \l__unravel_defined_tl 0 \exp_stop_f:
        \int_step_function:nN { \l__unravel_defined_tl 0 }
          \__unravel_print_assigned_set_shape_aux:n
      }
    \__unravel_omit_after_assignment:w
  }
\cs_new:Npn \__unravel_print_assigned_set_shape_aux:n #1
  { ~ \__unravel_the:w \l__unravel_tmpa_tl #1 \exp_stop_f: }
```

(*End of definition for* `\__unravel_just_print_assigned_token:` *and others.*)

`\__unravel_print_welcome:` Welcome message.

```
\cs_new_protected:Npn \__unravel_print_welcome:
  {
    \__unravel_print_message:nn { }
```

```
      {
        \bool_if:NTF \g__unravel_welcome_message_bool
          {
            \\
            ========~ Welcome~ to~ the~ unravel~ package~ ========\\
            \iow_indent:n
              {
                "<|"~ denotes~ the~ output~ to~ TeX's~ stomach. \\
                "||"~ denotes~ tokens~ waiting~ to~ be~ used. \\
                "|>"~ denotes~ tokens~ that~ we~ will~ act~ on. \\
                Press~<enter>~to~continue;~'h'~<enter>~for~help. \\
              }
          }
          { [=====~Start~=====] }
      }
    \__unravel_print_state:
    \__unravel_prompt:
  }
```

(*End of definition for* `\__unravel_print_welcome:`.)

`\__unravel_print_outcome:` Final message.

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

(*End of definition for* `\__unravel_print_outcome:`.)

2.14.2 Prompt

`\__unravel_ior_str_get:NN`
`\__unravel_ior_str_get:Nc`

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

(*End of definition for* `\__unravel_ior_str_get:NN`.)

`\__unravel_prompt:`

```
\cs_new_protected:Npn \__unravel_prompt:
  {
    \int_compare:nNnF \g__unravel_nonstop_int > 0
      {
        \group_begin:
          \__unravel_set_escapechar:n { -1 }
          \int_set:Nn \tex_endlinechar:D { -1 }
          \tl_use:N \g__unravel_before_prompt_tl
          \__unravel_prompt_aux:
        \group_end:
      }
  }
\cs_new_protected:Npn \__unravel_prompt_aux:
  {
    \clist_if_empty:NTF \g__unravel_prompt_input_clist
      {
        \int_compare:nNnT { \tex_interactionmode:D } = { 3 }
          {
```

```
      \bool_if:NTF \g__unravel_explicit_prompt_bool
        { \__unravel_ior_str_get:Nc \c__unravel_prompt_ior }
        { \__unravel_ior_str_get:Nc \c__unravel_noprompt_ior }
          { Your~input }
      \exp_args:Nv \__unravel_prompt_treat:n { Your~input }
     }
   }
   {
    \clist_gpop:NN \g__unravel_prompt_input_clist \l__unravel_tmpa_tl
    \group_begin:
     \__unravel_set_escapechar:n { 92 }
     \__unravel_print:e
      {
       \bool_if:NT \g__unravel_explicit_prompt_bool { Your~input= }
       \tl_to_str:N \l__unravel_tmpa_tl
      }
    \group_end:
    \exp_args:NV \__unravel_prompt_treat:n \l__unravel_tmpa_tl
   }
 }
\cs_new_protected:Npn \__unravel_prompt_treat:n #1
 {
  \tl_if_empty:nF {#1}
   {
    \str_case:enF { \tl_head:n {#1} }
     {
      { m } { \__unravel_print_meaning: \__unravel_prompt_aux: }
      { q }
       {
        \int_gset:Nn \g__unravel_current_online_int { -1 }
        \int_gzero:N \g__unravel_nonstop_int
       }
      { x }
       {
        \group_end:
        \__unravel_exit_hard:w
       }
      { X }
       {
        \tex_batchmode:D
        \tex_read:D -1 to \l__unravel_tmpa_tl
       }
      { s } { \__unravel_prompt_scan_int:nn {#1}
       \__unravel_prompt_silent_steps:n }
      { o } { \__unravel_prompt_scan_int:nn {#1}
       { \int_gset:Nn \g__unravel_current_online_int } }
      { C }
       {
        \use:e
         {
          \tl_gset_rescan:Nnn \exp_not:N \g__unravel_tmpc_tl
           { \exp_not:N \ExplSyntaxOn } { \tl_tail:n {#1} }
         }
        \tl_gput_left:Nn \g__unravel_tmpc_tl
```

```
                  { \tl_gclear:N \g__unravel_tmpc_tl }
                \group_insert_after:N \g__unravel_tmpc_tl
                \group_insert_after:N \__unravel_prompt:
              }
            { | } % |
              { \__unravel_prompt_scan_int:nn {#1}
              \__unravel_prompt_vert:n }
            { u } { \__unravel_prompt_until:n {#1} }
            { a } { \__unravel_prompt_all: }
          }
          { \__unravel_prompt_help: }
      }
  }
\cs_new_protected:Npn \__unravel_prompt_scan_int:nn #1
  {
    \tex_afterassignment:D \__unravel_prompt_scan_int_after:wn
    \l__unravel_prompt_tmpa_int =
      \tl_if_head_eq_charcode:fNF { \use_none:n #1 } - { 0 }
      \use_ii:nn #1 \scan_stop:
  }
\cs_new_protected:Npn \__unravel_prompt_scan_int_after:wn #1 \scan_stop: #2
  {
    #2 \l__unravel_prompt_tmpa_int
    \tl_if_blank:nF {#1} { \__unravel_prompt_treat:n {#1} }
  }
\cs_new_protected:Npn \__unravel_prompt_help:
  {
    \__unravel_print:n { "m":~meaning~of~first~token }
    \__unravel_print:n { "a":~print~state~again,~without~truncating }
    \__unravel_print:n { "s<num>":~do~<num>~steps~silently }
    \__unravel_print:n { "|<num>":~silent~steps~until~<num>~fewer~"||" }
    \__unravel_print:n { "u<text>":~silent~steps~until~the~input~starts~with~<text> }
    \__unravel_print:n
      { "o<num>":~1~=>~log~and~terminal,~0~=>~only~log,~-1~=>~neither.}
    \__unravel_print:n { "q":~semi-quiet~(same~as~"o-1") }
    \__unravel_print:n { "C<code>":~run~some~expl3~code~immediately }
    \__unravel_print:n { "x"/"X":~exit~this~instance~of~unravel/TeX }
    \__unravel_prompt_aux:
  }
\cs_new_protected:Npn \__unravel_prompt_silent_steps:n #1
  {
    \int_compare:nNnF {#1} < 0
      {
        \int_gset:Nn \g__unravel_current_online_int { -1 }
        \tl_gset:Nn \g__unravel_before_prompt_tl
          {
            \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
            \tl_gclear:N \g__unravel_before_prompt_tl
          }
        \int_gset:Nn \g__unravel_nonstop_int {#1}
      }
  }
\cs_new_protected:Npn \__unravel_prompt_vert:n #1
  {
    \int_compare:nNnTF {#1} < { 0 }
      { \__unravel_prompt_vert:Nn > {#1} }
      { \__unravel_prompt_vert:Nn < {#1} }
  }
\cs_new_protected:Npn \__unravel_prompt_vert:Nn #1#2
  {
    \int_gset:Nn \g__unravel_current_online_int { -1 }
    \tl_gset:Nf \g__unravel_before_print_state_tl
      {
        \exp_args:NNf \exp_stop_f: \int_compare:nNnTF
          { \int_eval:n { \__unravel_prev_input_count: - #2 } }
          #1 { \__unravel_prev_input_count: }
          {
            \int_gset:Nn \g__unravel_nonstop_int
              { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
          }
          {
            \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
            \tl_gclear:N \g__unravel_before_print_state_tl
          }
      }
  }
\cs_new_protected:Npn \__unravel_prompt_all:
  {
    \tl_gset:Ne \g__unravel_tmpc_tl
      {
        \exp_not:n
          {
            \tl_gclear:N \g__unravel_tmpc_tl
            \int_gset_eq:NN \g__unravel_max_output_int \c_max_int
            \int_gset_eq:NN \g__unravel_max_input_int \c_max_int
            \__unravel_print_state:
            \int_gdecr:N \g__unravel_nonstop_int
            \__unravel_prompt:
          }
        \__unravel_prompt_all_aux:N \g__unravel_max_output_int
        \__unravel_prompt_all_aux:N \g__unravel_max_input_int
      }
    \group_insert_after:N \g__unravel_tmpc_tl
  }
\cs_new:Npn \__unravel_prompt_all_aux:N #1
  { \exp_not:n { \int_gset:Nn #1 } { \int_use:N #1 } }
```

(*End of definition for* `\__unravel_prompt:`*.*)

`\__unravel_prompt_until:n`
`\g__unravel_until_tl`

```
\tl_new:N \g__unravel_until_tl
\cs_new_protected:Npn \__unravel_prompt_until:n #1
  {
    \tl_gset:Ne \g__unravel_until_tl { \tl_tail:n {#1} }
    \int_gset:Nn \g__unravel_current_online_int { -1 }
    \tl_gset:Nn \g__unravel_before_print_state_tl
      {
        \__unravel_input_get_left:N \l__unravel_tmpa_tl
        \use:e
          {
            \exp_not:N \tl_if_in:nnTF
              { \exp_not:N \__unravel:nn \tl_to_str:N \l__unravel_tmpa_tl }
              { \exp_not:N \__unravel:nn \tl_to_str:N \g__unravel_until_tl }
          }
          {
            \int_gzero:N \g__unravel_nonstop_int
            \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
            \tl_gclear:N \g__unravel_before_print_state_tl
          }
          {
            \int_gset:Nn \g__unravel_nonstop_int
              { \int_max:nn { \g__unravel_nonstop_int } { 2 } }
          }
      }
  }
```

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

2.14.3 Errors

`\__unravel_not_implemented:n`

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

(*End of definition for* `\__unravel_not_implemented:n`*.*)

`\__unravel_error:nnnnn`
`\__unravel_error:neeee`

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.

```
\cs_new_protected:Npn \__unravel_error:nnnnn #1#2#3#4#5
  {
    \group_begin:
    \msg_error:nnnnnn { unravel } {#1} {#2} {#3} {#4} {#5}
    \group_end:
  }
\cs_new_protected:Npn \__unravel_error:neeee #1#2#3#4#5
  {
    \group_begin:
    \msg_error:nneeee { unravel } {#1} {#2} {#3} {#4} {#5}
    \group_end:
  }
```

(*End of definition for* `\__unravel_error:nnnnn`*.*)

`\__unravel_tex_msg_new:nnn`

This stores a TEX error message.

```
\cs_new_protected:Npn \__unravel_tex_msg_new:nnn #1#2#3
  {
    \cs_new:cpn { __unravel_tex_msg_error_#1: } {#2}
    \cs_new:cpn { __unravel_tex_msg_help_#1: } {#3}
  }
```

(*End of definition for* `\__unravel_tex_msg_new:nnn`*.*)

`\__unravel_tex_error:nn`
`\__unravel_tex_error:nV`

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

```
\cs_new_protected:Npn \__unravel_tex_error:nn #1#2
  {
    \__unravel_error:neeee { tex-error }
      { \tl_to_str:n {#2} }
      { \use:c { __unravel_tex_msg_error_#1: } }
      { \use:c { __unravel_tex_msg_help_#1: } }
      { }
  }
\cs_generate_variant:Nn \__unravel_tex_error:nn { nV }
```

(*End of definition for* `\__unravel_tex_error:nn`.)

`\__unravel_tex_fatal_error:nn`
`\__unravel_tex_fatal_error:nV`

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

```
\cs_new_protected:Npn \__unravel_tex_fatal_error:nn #1#2
  {
    \__unravel_error:neeee { tex-fatal }
      { \tl_to_str:n {#2} }
      { \use:c { __unravel_tex_msg_error_#1: } }
      { \use:c { __unravel_tex_msg_help_#1: } }
      { }
  }
\cs_generate_variant:Nn \__unravel_tex_fatal_error:nn { nV }
```

(*End of 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.

```
\keys_define:nn { unravel/defaults }
  {
    explicit-prompt  .bool_gset:N = \g__unravel_default_explicit_prompt_bool ,
    internal-debug   .bool_gset:N = \g__unravel_default_internal_debug_bool ,
    max-action       .int_gset:N  = \g__unravel_default_max_action_int ,
    max-output       .int_gset:N  = \g__unravel_default_max_output_int ,
    max-input        .int_gset:N  = \g__unravel_default_max_input_int ,
    number-steps     .bool_gset:N = \g__unravel_default_number_steps_bool ,
    online           .int_gset:N  = \g__unravel_default_online_int ,
    output-file      .code:n      = {
      \int_gzero:N \g__unravel_default_online_int
      \tl_gset:Nn \g__unravel_default_output_file_tl {#1}
    } ,
    prompt-input     .code:n
      = \__unravel_prompt_input:Nn \g__unravel_default_prompt_input_clist {#1} ,
    trace-assigns    .bool_gset:N = \g__unravel_default_trace_assigns_bool ,
    trace-expansion  .bool_gset:N = \g__unravel_default_trace_expansion_bool ,
    trace-other      .bool_gset:N = \g__unravel_default_trace_other_bool ,
    welcome-message  .bool_gset:N = \g__unravel_default_welcome_message_bool ,
  }
\keys_define:nn { unravel }
  {
    explicit-prompt  .bool_gset:N = \g__unravel_explicit_prompt_bool ,
    internal-debug   .bool_gset:N = \g__unravel_internal_debug_bool ,
    max-action       .int_gset:N  = \g__unravel_max_action_int ,
    max-output       .int_gset:N  = \g__unravel_max_output_int ,
    max-input        .int_gset:N  = \g__unravel_max_input_int ,
    number-steps     .bool_gset:N = \g__unravel_number_steps_bool ,
    online           .int_gset:N  = \g__unravel_online_int ,
    output-file      .code:n      = {
      \int_gzero:N \g__unravel_online_int
      \tl_gset:Nn \g__unravel_output_file_tl {#1}
    } ,
    prompt-input     .code:n
      = \__unravel_prompt_input:Nn \g__unravel_prompt_input_clist {#1} ,
    trace-assigns    .bool_gset:N = \g__unravel_trace_assigns_bool ,
    trace-expansion  .bool_gset:N = \g__unravel_trace_expansion_bool ,
    trace-other      .bool_gset:N = \g__unravel_trace_other_bool ,
    welcome-message  .bool_gset:N = \g__unravel_welcome_message_bool ,
  }
```

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.

```
\tl_map_inline:nn { { /defaults } { } }
  {
    \keys_define:nn { unravel #1 }
      {
        machine        .meta:nn =
          { unravel #1 }
          {
            explicit-prompt = false ,
            internal-debug  = false ,
            max-action      = \c_max_int ,
            max-output      = \c_max_int ,
            max-input       = \c_max_int ,
            number-steps    = false ,
            welcome-message = false ,
          } ,
        mute          .meta:nn =
          { unravel #1 }
          {
            trace-assigns = false ,
            trace-expansion = false ,
            trace-other = false ,
            welcome-message = false ,
            online = -1 ,
          }
      }
  }
```

2.16 Main command

\unravel Simply call an underlying code-level command.

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

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

\unravelsetup Simply call an underlying code-level command.

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

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

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

```
\cs_new_protected:Npn \unravel_setup:n #1
  {
    \keys_set:nn { unravel/defaults } {#1}
    \keys_set:nn { unravel } {#1}
  }
```

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

\unravel:nn
__unravel:nn
__unravel_unravel_marker:

The command starts with `\__unravel_unravel_marker:` to detect nesting of `\unravel` in `\unravel` and avoid re-initializing important variables. Initialize and setup keys. Initialize and setup other variables including the input. Welcome the user. Then comes the main loop: until the input is exhausted, print the current status and do one step. The main loop is exited by skipping to the first `\__unravel_exit_point:`, while some abort procedures jump to the second (and last) one instead. If the main loop finished correctly, print its outcome and finally test that everything is all right.

```
\cs_new_protected:Npn \unravel:nn { \__unravel_unravel_marker: \__unravel:nn }
\cs_new_eq:NN \__unravel_unravel_marker: \__unravel_special_relax:
\cs_new_protected:Npn \__unravel:nn #1#2
  {
    \__unravel_init_key_vars:
    \keys_set:nn { unravel } {#1}
    \__unravel_init_vars:
    \__unravel_input_gset:n {#2}
    \__unravel_print_welcome:
    \__unravel_main_loop:
    \__unravel_exit_point:
    \__unravel_print_outcome:
    \__unravel_final_test:
    \__unravel_exit_point:
  }
\cs_new_protected:Npn \unravel_get:nnN #1#2#3
  {
    \unravel:nn {#1} {#2}
    \tl_set:Ne #3 { \gtl_left_tl:N \g__unravel_output_gtl }
  }
```

(*End of 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).

```
\cs_new_protected:Npn \__unravel_init_key_vars:
  {
    \sys_if_engine_luatex:T { \tl_gset:No \g__unravel_lastnamedcs_tl { \tex_lastnamedcs:D }
    \bool_gset_eq:NN \g__unravel_explicit_prompt_bool \g__unravel_default_explicit_prompt_bo
    \bool_gset_eq:NN \g__unravel_internal_debug_bool \g__unravel_default_internal_debug_bool
    \bool_gset_eq:NN \g__unravel_number_steps_bool \g__unravel_default_number_steps_bool
    \int_gset_eq:NN  \g__unravel_online_int \g__unravel_default_online_int
    \tl_gset_eq:NN \g__unravel_output_file_tl \g__unravel_default_output_file_tl
    \clist_gset_eq:NN \g__unravel_prompt_input_clist \g__unravel_default_prompt_input_clist
    \bool_gset_eq:NN \g__unravel_trace_assigns_bool \g__unravel_default_trace_assigns_bool
    \bool_gset_eq:NN \g__unravel_trace_expansion_bool \g__unravel_default_trace_expansion_bo
    \bool_gset_eq:NN \g__unravel_trace_other_bool \g__unravel_default_trace_other_bool
    \bool_gset_eq:NN \g__unravel_welcome_message_bool \g__unravel_default_welcome_message_bo
    \int_gset_eq:NN \g__unravel_max_action_int \g__unravel_default_max_action_int
    \int_gset_eq:NN \g__unravel_max_output_int \g__unravel_default_max_output_int
    \int_gset_eq:NN \g__unravel_max_input_int  \g__unravel_default_max_input_int
    \int_gzero:N \g__unravel_nonstop_int
    \tl_gclear:N \g__unravel_before_print_state_tl
    \tl_gclear:N \g__unravel_before_prompt_tl
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_init_vars:
  {
    \int_gset_eq:NN \g__unravel_current_online_int \g__unravel_online_int
    \tl_if_eq:NNF \g__unravel_output_file_tl \g__unravel_current_output_file_tl
      {
        \iow_close:N \g__unravel_iow
        \iow_open:Nn \g__unravel_iow \g__unravel_output_file_tl
        \tl_gset_eq:NN \g__unravel_current_output_file_tl \g__unravel_output_file_tl
      }
    \seq_gclear:N \g__unravel_prev_input_seq
    \gtl_gclear:N \g__unravel_output_gtl
    \int_gzero:N  \g__unravel_step_int
    \tl_gclear:N  \g__unravel_if_limit_tl
    \int_gzero:N  \g__unravel_if_limit_int
    \int_gzero:N  \g__unravel_if_depth_int
    \gtl_gclear:N \g__unravel_after_assignment_gtl
    \bool_gset_true:N  \g__unravel_set_box_allowed_bool
    \bool_gset_false:N \g__unravel_name_in_progress_bool
    \gtl_clear:N \l__unravel_after_group_gtl
  }
```

(*End of definition for* `\__unravel_init_vars:`*.*)

`\__unravel_main_loop:`
`\__unravel_get_x_next_or_done:`

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.

```
\cs_new_protected:Npn \__unravel_get_x_next_or_done:
  {
    \__unravel_input_if_empty:TF { \__unravel_exit:w } { }
    \__unravel_get_next:
    \__unravel_token_if_expandable:NT \l__unravel_head_token
      { \__unravel_expand_do:N \__unravel_get_x_next_or_done: }
  }
\cs_new_protected:Npn \__unravel_main_loop:
  {
    \__unravel_get_x_next_or_done:
    \__unravel_set_cmd:
    \__unravel_do_step:
    \__unravel_main_loop:
  }
```

(*End of 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.

```
\cs_new_protected:Npn \__unravel_do_step:
  {
    \__unravel_set_action_text:
    \bool_if:NT \g__unravel_internal_debug_bool
      { \iow_term:e { Cmd:~\int_to_arabic:n { \l__unravel_head_cmd_int } } }
    \cs_if_exist_use:cF
      { __unravel_cmd_ \int_use:N \l__unravel_head_cmd_int : }
      { \__unravel_error:neeee { internal } { unknown-command } { } { } { } }
  }
```

(*End of definition for* `\__unravel_do_step:`*.*)

`\__unravel_final_test:`
`\__unravel_final_bad:`

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

```
\cs_new_protected:Npn \__unravel_final_test:
  {
    \__unravel_input_if_empty:TF
      {
        \seq_if_empty:NTF \g__unravel_prev_input_seq
          {
            \tl_if_empty:NTF \g__unravel_if_limit_tl
              { \int_compare:nNnF \g__unravel_if_limit_int = 0 { \__unravel_final_bad: } }
              { \__unravel_final_conditionals: }
          }
          { \__unravel_final_bad: }
      }
      { \__unravel_final_bad: }
    \__unravel_final_after_assignment:
  }
\cs_new_protected:Npn \__unravel_final_bad:
  {
    \__unravel_error:nnnnn { internal }
      { the-last-unravel-finished-badly } { } { } { }
  }
```

```
\cs_new_protected:Npn \__unravel_final_conditionals:
  {
    \group_begin:
    \msg_warning:nne { unravel } { dangling-conditionals }
      { \tl_count:N \g__unravel_if_limit_tl }
    \group_end:
    \tl_greverse:N \g__unravel_if_limit_tl
    \tl_gput_right:NV \g__unravel_if_limit_tl \g__unravel_if_limit_int
    \tl_gset:Ne \g__unravel_if_limit_tl { \tl_tail:N \g__unravel_if_limit_tl } % remove the
    \prg_replicate:nn { \tl_count:N \g__unravel_if_limit_tl } { \fi: }
    \tl_map_function:NN \g__unravel_if_limit_tl \__unravel_final_cond_aux:n
  }
\cs_new:Npn \__unravel_final_cond_aux:n #1
  {
    \int_case:nnF {#1}
      {
        { 2 } { \if_false: \else: }
        { 3 } { \if_true: }
        { 4 } { \if_case:w 0 ~ }
      }
      { \__unravel_final_bad: }
  }
```

(*End of definition for* `\__unravel_final_test:` *and* `\__unravel_final_bad:`*.*)

`\__unravel_final_after_assignment:`

Salvage any remaining `\afterassignment` token.

```
\cs_new_protected:Npn \__unravel_final_after_assignment:
  {
    \gtl_if_empty:NF \g__unravel_after_assignment_gtl
      { \gtl_head_do:NN \g__unravel_after_assignment_gtl \tex_afterassignment:D }
  }
```

(*End of definition for* `\__unravel_final_after_assignment:`*.*)

2.17 Messages

```
\msg_new:nnnn { unravel } { prev-input }
  { Internal~error:~unexpected~type~of~‘‘prev_input’’~entry. }
  {
    Found~type~#2~instead~of~#1~to~assign~to~variable~#3.~Contents:\\
    \iow_indent:n {#4}
  }
\msg_new:nnn { unravel } { unknown-primitive }
  { Internal~error:~the~primitive~’#1’~is~not~known. }
\msg_new:nnn { unravel } { extra-fi-or-else }
  { Extra~fi,~or,~or~else. }
\msg_new:nnn { unravel } { missing-dollar }
  { Missing~dollar~inserted. }
\msg_new:nnn { unravel } { unknown-expandable }
  { Internal~error:~the~expandable~command~’#1’~is~not~known. }
\msg_new:nnn { unravel } { missing-font-id }
  { Missing~font~identifier.~\iow_char:N\\nullfont~inserted. }
\msg_new:nnn { unravel } { missing-rparen }
  { Missing~right~parenthesis~inserted~for~expression. }
\msg_new:nnn { unravel } { missing-cs }
```

```
  { Missing~control~sequence.~\iow_char:N\\inaccessible~inserted. }
\msg_new:nnn { unravel } { missing-box }
  { Missing~box~inserted. }
\msg_new:nnn { unravel } { missing-to }
  { Missing~keyword~’to’~inserted. }
\msg_new:nnn { unravel } { improper-leaders }
  { Leaders~not~followed~by~proper~glue. }
\msg_new:nnn { unravel } { extra-close }
  { Extra~right~brace~or~\iow_char:N\\endgroup. }
\msg_new:nnn { unravel } { off-save }
  { Something~is~wrong~with~groups. }
\msg_new:nnn { unravel } { hrule-bad-mode }
  { \iow_char\\hrule~used~in~wrong~mode. }
\msg_new:nnn { unravel } { invalid-mode }
  { Invalid~mode~for~this~command. }
\msg_new:nnn { unravel } { color-stack-action-missing }
  { Missing~color~stack~action. }
\msg_new:nnn { unravel } { action-type-missing }
  { Missing~action~type. }
\msg_new:nnn { unravel } { identifier-type-missing }
  { Missing~identifier~type. }
\msg_new:nnn { unravel } { destination-type-missing }
  { Missing~destination~type. }
\msg_new:nnn { unravel } { erroneous-prefixes }
  { Prefixes~appplied~to~non-assignment~command. }
\msg_new:nnn { unravel } { improper-setbox }
  { \iow_char:N\\setbox~while~fetching~base~of~an~accent. }
\msg_new:nnn { unravel } { after-advance }
  {
    Missing~register~after~\iow_char:N\\advance,~
    \iow_char:N\\multiply,~or~\iow_char:N\\divide.
  }
\msg_new:nnn { unravel } { bad-unless }
  { \iow_char:N\\unless~not~followed~by~conditional. }
\msg_new:nnn { unravel } { runaway-if }
  { Runaway~\iow_char:N\\if...~Exiting~\iow_char:N\\unravel }
\msg_new:nnn { unravel } { runaway-macro-parameter }
  {
    Runaway~macro~parameter~\# #2~after \\\\
    \iow_indent:n {#1}
  }
\msg_new:nnn { unravel } { runaway-text }
  { Runaway~braced~argument~for~TeX~primitive.~Exiting~\iow_char:N\\unravel }
\msg_new:nnn { unravel } { extra-or }
  { Extra~\iow_char:N\\or. }
\msg_new:nnn { unravel } { missing-equals }
  { Missing~equals~for~\iow_char:N\\ifnum~or~\iow_char:N\\ifdim. }
\msg_new:nnn { unravel } { internal }
  { Internal~error:~’#1’.~\ Please~report. }
\msg_new:nnn { unravel } { not-implemented }
  { The~following~feature~is~not~implemented:~’#1’. }
\msg_new:nnn { unravel } { endinput-ignored }
  { The~primitive~\iow_char:N\\endinput~was~ignored. }
\msg_new:nnn { unravel } { missing-something }
```

```
  { Something~is~missing,~sorry! }
\msg_new:nnn { unravel } { nested-unravel }
  { The~\iow_char:N\\unravel~command~may~not~be~nested. }
\msg_new:nnnn { unravel } { tex-error }
  { TeX~sees~"#1"~and~throws~an~error:\\\\ \iow_indent:n {#2} }
  {
    \tl_if_empty:nTF {#3}
      { TeX~provides~no~further~help~for~this~error. }
      { TeX’s~advice~is:\\\\ \iow_indent:n {#3} }
  }
\msg_new:nnnn { unravel } { tex-fatal }
  { TeX~sees~"#1"~and~throws~a~fatal~error:\\\\ \iow_indent:n {#2} }
  {
    \tl_if_empty:nTF {#3}
      { TeX~provides~no~further~help~for~this~error. }
      { TeX’s~advice~is:\\\\ \iow_indent:n {#3} }
  }
\msg_new:nnnn { unravel } { runaway-unravel }
  { Runaway~\iow_char:N\\unravel,~so~\iow_char:N\\relax~inserted. }
  {
    Some~TeX~command~expects~input~beyond~the~end~of~
    the~argument~of~\iow_char:N\\unravel.
  }
\msg_new:nnn { unravel } { dangling-conditionals }
  { Attempting~to~issue~#1~dangling~conditionals. }
```

Some error messages from TEX itself.

```
\__unravel_tex_msg_new:nnn { forbidden-case }
  {
    You~can’t~use~‘\exp_after:wN \token_to_str:N \l__unravel_head_tl’~in~
    \mode_if_vertical:TF { vertical }
      {
        \mode_if_horizontal:TF { horizontal }
          { \mode_if_math:TF { math } { no } }
      } ~ mode.
  }
  {
    Sorry,~but~I’m~not~programmed~to~handle~this~case;~
    I’ll~just~pretend~that~you~didn’t~ask~for~it.~
    If~you’re~in~the~wrong~mode,~you~might~be~able~to~
    return~to~the~right~one~by~typing~‘I\iow_char:N\}’~or~
    ‘I\iow_char:N\$’~or~‘I\iow_char:N\\par’.
  }
\__unravel_tex_msg_new:nnn { incompatible-mag }
  {
    Incompatible~magnification~
    ( \int_to_arabic:n { \__unravel_mag: } );~
    the~previous~value~will~be~retained
  }
  {
    I~can~handle~only~one~magnification~ratio~per~job.~So~I’ve~
    reverted~to~the~magnification~you~used~earlier~on~this~run.
  }
\__unravel_tex_msg_new:nnn { illegal-mag }
  {
    Illegal~magnification~has~been~changed~to~1000~
    ( \int_to_arabic:n { \__unravel_mag: } )
  }
  { The~magnification~ratio~must~be~between~1~and~32768. }
\__unravel_tex_msg_new:nnn { missing-number }
  { Missing~number,~treated~as~zero }
  {
    A~number~should~have~been~here;~I~inserted~‘0’.~
    If~you~can’t~figure~out~why~I~needed~to~see~a~number,~
    look~up~‘weird~error’~in~the~index~to~The~TeXbook.
  }
\__unravel_tex_msg_new:nnn { the-cannot }
  { You~can’t~use~‘\tl_to_str:N\l__unravel_head_tl’~after~\iow_char:N\\the }
  { I’m~forgetting~what~you~said~and~using~zero~instead. }
\__unravel_tex_msg_new:nnn { incompatible-units }
  { Incompatible~glue~units }
  { I’m~going~to~assume~that~1mu=1pt~when~they’re~mixed. }
\__unravel_tex_msg_new:nnn { missing-mu }
  { Illegal~unit~of~measure~(mu~inserted) }
  {
    The~unit~of~measurement~in~math~glue~must~be~mu.~
    To~recover~gracefully~from~this~error,~it’s~best~to~
    delete~the~erroneous~units;~e.g.,~type~‘2’~to~delete~
    two~letters.~(See~Chapter~27~of~The~TeXbook.)
  }
\__unravel_tex_msg_new:nnn { missing-pt }
  { Illegal~unit~of~measure~(pt~inserted) }
  {
    Dimensions~can~be~in~units~of~em,~ex,~in,~pt,~pc,~
    cm,~mm,~dd,~cc,~nd,~nc,~bp,~or~sp;~but~yours~is~a~new~one!~
    I’ll~assume~that~you~meant~to~say~pt,~for~printer’s~points.~
    To~recover~gracefully~from~this~error,~it’s~best~to~
    delete~the~erroneous~units;~e.g.,~type~‘2’~to~delete~
    two~letters.~(See~Chapter~27~of~The~TeXbook.)
  }
\__unravel_tex_msg_new:nnn { missing-lbrace }
  { Missing~\iow_char:N\{~inserted }
  {
    A~left~brace~was~mandatory~here,~so~I’ve~put~one~in.~
    You~might~want~to~delete~and/or~insert~some~corrections~
    so~that~I~will~find~a~matching~right~brace~soon.~
    (If~you’re~confused~by~all~this,~try~typing~‘I\iow_char:N\}’~now.)
  }
\__unravel_tex_msg_new:nnn { extra-endcsname }
  { Extra~\token_to_str:c{endcsname} }
  { I’m~ignoring~this,~since~I~wasn’t~doing~a~\token_to_str:c{csname}. }
\__unravel_tex_msg_new:nnn { missing-endcsname }
  { Missing~\token_to_str:c{endcsname}~inserted }
  {
    The~control~sequence~marked~<to~be~read~again>~should~
    not~appear~between~\token_to_str:c{csname}~and~
    \token_to_str:c{endcsname}.
  }
\__unravel_tex_msg_new:nnn { missing-delim }
```

```
  { Missing~delimiter~(.~inserted) }
  {
    I~was~expecting~to~see~something~like~`(’~or~`\token_to_str:N\{’~or~
    `\token_to_str:N\}’~here.~If~you~typed,~e.g.,~
    `\{’~instead~of~`\token_to_str:N\{’,~you~
    should~probably~delete~the~`\{’~by~typing~`1’~now,~so~that~
    braces~don’t~get~unbalanced.~Otherwise~just~proceed.~
    Acceptable~delimiters~are~characters~whose~\token_to_str:c{delcode}~is~
    nonnegative,~or~you~can~use~`\token_to_str:c{delimiter}~<delimiter~code>’.
  }
```

Fatal TeX error messages.

```
\__unravel_tex_msg_new:nnn { cannot-read }
  { ***~(cannot~\iow_char:N\\read~from~terminal~in~nonstop~modes) }
  { }
\__unravel_tex_msg_new:nnn { file-error }
  { ***~(job~aborted,~file~error~in~nonstop~mode) }
  { }
\__unravel_tex_msg_new:nnn { interwoven-preambles }
  { (interwoven~alignment~preambles~are~not~allowed) }
  { }
```

Restore catcodes to their original values.

```
\__unravel_setup_restore:
⟨/package⟩
```