The l3doc class – experimental*

The LaTeX3 Project†

Released 2023-12-11

目录

1 Introduction **3**

2 Features of other packages **3**

2.1 The hypdoc package 3

2.2 The docmfp package 3

2.3 The xdoc2 package 4

2.4 The gmdoc package 4

3 Problems & Todo **4**

4 Documentation **5**

4.1 Configuration 5

4.2 Class options 5

4.3 Partitioning documentation and implementation 6

4.4 General text markup 7

4.5 Describing functions in the documentation 8

4.6 Describing functions in the implementation 10

4.7 Keeping things consistent 11

4.8 Documenting templates 12

*根据广泛需求，我们现在发布了这个实验性类的文档。但请注意，它绝不是最终版本，并且很有可能会经历修改，甚至是不兼容的修改！因此，如果类发生变化，可能需要进行更新才能继续使用。

†https://www.latex-project.org/latex3/

5 l3doc implementation 13
5.1 Variables 13
5.2 Variants and helpers 18
5.3 Messages 26
5.4 Options and configuration 27
5.5 Class and package loading 28
5.6 Configuration and tweaks 29
5.7 Design 30
5.8 Text markup 32
5.9 Implementing text markup 37
5.9.1 Common between `macro` and `function` 40
5.9.2 The `function` environment 46
5.9.3 The `macro` environment 56
5.9.4 Misc 67
5.9.5 NB and NOTE 68
5.10 Footnote support 69
5.11 Documenting templates 70
5.12 Inheriting doc 71
5.12.1 The `macrocode` environment 76
5.13 At end document 78
5.14 Indexing 81
5.14.1 Necessary patching 81
5.14.2 Userspace commands 82
5.14.3 Internal index commands 83
5.14.4 Finding sort-key and module 87
5.15 Change history 91
5.16 Default configuration 91
5.17 Internal macros for LaTeX3 sources 92
5.18 Math extras 93
5.19 Makeindex configuration 93

Index 94

1 Introduction

Code and documentation for this class have been written prior to the change of doc from version 2 to version 3, which already shows how far behind this class currently is. So take the following warning seriously please:

It is much less stable than the main expl3 packages.
Use at own risk!

This is an ad-hoc class for documenting the expl3 bundle, a collection of modules or packages that make up LaTeX3's programming environment. Eventually it will replace the ltxdoc class for LaTeX3, but not before the good ideas in hypdoc, xdoc2, docmfp, and gmdoc are incorporated.

It is written as a "self-contained" docstrip file: executing `latex l3doc.dtx` generates the `l3doc.cls` file and typesets this documentation; execute `tex l3doc.dtx` to only generate `l3doc.cls`.

2 Features of other packages

This class builds on the ltxdoc class and the doc package, but in the time since they were originally written some improvements and replacements have appeared that we would like to use as inspiration.

These packages or classes are hypdoc, docmfp, gmdoc, and xdoc. I have summarised them below in order to work out what sort of features we should aim at a minimum for l3doc.

2.1 The hypdoc package

This package provides hyperlink support for the doc package. I have included it in this list to remind me that cross-referencing between documentation and implementation of methods is not very good. (*E.g.*, it would be nice to be able to automatically hyperlink the documentation for a function from its implementation and vice-versa.)

2.2 The docmfp package

- Provides `\DescribeRoutine` and the `routine` environment (*etc.*) for MetaFont and MetaPost code.

- Provides `\DescribeVariable` and the `variable` environment (*etc.*) for more general code.
- Provides `\Describe` and the `Code` environment (*etc.*) as a generalisation of the above two instantiations.
- Small tweaks to the DocStrip system to aid non-LaTeX use.

2.3 The xdoc2 package

- Two-sided printing.
- `\NewMacroEnvironment`, `\NewDescribeEnvironment`; similar idea to docmfp but more comprehensive.
- Tons of small improvements.

2.4 The gmdoc package

Radical re-implementation of doc as a package or class.

- Requires no `\begin{macrocode}` blocks!
- Automatically inserts `\begin{macro}` blocks!
- And a whole bunch of other little things.

3 Problems & Todo

Problems at the moment: (1) not flexible in the types of things that can be documented; (2) no obvious link between the `\begin{function}` environment for documenting things to the `\begin{macro}` function that's used analogously in the implementation.

The `macro` should probably be renamed to `function` when it is used within an implementation section. But they should have the same syntax before that happens!

Furthermore, we need another "layer" of documentation commands to account for "user-macro" as opposed to "code-functions"; the expl3 functions should be documented differently, probably, to the ltcmd user macros (at least in terms of indexing).

In no particular order, a list of things to do:

- Rename `function`/`macro` environments to better describe their use.

- Generalise `function`/`macro` for documenting "other things", such as environment names, package options, even keyval options.
- New function like `\part` but for files (remove awkward "File" as `\partname`).
- Something better to replace `\StopEventually`; I'm thinking two environments `documentation` and `implementation` that can conditionally typeset/ignore their material. (This has been implemented but needs further consideration.)
- Hyperlink documentation and implementation of macros (see the DTX file of svn-multi v2 as an example). This is partially done, now, but should be improved.

4 Documentation

4.1 Configuration

Before class options are processed, l3doc loads a configuration file `l3doc.cfg` if it exists, allowing you to customise the behaviour of the class without having to change the documentation source files.

For example, to produce documentation on letter-sized paper instead of the default A4 size, create `l3doc.cfg` and include the line

```
\PassOptionsToClass{letterpaper}{l3doc}
```

By default, l3doc selects the `T1` font encoding and loads the Latin Modern fonts. To prevent this, use the class option `cm-default`.

4.2 Class options

The class recognises a number of options, some of which are generally useful and some of which are aimed squarely at use by the kernel team only.

`full` `onlydoc` When the `full` option is set (the standard setting), both the documentation and implementation parts of the source are typeset. If on the other hand the `onlydoc` option is set, only the documentation part is typeset.

`lm-default` Selects whether the standard font set up is Latin Modern in the `T1` encoding (the standard setting) or leaves the font setup unchanged.

`kernel` Determines whether l3doc treats `\__kernel_` commands and `\(cgl)__kernel_` variables as allowable in code. In general, *no* internal material from outside the current module is allowed. However, for bootstrapping the expl3 kernel, a small

number of cross-module functions are needed. To suppress the error message that would otherwise arise, the class option `kernel` may be given.

`check` When the `check` option is given, the class will record all commands defined and documented in a ⟨*name*⟩`.cmds` file. This will show which are both documented and defined, which are only documented and which are only defined. (Here, "defined" means listed using a `macro` or `variable` environment in the implementation part of the source file).

`checktest` When `checktest` is given as an option, the class will check that each function entry in the implementation part of the source is marked using `\UnitTest`.

`show-notes` `hide-notes` These complementary options determine if the information given using the `\NB` and `\NOTE` commands is printed.

`cs-break` `cs-break-nohyphen` The commands `\cmd` and `\cs` allow hyphenation of control sequences after (most) underscores. By default, a hyphen is used to mark the hyphenation, but this can be changed with the `cs-break-nohyphen` class option. To disable hyphenation of control sequences entirely, use `cs-break = false`.

4.3 Partitioning documentation and implementation

`doc` uses the `\OnlyDocumentation`/`\AlsoImplementation` macros to guide the use of `\StopEventually{}`, which is intended to be placed to partition the documentation and implementation within a single `.dtx` file.

This isn't very flexible, since it assumes that we *always* want to print the documentation. For the `expl3` sources, I wanted to be able to input `.dtx` files in two modes: only displaying the documentation, and only displaying the implementation. For example:

```
\DisableImplementation
\DocInput{l3basics,l3prg,...}
\EnableImplementation
\DisableDocumentation
\DocInputAgain
```

The idea being that the entire `expl3` bundle can be documented, with the implementation included at the back. Now, this isn't perfect, but it's a start.

Use `\begin{documentation}...\end{documentation}` around the documentation, and `\begin{implementation}...\end{implementation}` around the implementation. The `\EnableDocumentation`/`\EnableImplementation` causes them to

be typeset when the `.dtx` file is `\DocInput`; use `\DisableDocumentation`/`\DisableImplementatio` to omit the contents of those environments.

Note that `\DocInput` now takes comma-separated arguments, and `\DocInputAgain` can be used to re-input all `.dtx` files previously input in this way.

4.4 General text markup

Many of the commands in this section come from ltxdoc with some improvements.

`\cmd` `\cmd [⟨options⟩] ⟨control sequence⟩`
`\cs` `\cs [⟨options⟩] {⟨csname⟩}`

These commands are provided to typeset control sequences. `\cmd\foo` produces "`\foo`" and `\cs{foo}` produces the same. In general, `\cs` is more robust since it doesn't rely on catcodes being "correct" and is therefore recommended.

These commands are aware of the `@@` l3docstrip syntax and replace such instances correctly in the typeset documentation. This only happens after a `%<@@=`⟨*module*⟩`>` declaration.

Additionally, commands can be used in the argument of `\cs`. For instance, `\cs{\meta{name}:\meta{signature}}` produces `\`⟨*name*⟩`:`⟨*signature*⟩.

The ⟨*options*⟩ are a key–value list which can contain the following keys:

- `index=`⟨*name*⟩: the ⟨*csname*⟩ is indexed as if one had written `\cs{`⟨*name*⟩`}`.
- `no-index`: the ⟨*csname*⟩ is not indexed.
- `module=`⟨*module*⟩: the ⟨*csname*⟩ is indexed in the list of commands from the ⟨*module*⟩; the ⟨*module*⟩ can in particular be `TeX` for "TEX and LATEX 2ε" commands, or empty for commands which should be placed in the main index. By default, the ⟨*module*⟩ is deduced automatically from the command name.
- `replace` is a boolean key (`true` by default) which indicates whether to replace `@@` as l3docstrip does.

These commands allow hyphenation of control sequences after (most) underscores. By default, a hyphen is used to mark the hyphenation, but this can be changed with the `cs-break-nohyphen` class option. To disable hyphenation of control sequences entirely, use `cs-break = false`.

\tn `\tn [⟨options⟩] {⟨csname⟩}`

Analoguous to `\cs` but intended for "traditional" TEX or LATEX 2ε commands; they are indexed accordingly. This is in fact equivalent to `\cs [module=TeX, replace=false, ⟨options⟩] {⟨csname⟩}`.

\meta `\meta {⟨name⟩}`

`\meta` typesets the ⟨*name*⟩ italicised in ⟨*angle brackets*⟩. Within a `function` environment or similar, angle brackets `<...>` are set up to be a shorthand for `\meta{...}`.

This function has additional functionality over its ltxdoc versions; underscores can be used to subscript material as in math mode. For example, `\meta{arg_{xy}}` produces "⟨arg_{xy}⟩".

\Arg `\Arg {⟨name⟩}`
\marg
\oarg
\parg

Typesets the ⟨*name*⟩ as for `\meta` and wraps it in braces.

The `\marg`/`\oarg`/`\parg` versions follow from ltxdoc in being used for "mandatory" or "optional" or "picture" brackets as per LATEX 2ε syntax.

\file `\pkg {⟨name⟩}`
\env
\pkg
\cls

These all take one argument and are intended to be used as semantic commands for representing files, environments, package names, and class names, respectively.

\NB `\NB {⟨tag⟩} {⟨comments⟩}`
\NOTE `\begin{NOTE} {⟨tag⟩}`
⟨*comments*⟩
`\end{NOTE}`

Make notes in the source that are not typeset by default. When the `show-notes` class option is active, the comments are typeset in a detokenized and verbatim mode, respectively.

4.5 Describing functions in the documentation

function (*env.*)
variable (*env.*)
syntax (*env.*)

Two heavily-used environments are defined to describe expl3 functions and variables. If describing a variable, use the latter environment; it behaves identically to the `function` environment. Both of the above environments are typically combined with the `syntax` environment, to describe their syntax.

```
\begin{function}{\package_function_one:N, \package_function_two:n}
  \begin{syntax}
    \cs{package_function_one:N} \meta{cs}
    \cs{package_function_two:n} \marg{Argument}
  \end{syntax}
Descriptive text here ...
\end{function}
```

\package_function_one:N \package_function_one:N ⟨*cs*⟩
\package_function_two:n \package_function_two:n {⟨*Argument*⟩}
Descriptive text here ⋯

Function environments take an optional argument to indicate whether the function(s) it describes are expandable (use `EXP`) or restricted-expandable (use `rEXP`) or defined in conditional forms (use `TF`, `pTF`, or `noTF`). Note that `pTF` implies `EXP` since predicates must always be expandable, and that `noTF` means that the function without `TF` should be documented in addition to `TF`. For the conditional forms `TF` and `pTF`, the argument of the `function` environment is *not* in fact a command that exists: in the example below, `\tl_if_empty:N` does not exist, but its conditional forms `\tl_if_empty:NT`, `\tl_if_empty:NF`, `\tl_if_empty:NTF` and predicate form `\tl_if_empty_p:N` exist:

```
\begin{function}[pTF]{\tl_if_empty:N, \tl_if_empty:c}
  \begin{syntax}
    \cs{tl_if_empty_p:N} \meta{tl~var}
    \cs{tl_if_empty:NTF} \meta{tl~var} \Arg{true code} \Arg{false code}
  \end{syntax}
  Tests if the \meta{token list variable} is entirely empty
  (\emph{i.e.}~contains no tokens at all).
\end{function}
```

\tl_if_empty_p:N ⋆ \tl_if_empty_p:N ⟨*tl var*⟩
\tl_if_empty_p:c ⋆ \tl_if_empty:NTF ⟨*tl var*⟩ {⟨*true code*⟩}
\tl_if_empty:NTF ⋆ {⟨*false code*⟩}
\tl_if_empty:c ⋆ Tests if the ⟨*token list variable*⟩ is entirely empty (*i.e.* contains no tokens at all).

texnote (*env.*) This environment is used to call out sections within `function` and similar environments that are only of interest to seasoned TEX developers.

4.6 Describing functions in the implementation

macro (*env.*) The well-used environment from LATEX 2ε for marking up the implementation of macros/functions remains the `macro` environment. Some changes in l3doc: it now accepts comma-separated lists of functions, to avoid a very large number of consecutive `\end{macro}` statements. Spaces and new lines are ignored (the option `[verb]` prevents this).

```
% \begin{macro}{\foo:N, \foo:c}
%   \begin{macrocode}
... code for \foo:N and \foo:c ...
%   \end{macrocode}
% \end{macro}
```

If you are documenting an auxiliary macro, it's generally not necessary to highlight it as much and you also don't need to check it for, say, having a test function and having a documentation chunk earlier in a `function` environment. l3doc will pick up these cases from the presence of `__` in the name, or you may force marking as internal by using `\begin{macro}[int]` to mark it as such. The margin call-out is then printed in grey for such cases.

For documenting expl3-type conditionals, you may also pass this environment a `TF` option (and omit it from the function name) to denote that the function is provided with `T`, `F`, and `TF` suffixes. A similar `pTF` option prints both `TF` and `_p` predicate forms. An option `noTF` prints both the `TF` forms and a form with neither `T` nor `F`, to document functions such as `\prop_get:NN` which also have conditional forms (`\prop_get:NNTF`).

In a very small number of cases, there is no user documentation for a "public" function. In these rare cases, the option `no-user-doc` may be added to suppress the undefined reference that would otherwise then arises.

`\TestFiles` `\TestFiles{`⟨*list of files*⟩`}` is used to indicate which test files are used for the current code; they are printed in the documentation.

`\UnitTested` Within a `macro` environment, it is a good idea to mark whether a unit test has been created for the commands it defines. This is indicated by writing `\UnitTested` anywhere within `\begin{macro}` …`\end{macro}`.

If the class option `checktest` is enabled, then it is an *error* to have a `macro` environment without a call to `Testfiles`. This is intended for large packages such as expl3 that should have absolutely comprehensive tests suites and whose authors may not always be as sharp at adding new tests with new code as they should be.

`\TestMissing` If a function is missing a test, this may be flagged by writing (as many times as needed) `\TestMissing {`⟨*explanation of test required*⟩`}`. These missing tests are summarised in the listing printed at the end of the compilation run.

`variable` (*env.*) When documenting variable definitions, use the `variable` environment instead. Here it behaves identically to the `macro` environment, except that if the class option `checktest` is enabled, variables are not required to have a test file.

`arguments` (*env.*) Within a `macro` environment, you may use the `arguments` environment to describe the arguments taken by the function(s). It behaves like a modified enumerate environment.

```
% \begin{macro}{\foo:nn, \foo:VV}
% \begin{arguments}
%   \item Name of froozle to be frazzled
%   \item Name of muble to be jubled
% \end{arguments}
%   \begin{macrocode}
... code for \foo:nn and \foo:VV ...
%   \end{macrocode}
% \end{macro}
```

4.7 Keeping things consistent

Whenever a function is either documented or defined with `function` and `macro` respectively, its name is stored in a sequence for later processing.

At the end of the document (*i.e.*, after the `.dtx` file has finished processing), the list of names is analysed to check whether all defined functions have been documented and vice versa. The results are printed in the console output.

If you need to do more serious work with these lists of names, take a look at the implementation for the data structures and methods used to store and access them directly.

4.8 Documenting templates

The following macros are provided for documenting templates; might end up being something completely different but who knows.

`\begin{TemplateInterfaceDescription}` {⟨*template type name*⟩}
  `\TemplateArgument{none}{---}`
OR ONE OR MORE OF THESE:
  `\TemplateArgument` {⟨*arg no*⟩} {⟨*meaning*⟩}
AND
`\TemplateSemantics`
  ⟨*text describing the template type semantics*⟩
`\end{TemplateInterfaceDescription}`

`\begin{TemplateDescription}` {⟨*template type name*⟩} {⟨*name*⟩}
ONE OR MORE OF THESE:
  `\TemplateKey` {⟨*key name*⟩} {⟨*type of key*⟩}
    {⟨*textual description of meaning*⟩}
    {⟨*default value if any*⟩}
AND
`\TemplateSemantics`
  ⟨*text describing special additional semantics of the template*⟩
`\end{TemplateDescription}`

`\begin{InstanceDescription}` [⟨*text to specify key column width (optional)*⟩]
      {⟨*template type name*⟩}{⟨*instance name*⟩}{⟨*template name*⟩}
ONE OR MORE OF THESE:
  `\InstanceKey` {⟨*key name*⟩} {⟨*value*⟩}
AND
`\InstanceSemantics`
  ⟨*text describing the result of this instance*⟩
`\end{InstanceDescription}`

5 l3doc implementation

```
1 ⟨*class⟩
2 ⟨@@=codedoc⟩
```

5.1 Variables

\g_docinput_clist The list of files which have been input through \DocInput.

```
3 \clist_new:N \g_docinput_clist
```

(*End of definition for* \g_docinput_clist. *This variable is documented on page* ??.)

\g_doc_functions_seq
\g_doc_macros_seq All functions documented through function, and all macros introduced through macro. They can be compared to see what documentation or code is missing.

```
4 \seq_new:N \g_doc_functions_seq
5 \seq_new:N \g_doc_macros_seq
```

(*End of definition for* \g_doc_functions_seq *and* \g_doc_macros_seq. *These variables are documented on page* ??.)

\l__codedoc_detect_internals_bool
\l__codedoc_detect_internals_tl If true, l3doc will check for use of internal commands __⟨*pkg*⟩_… from other packages in the argument of the macro environment, and in the code typeset in macrocode environments, but not in \cs. Also a token list to store temporary data for this purpose.

```
6 \bool_new:N \l__codedoc_detect_internals_bool
7 \bool_set_true:N \l__codedoc_detect_internals_bool
8 \tl_new:N \l__codedoc_detect_internals_tl
9 \tl_new:N \l__codedoc_detect_internals_cs_tl
```

(*End of definition for* \l__codedoc_detect_internals_bool *and* \l__codedoc_detect_internals_tl.)

\l__codedoc_output_coffin The function environment is typeset by combining coffins containing various pieces (function names, description, *etc.*) into this coffin.

```
10 \coffin_new:N \l__codedoc_output_coffin
```

(*End of definition for* \l__codedoc_output_coffin.)

\l__codedoc_functions_coffin
\l__codedoc_descr_coffin
\l__codedoc_syntax_coffin These coffins contain respectively the list of function names (argument of the function environment), the text between \begin{function} and \end{function}, and the syntax given in the syntax environment.

```
11 \coffin_new:N \l__codedoc_functions_coffin
12 \coffin_new:N \l__codedoc_descr_coffin
13 \coffin_new:N \l__codedoc_syntax_coffin
```

(*End of definition for* `\l__codedoc_functions_coffin`, `\l__codedoc_descr_coffin`, *and* `\l__codedoc_syntax_coffin`.)

`\g__codedoc_syntax_box` The contents of the `syntax` environment are typeset in this box before being transferred to `\l__codedoc_syntax_coffin`.

```
\box_new:N \g__codedoc_syntax_box
```

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

`\l__codedoc_in_function_bool` True when inside a `function` or `variable` environment. Used by the `syntax` environment to determine its behaviour.

```
\bool_new:N \l__codedoc_in_function_bool
```

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

`\l__codedoc_long_name_bool` `\l__codedoc_trial_width_dim` The boolean `\l__codedoc_long_name_bool` is `true` if the width `\l__codedoc_trial_width_dim` of the coffin `\l__codedoc_functions_coffin` (containing the current function names) is bigger than the space available in the margin.

```
\bool_new:N \l__codedoc_long_name_bool
\dim_new:N \l__codedoc_trial_width_dim
```

(*End of definition for* `\l__codedoc_long_name_bool` *and* `\l__codedoc_trial_width_dim`.)

`\l__codedoc_nested_macro_int` The nesting of `macro` environments (this is now 0 outside a `macro` environment).

```
\int_new:N \l__codedoc_nested_macro_int
```

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

`\l__codedoc_macro_tested_bool` `\g__codedoc_missing_tests_prop` `\g__codedoc_not_tested_seq` `\g__codedoc_testfiles_seq` A boolean describing whether the current macro has tests, and some global structures which contain information about test files and which tests are missing.

```
\bool_new:N \l__codedoc_macro_tested_bool
\prop_new:N \g__codedoc_missing_tests_prop
\seq_new:N \g__codedoc_not_tested_seq
\seq_new:N \g__codedoc_testfiles_seq
```

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

`\l__codedoc_macro_deprecated_bool` `\l__codedoc_macro_internal_bool` `\l__codedoc_macro_nodoc_bool` `\l__codedoc_macro_TF_bool` `\l__codedoc_macro_pTF_bool` `\l__codedoc_macro_noTF_bool` `\l__codedoc_macro_EXP_bool` `\l__codedoc_macro_rEXP_bool` `\l__codedoc_macro_var_bool` `\l__codedoc_override_module_tl` `\l__codedoc_macro_documented_tl` Contain information about some options of function/macro environments. We initialize `\l__codedoc_override_module_tl` to avoid overriding module names by an empty name (meaning no module).

```
\bool_new:N \l__codedoc_macro_deprecated_bool
\bool_new:N \l__codedoc_macro_internal_bool
\bool_new:N \l__codedoc_macro_nodoc_bool
```

```
\bool_new:N \l__codedoc_macro_TF_bool
\bool_new:N \l__codedoc_macro_pTF_bool
\bool_new:N \l__codedoc_macro_noTF_bool
\bool_new:N \l__codedoc_macro_EXP_bool
\bool_new:N \l__codedoc_macro_rEXP_bool
\bool_new:N \l__codedoc_macro_var_bool
\tl_new:N \l__codedoc_override_module_tl
\tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
\tl_new:N \l__codedoc_macro_documented_tl
```

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

`\g__codedoc_lmodern_bool`
`\g__codedoc_checkfunc_bool`
`\g__codedoc_checktest_bool`
`\g__codedoc_cs_break_bool`
`\g__codedoc_show_notes_bool`
`\g__codedoc_kernel_bool`

Information about package options.

```
\bool_new:N \g__codedoc_lmodern_bool
\bool_new:N \g__codedoc_checkfunc_bool
\bool_new:N \g__codedoc_checktest_bool
\bool_new:N \g__codedoc_kernel_bool
\bool_new:N \g__codedoc_cs_break_bool
\bool_new:N \g__codedoc_show_notes_bool
\bool_gset_true:N \g__codedoc_cs_break_bool
```

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

`\l__codedoc_tmpa_tl`
`\l__codedoc_tmpb_tl`
`\l__codedoc_tmpa_int`
`\l__codedoc_tmpa_seq`

Some temporary variables.

```
\tl_new:N \l__codedoc_tmpa_tl
\tl_new:N \l__codedoc_tmpb_tl
\int_new:N \l__codedoc_tmpa_int
\int_new:N \l__codedoc_tmpa_seq
```

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

`\l__codedoc_names_block_tl`

List of local sequence variables (produced through `\__codedoc_lseq_name:n`), one for each set of variants in a `function` or `macro` environment. More precisely these sequences are named after the base forms, such as `\clist_count:n` or `\clist_count:N` (which are not variants). Each of these sequences have the base name (without any signature) as their first item, followed by the list of variant's signatures, or `\scan_stop:` to denote the absence of signature (no colon).

```
\tl_new:N \l__codedoc_names_block_tl
```

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

\g__codedoc_variants_seq

Stores rather temporarily the list of variants (signatures only) of a function/macro that is being documented. It is global because we need it to keep its value throughout cells of an alignment.

```
\seq_new:N \g__codedoc_variants_seq
```

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

\l__codedoc_names_verb_bool

Set to `true` if the main argument of a macro/function environment should be used as is, without removing any comma or space.

```
\bool_new:N \l__codedoc_names_verb_bool
```

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

\l__codedoc_names_seq

List of functions/environments/… appearing as arguments of a given `function` or `macro` environment. These are the names after conversion of `_@@` and `@@` to `__`⟨*module name*⟩ and other sanitizing.

```
\seq_new:N \l__codedoc_names_seq
```

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

\g__codedoc_nested_names_seq

Collects all macros in nested `macro` environments, to use them in the "End definition" text.

```
\seq_new:N \g__codedoc_nested_names_seq
```

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

\l__codedoc_index_macro_tl
\l__codedoc_index_key_tl
\l__codedoc_index_module_tl
\l__codedoc_index_internal_bool
\l__codedoc_macro_do_not_index_tl

When analyzing a control sequence found within a `macrocode` environment, `\l__codedoc_index_macro_tl` holds the control sequence (partially a string), `\l__codedoc_index_key_tl` holds the future sort key in the index, and `\l__codedoc_index_module_tl` is the subindex in which the control sequence should be listed. `\l__codedoc_index_internal_bool` indicates when the control sequence is internal and should be indexed in a slightly different subindex. Finally, `\l__codedoc_macro_do_not_index_tl` indicates control sequences which should not be indexed in a specifiv `macro` envronment.

```
\tl_new:N \l__codedoc_index_macro_tl
\tl_new:N \l__codedoc_index_key_tl
\tl_new:N \l__codedoc_index_module_tl
\tl_new:N \l__codedoc_macro_do_not_index_tl
\bool_new:N \l__codedoc_index_internal_bool
```

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

\g__codedoc_module_name_tl

The module name, set when reading a line `<@@=`⟨*module*⟩`>`.

```
\tl_new:N \g__codedoc_module_name_tl
```

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

\c__codedoc_iow_rule_tl
\c__codedoc_iow_midrule_tl

40 equal signs.

```
\tl_const:Nn \c__codedoc_iow_rule_tl
  { ======================================== }
\tl_const:Nn \c__codedoc_iow_mid_rule_tl
  { ---------------------------------------- }
```

(*End of definition for* `\c__codedoc_iow_rule_tl` *and* `\c__codedoc_iow_midrule_tl`.)

\l__codedoc_macro_box
\l__codedoc_macro_index_box
\l__codedoc_macro_int

A vertical box in which the names given to the macro environment are typeset, a horizontal box in which we store the targets created by indexing commands, and the number of macros so far (including those from surrounding `macro` environments).

```
\box_new:N \l__codedoc_macro_box
\box_new:N \l__codedoc_macro_index_box
\int_new:N \l__codedoc_macro_int
```

(*End of definition for* `\l__codedoc_macro_box`, `\l__codedoc_macro_index_box`, *and* `\l__codedoc_macro_int`.)

\l__codedoc_cmd_tl
\l__codedoc_cmd_index_tl
\l__codedoc_cmd_module_tl
\l__codedoc_cmd_noindex_bool
\l__codedoc_cmd_replace_bool

Variables used to control the behaviour of `\cmd`, `\cs` and `\tn`.

```
\tl_new:N \l__codedoc_cmd_tl
\tl_new:N \l__codedoc_cmd_index_tl
\tl_new:N \l__codedoc_cmd_module_tl
\bool_new:N \l__codedoc_cmd_noindex_bool
\bool_new:N \l__codedoc_cmd_replace_bool
```

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

\l__codedoc_in_implementation_bool

This boolean is `true` within the `implementation` environment, and `false` anywhere else.

```
\bool_new:N \l__codedoc_in_implementation_bool
```

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

\g__codedoc_typeset_documentation_bool
\g__codedoc_typeset_implementation_bool

These booleans control whether the documentation/implementation should be typeset. By default both should be.

```
\bool_new:N \g__codedoc_typeset_documentation_bool
\bool_new:N \g__codedoc_typeset_implementation_bool
\bool_set_true:N \g__codedoc_typeset_documentation_bool
\bool_set_true:N \g__codedoc_typeset_implementation_bool
```

(*End of definition for* `\g__codedoc_typeset_documentation_bool` *and* `\g__codedoc_typeset_implementation_bool`.)

`\g__codedoc_base_name_tl` The name of the macro which is being documented (without its signature), and a `\l__codedoc_variants_prop` property list mapping base forms of variants to all variants which have the same base form.

```
\tl_new:N \g__codedoc_base_name_tl
\prop_new:N \l__codedoc_variants_prop
```

(*End of definition for* `\g__codedoc_base_name_tl` *and* `\l__codedoc_variants_prop`.)

`\l__codedoc_function_label_clist` Option of a `function` environment which replaces the label that would normally be `\l__codedoc_no_label_bool` inserted by labels for the given list of control sequences. This is only useful to avoid duplicate labels when a function's documentation appears multiple times.

```
\clist_new:N \l__codedoc_function_label_clist
\bool_new:N \l__codedoc_no_label_bool
```

(*End of definition for* `\l__codedoc_function_label_clist` *and* `\l__codedoc_no_label_bool`.)

`\l__codedoc_date_added_tl` Values of some options of the `function` environment.
`\l__codedoc_date_updated_tl`

```
\tl_new:N \l__codedoc_date_added_tl
\tl_new:N \l__codedoc_date_updated_tl
```

(*End of definition for* `\l__codedoc_date_added_tl` *and* `\l__codedoc_date_updated_tl`.)

`\l__codedoc_macro_argument_tl` Save the argument of a `macro` or `function` environment for use in error messages.

```
\tl_new:N \l__codedoc_macro_argument_tl
```

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

```
% \int_new:N \c@CodelineNo
```

5.2 Variants and helpers

`\__codedoc_tmpa:w` Auxiliary macros for temporary use.
`\__codedoc_tmpb:w`

```
\cs_new_eq:NN \__codedoc_tmpa:w ?
\cs_new_eq:NN \__codedoc_tmpb:w ?
```

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

`\seq_set_split:NoV` A few missing variants.
`\tl_to_str:f`

```
\cs_generate_variant:Nn \seq_set_split:Nnn { NoV }
\cs_generate_variant:Nn \tl_to_str:n { f }
```

(*End of definition for* `\seq_set_split:NoV` *and* `\tl_to_str:f`. *These functions are documented on page* ??.)

`\__codedoc_if_almost_str:nTF` Used to test if the argument of `\cmd` or other macros to be indexed is almost a string or not: for instance this is `false` if `#1` contains `\meta{...}`. The surprising `f`-expansion is there to cope with the case of `#1` starting with `\c_backslash_str` which should be expanded and considered to be "normal".

```
\prg_new_protected_conditional:Npnn \__codedoc_if_almost_str:n #1 { TF , T , F }
  {
    \int_compare:nNnTF
      { \tl_count:n {#1} }
      < { \tl_count:e { \tl_to_str:f {#1} } }
      { \prg_return_false: }
      { \prg_return_true: }
  }
\prg_generate_conditional_variant:Nnn \__codedoc_if_almost_str:n { V } { T }
```

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

`\__codedoc_trim_right:Nn` `\__codedoc_trim_right:No` Removes all material after `#2` in the token list variable `#1`. Perhaps combine with `\__codedoc_key_trim_module:n`?

```
\cs_new_protected:Npn \__codedoc_trim_right:Nn #1#2
  {
    \cs_set:Npn \__codedoc_tmp:w ##1 #2 ##2 \q_stop { \exp_not:n {##1} }
    \__kernel_tl_set:Ne #1 { \exp_after:wN \__codedoc_tmp:w #1 #2 \q_stop }
  }
\cs_generate_variant:Nn \__codedoc_trim_right:Nn { No }
```

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

`\__codedoc_str_if_begin:nnTF` `\__codedoc_str_if_begin:ooTF` True if the first string starts with the second.

```
\prg_new_protected_conditional:Npnn \__codedoc_str_if_begin:nn #1#2 { TF , T , F }
  {
    \tl_if_in:ooTF
      { \exp_after:wN \scan_stop: \tl_to_str:n {#1} }
      { \exp_after:wN \scan_stop: \tl_to_str:n {#2} }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn \__codedoc_str_if_begin:nn
  { oo } { TF , T , F }
```

(*End of definition for* `\__codedoc_str_if_begin:nnTF`.)

__codedoc_replace_at_at:N
__codedoc_replace_at_at_aux:Nn

The goal is to replace `@@` by the current module name. We take advantage of this function to also detect internal macros. If there is no ⟨*module name*⟩, do nothing. Otherwise, sanitize the catcodes of `@` and `_`, temporarily change `@@@@` to `aa` with different catcodes and later to `@@`, and replace `__@@` and `_@@` and `@@` by `__`⟨*module name*⟩. The result contains `_` with category code letter because this is what the `macrocode` environment expects. Other use cases can apply `\tl_to_str:n` if needed. Note that we include spaces between the `@` in the code below, since it is also processed through the same replacement rules.

```
\cs_new_protected:Npn \__codedoc_replace_at_at:N #1
  {
    \tl_if_empty:NF \g__codedoc_module_name_tl
      {
        \exp_args:NNo \__codedoc_replace_at_at_aux:Nn
          #1 \g__codedoc_module_name_tl
      }
  }
\cs_new_protected:Npe \__codedoc_replace_at_at_aux:Nn #1#2
  {
    \tl_replace_all:Nnn #1 { \token_to_str:N @ } { @ }
    \tl_replace_all:Nnn #1 { \token_to_str:N _ } { _ }
    \tl_replace_all:Nnn #1 { @ @ @ @ } { \token_to_str:N a a }
    \tl_replace_all:Nnn #1 { _ _ @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 {   _ @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 {     @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 { \token_to_str:N a a } { @ @ }
  }
```

(*End of definition for* `\__codedoc_replace_at_at:N` *and* `\__codedoc_replace_at_at_aux:Nn`.)

__codedoc_detect_internals:N
__codedoc_detect_internals_aux:N
__codedoc_if_detect_internals_ok:NF

After splitting at each `__` and removing the leading item from the sequence (since it does not follow `__`), remove everything after any space or end-of-line to get a good approximation of the control sequence (for the warning message). Then check if that starts with something allowed: `@@` module name and `:` or `_`, or if the relevant boolean is set `kernel_` (it seems safe to assume we will not define a `\__kernel:...` command). For the message itself remove anything after any `_` or `:` (with either catcode) to get a guess of the module name.

```
\cs_new_protected:Npn \__codedoc_detect_internals:N #1
  {
    \bool_if:NT \l__codedoc_detect_internals_bool
      { \__codedoc_detect_internals_aux:N #1 }
```

```
  }
\group_begin:
  \char_set_catcode_active:N \^^M
  \cs_new_protected:Npn \__codedoc_detect_internals_aux:N #1
    {
      \tl_set_eq:NN \l__codedoc_detect_internals_tl #1
      \tl_replace_all:NVn \l__codedoc_detect_internals_tl \c_underscore_str { _ }
      \seq_set_split:NnV \l__codedoc_tmpa_seq { _ _ } \l__codedoc_detect_internals_tl
      \seq_pop_left:NN \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
      \seq_map_variable:NNn \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
        {
          \__codedoc_trim_right:No \l__codedoc_detect_internals_tl
            \c_catcode_active_space_tl
          \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl ^^M
          \__codedoc_if_detect_internals_ok:NF \l__codedoc_detect_internals_tl
            {
              \tl_set_eq:NN \l__codedoc_detect_internals_cs_tl \l__codedoc_detect_internals_
              \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl _
              \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl :
              \__codedoc_trim_right:No \l__codedoc_detect_internals_tl { \token_to_str:N : }
              \msg_warning:nneee { l3doc } { foreign-internal }
                { \tl_to_str:N \l__codedoc_detect_internals_cs_tl }
                { \tl_to_str:N \l__codedoc_detect_internals_tl }
                { \tl_to_str:N \g__codedoc_module_name_tl }
            }
        }
    }
\group_end:
\prg_new_protected_conditional:Npnn \__codedoc_if_detect_internals_ok:N #1 { F }
  {
    \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl _ }
      { \prg_return_true: }
      {
        \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl : }
          { \prg_return_true: }
          {
            \bool_if:NTF \g__codedoc_kernel_bool
              {
                \__codedoc_str_if_begin:ooTF {#1} { kernel _ }
                  { \prg_return_true: }
                  { \prg_return_false: }
              }
```

```
            { \prg_return_false: }
        }
    }
}
```

(*End of definition for* `\__codedoc_detect_internals:N`, `\__codedoc_detect_internals_aux:N`, *and* `\__codedoc_if_detect_internals_ok:NF`.)

`\__codedoc_signature_base_form:n`
`\__codedoc_signature_base_form_aux:n`
`\__codedoc_signature_base_form_aux:w`

Expands to the "base form" of the signature. For instance, given `noxcfvV` it would obtain `nnnNnnn`, or given `ow` it would obtain `nw`. The loop stops at the first token that is not recognized; the rest is enclosed in `\exp_not:n`.

```
\cs_new:Npn \__codedoc_signature_base_form:n #1
  { \__codedoc_signature_base_form_aux:n #1 \q_stop }
\cs_new:Npn \__codedoc_signature_base_form_aux:n #1
  {
    \str_case:nnTF {#1}
      {
        { N } { N }
        { c } { N }
        { n } { n }
        { o } { n }
        { f } { n }
        { e } { n }
        { x } { n }
        { V } { n }
        { v } { n }
      }
      { \__codedoc_signature_base_form_aux:n }
      { \__codedoc_signature_base_form_aux:w #1 }
  }
\cs_new:Npn \__codedoc_signature_base_form_aux:w #1 \q_stop
  { \exp_not:n {#1} }
```

(*End of definition for* `\__codedoc_signature_base_form:n`, `\__codedoc_signature_base_form_aux:n`, *and* `\__codedoc_signature_base_form_aux:w`.)

`\__codedoc_predicate_from_base:n`

Get predicate from a function's base name. The code is not broken by functions with no signature. The `n`-type version can be used for keys and other non-control sequences. The output after `e`-expansion is a string.

```
\cs_new:Npn \__codedoc_predicate_from_base:n #1
  {
    \__codedoc_get_function_name:n {#1}
    \tl_to_str:n { _p: }
```

```
    \__codedoc_get_function_signature:n {#1}
  }
```

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

`\__codedoc_split_function_do:nn`
`\__codedoc_split_function_do:on`
`\__codedoc_get_function_name:n`
`\__codedoc_get_function_signature:n`
`\__codedoc_split_function_auxi:w`
`\__codedoc_split_function_auxii:w`

Similar to internal functions defined in l3basics, but here we operate on strings directly rather than control sequences.

```
\cs_new:Npn \__codedoc_get_function_name:n #1
  { \__codedoc_split_function_do:nn {#1} { \use_i:nnn } }
\cs_new:Npn \__codedoc_get_function_signature:n #1
  { \__codedoc_split_function_do:nn {#1} { \use_ii:nnn } }
\cs_set_protected:Npn \__codedoc_tmpa:w #1
  {
    \cs_new:Npn \__codedoc_split_function_do:nn ##1
      {
        \exp_after:wN \__codedoc_split_function_auxi:w
        \tl_to_str:n {##1} \q_mark \c_true_bool
        #1 \q_mark \c_false_bool
        \q_stop
      }
    \cs_new:Npn \__codedoc_split_function_auxi:w
      ##1 #1 ##2 \q_mark ##3##4 \q_stop ##5
      { \__codedoc_split_function_auxii:w {##5} ##1 \q_mark \q_stop {##2} ##3 }
    \cs_new:Npn \__codedoc_split_function_auxii:w
      ##1##2 \q_mark ##3 \q_stop
      { ##1 {##2} }
  }
\exp_args:No \__codedoc_tmpa:w { \token_to_str:N : }
\cs_generate_variant:Nn \__codedoc_split_function_do:nn { o }
```

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

`\__codedoc_key_get_base:nN`

Get the base form of a function and store it. As part of getting the base form, change trailing `T` or `F` to `TF`, skipping that change if the function contains no colon to avoid changing for instance some names ending in `PDF` or similar. The various letters `z` serve as end-delimiters different from any outcome of `\tl_to_str:n`.

```
\cs_new_protected:Npn \__codedoc_key_get_base:nN #1#2
  {
    \__codedoc_if_almost_str:nTF {#1}
      {
        \__codedoc_key_get_base_TF:nN {#1} \l__codedoc_tmpa_tl
        \__kernel_tl_set:Ne #2
```

```
          { \__codedoc_split_function_do:on \l__codedoc_tmpa_tl { \__codedoc_base_form_aux:n
      }
      { \tl_set:Nn #2 {#1} }
  }
\cs_new:Npe \__codedoc_key_get_base_TF:nN #1#2
  {
    \__kernel_tl_set:Ne #2 { \exp_not:N \tl_to_str:n {#1} }
    \tl_if_in:NoF #2 { \tl_to_str:n {:} }
      { \exp_not:N \prg_break: }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {TF} z }
      { \exp_not:N \prg_break: }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {T} z }
      {
        \tl_put_right:Nn #2 { \tl_to_str:n {F} }
        \exp_not:N \prg_break:
      }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {F} z }
      {
        \tl_put_right:Nn #2 { z }
        \tl_replace_once:Nnn #2 { \tl_to_str:n {F} z } { \tl_to_str:n {TF} }
        \exp_not:N \prg_break:
      }
    \exp_not:N \prg_break_point:
  }
\cs_new:Npn \__codedoc_base_form_aux:nnN #1#2#3
  {
    \exp_not:n {#1}
    \bool_if:NT #3
      {
        \token_to_str:N :
        \bool_lazy_or:nnTF
            { \str_if_eq_p:nn { #1 ~ } { \exp_args } }
            { \str_if_eq_p:nn { #1 ~ } { \exp_last_unbraced } }
          { \exp_not:n {#2} }
          { \__codedoc_signature_base_form:n {#2} }
      }
  }
```

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

`\__codedoc_base_form_signature_do:nnn` Do `#2{#1}` if there is no signature, or if `#1` contains two colons in a row (this covers the weird function `\::N` and so on). Otherwise apply `#3` with the following two

arguments: the base form of #1, and the original signature with an extra pair of braces.

```
\cs_new_protected:Npn \__codedoc_base_form_signature_do:nnn #1#2#3
  {
    \__codedoc_split_function_do:nn {#1}
      { \__codedoc_base_form_aux:nnnnnN {#1} {#2} {#3} }
  }
\cs_new_protected:Npn \__codedoc_base_form_aux:nnnnnN #1#2#3#4#5#6
  {
    \bool_if:NTF #6
      {
        \tl_if_head_eq_charcode:nNTF {#4} :
          { #2 {#1} }
          {
            \use:e
              {
                \exp_not:n {#3}
                { \__codedoc_base_form_aux:nnN {#4} {#5} #6 }
              }
                {#4} {#5}
          }
      }
      { #2 {#1} }
  }
```

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

__codedoc_date_compare_p:nNn
__codedoc_date_compare:nNnTF
__codedoc_date_compare_aux:nnnNnnn
__codedoc_date_compare_aux:w

Expects #1 and #3 to be dates in the format YYYY-MM-DD (but accepts YYYY or YYYY-MM too). Compares them using #2 (one of <, =, >), filling in zeros for missing data.

```
\prg_new_conditional:Npnn \__codedoc_date_compare:nNn #1#2#3 { TF , T , F , p }
  { \__codedoc_date_compare_aux:w #1--- \q_mark #2 #3--- \q_stop }
\cs_new:Npn \__codedoc_date_compare_aux:w
    #1 - #2 - #3 - #4 \q_mark #5 #6 - #7 - #8 - #9 \q_stop
  {
    \__codedoc_date_compare_aux:nnnNnnn
      { \tl_if_empty:nTF {#1} { 0 } {#1} }
      { \tl_if_empty:nTF {#2} { 0 } {#2} }
      { \tl_if_empty:nTF {#3} { 0 } {#3} }
      #5
      { \tl_if_empty:nTF {#6} { 0 } {#6} }
      { \tl_if_empty:nTF {#7} { 0 } {#7} }
```

```
      { \tl_if_empty:nTF {#8} { 0 } {#8} }
  }
\cs_new:Npn \__codedoc_date_compare_aux:nnnNnnn #1#2#3#4#5#6#7
  {
    \int_compare:nNnTF {#1} = {#5}
      {
        \int_compare:nNnTF {#2} = {#6}
          {
            \int_compare:nNnTF {#3} #4 {#7}
              { \prg_return_true: } { \prg_return_false: }
          }
          {
            \int_compare:nNnTF {#2} #4 {#6}
              { \prg_return_true: } { \prg_return_false: }
          }
      }
      {
        \int_compare:nNnTF {#1} #4 {#5}
          { \prg_return_true: } { \prg_return_false: }
      }
    \use_none:n
    \q_stop
  }
```

(*End of definition for* `\__codedoc_date_compare:nNnTF`*,* `\__codedoc_date_compare_aux:nnnNnnn`*, and* `\__codedoc_date_compare_aux:w`.)

`\__codedoc_gprop_name:n`
`\__codedoc_lseq_name:n`

We need to keep track of some information about control sequences (and other strings) that are being (or have been) documented. Some is stored into global props and some into local seqs, whose name does not follow conventions: it is `\g__codedoc` or `\l__codedoc` followed by a space and by the string, which can be arbitrary. We cannot reasonably use a single big `prop` for speed reasons.

```
\cs_new:Npn \__codedoc_gprop_name:n #1 { g__codedoc ~ \tl_to_str:n {#1} }
\cs_new:Npn \__codedoc_lseq_name:n #1 { l__codedoc ~ \tl_to_str:n {#1} }
```

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

5.3 Messages

```
\msg_new:nnnn { l3doc } { no-signature-TF }
  { Function/macro~'#1'~cannot~be~turned~into~a~conditional. }
  {
    A~function~or~macro~environment~with~option~pTF,~TF~or~noTF~
```

```
    received~the~argument~'#1'.~This~function's~name~has~no~
    ':'~hence~it~is~not~clear~where~to~add~'_p'~or~'TF'.~
    Please~follow~expl3~naming~conventions.
  }
\msg_new:nnn { l3doc } { date-format }
  { The~date~'#1'~should~be~given~in~YYYY-MM-DD~format. }
\msg_new:nnn { l3doc } { future-date }
  { The~added/updated~date~'#2'~of~'#1'~is~in~the~future. }
\msg_new:nnn { l3doc } { syntax-nested-function }
  {
    The~'syntax'~environment~should~be~used~in~the~
    innermost~'function'~environment.
  }
\msg_new:nnn { l3doc } { multiple-syntax }
  {
    The~'syntax'~environment~should~only~be~used~once~in~
    a~'function'~environment.
  }
\msg_new:nnn { l3doc } { deprecated-option }
  { The~option~'#1'~has~been~deprecated~for~'#2'. }
\msg_new:nnn { l3doc } { foreign-internal }
  {
    A~control~sequence~of~the~form~'..._#1'~was~used.~
    It~should~only~be~used~in~the~module~'#2'
    \tl_if_empty:nF {#3} { ,~not~in~'#3' } .
  }
```

5.4 Options and configuration

```
\DeclareKeys [ l3doc / options ]
  {
    a5paper .code:n = \@latexerr { Option~not~supported } { } ,
    full .code:n =
      {
        \bool_gset_true:N \g__codedoc_typeset_documentation_bool
        \bool_gset_true:N \g__codedoc_typeset_implementation_bool
      } ,
   onlydoc .code:n =
    {
      \bool_gset_true:N \g__codedoc_typeset_documentation_bool
      \bool_gset_false:N \g__codedoc_typeset_implementation_bool
    } ,
    check .bool_gset:N = \g__codedoc_checkfunc_bool ,
```

```
    checktest .bool_gset:N = \g__codedoc_checktest_bool ,
    kernel .bool_gset:N = \g__codedoc_kernel_bool ,
    stdmodule .bool_gset_inverse:N = \g__codedoc_kernel_bool ,
    lm-default .bool_gset:N = \g__codedoc_lmodern_bool ,
    cs-break .bool_gset_inverse:N = \g__codedoc_cs_break_bool ,
    cs-break-nohyphen .code:n = \PassOptionsToPackage{nohyphen}{underscore} ,
    show-notes .bool_gset:N = \g__codedoc_show_notes_bool,
    hide-notes .bool_gset_inverse:N = \g__codedoc_show_notes_bool
  }
\DeclareUnknownKeyHandler [ l3doc / options ]
  { \PassOptionsToClass { \CurrentOption } { article } }
\SetKeys [ l3doc / options ]
  { full , kernel , check = false , checktest = false , lm-default }
\PassOptionsToClass { a4paper } { article }
```

Input a local configuration file, if it exists, with a message to the console that this has happened. Since we distribute a `.cfg` file with the class, this should usually always be true. Therefore, check for `\ExplMakeTitle` (defined in "our" `.cfg` file) and only output the informational message if it's not found.

```
\msg_new:nnn { l3doc } { input-cfg }
  { Local~config~file~l3doc.cfg~loaded. }
\file_if_exist:nT { l3doc.cfg }
  {
    \file_input:n { l3doc.cfg }
    \cs_if_exist:cF { ExplMakeTitle }
      { \msg_info:nn { l3doc } { input-cfg } }
  }
\ProcessKeyOptions [ l3doc / options ]
```

5.5 Class and package loading

```
\LoadClass{article}
\RequirePackage{doc}
\RequirePackage
  {
    array,
    alphalph,
    amsmath,
    amssymb,
    booktabs,
    color,
    colortbl,
    hologo,
```

```
    enumitem,
    pifont,
    textcomp,
    trace,
    csquotes,
    fancyvrb,
    underscore,
    verbatim
  }
\raggedbottom
```

Depending on the option, load the package lmodern to set the font. Then replace the italic typewriter font with the oblique shape instead; the former makes my skin crawl. (Will, Aug 2011)

```
\bool_if:NT \g__codedoc_lmodern_bool
  {
    \RequirePackage[T1]{fontenc}
    \RequirePackage{lmodern}
    \group_begin:
      \ttfamily
      \DeclareFontShape{T1}{lmtt}{m}{it}{<->ec-lmtto10}{}
    \group_end:
  }
```

Must be last, as usual.

```
\RequirePackage{hypdoc}
```

5.6 Configuration and tweaks

\MakePrivateLetters

A few more letters are "private" in a LaTeX3 programming environment.

```
\cs_gset:Npn \MakePrivateLetters
  {
    \char_set_catcode_letter:N \@
    \char_set_catcode_letter:N \_
    \char_set_catcode_letter:N \:
  }
```

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

CodelineNo

Some configurations which have to do with line numbering.

```
\setcounter{StandardModuleDepth}{1}
\@addtoreset{CodelineNo}{part}
\tl_replace_once:Nnn \theCodelineNo
  { \HDorg@theCodelineNo }
  { \textcolor[gray]{0.5} { \sffamily\tiny\arabic{CodelineNo} } }
```

(*End of definition for* `CodelineNo`. *This function is documented on page* ??.)

`\verbatim` `\endverbatim` In `.dtx` documents, the `verbatim` environment adds extra space because it only removes the first "%" sign, and not the indentation (typically a space). Fix it with fancyvrb:

```
\fvset{gobble=2}
\cs_gset_eq:NN \verbatim \Verbatim
\cs_gset_eq:NN \endverbatim \endVerbatim
```

(*End of definition for* `\verbatim` *and* `\endverbatim`. *These functions are documented on page* ??.)

`\ifnot@excluded` This function tests whether a macro name stored in `\macro@namepart` was excluded from indexing by `\DoNotIndex`. Rather than trying to fix catcodes that come into here, turn everything to string catcodes. This is slightly inefficient as we could have ensured that `\index@excludelist` has string catcodes in the first place.

```
\cs_set_protected:Npn \ifnot@excluded
  {
    \exp_args:Nee \expanded@notin
      { \c_backslash_str \tl_to_str:N \macro@namepart , }
      { \exp_args:NV \tl_to_str:n \index@excludelist }
  }
```

(*End of definition for* `\ifnot@excluded`. *This function is documented on page* ??.)

`\pdfstringnewline` `\__codedoc_pdfstring_newline:w` We avoid some hyperref warnings by making `\\` (almost) trivial in bookmarks: more precisely it might be used with a star and an optional argument, which we thus remove using an ltcmd expandable command. Since there cannot be trailing optional arguments, pick up an extra mandatory one and put it back.

```
\cs_new:Npn \pdfstringnewline { : ~ }
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_newline:w } { s o m } { \pdfstringnewline #3 }
\pdfstringdefDisableCommands
  { \cs_set_eq:NN \\ \__codedoc_pdfstring_newline:w }
```

(*End of definition for* `\pdfstringnewline` *and* `\__codedoc_pdfstring_newline:w`. *This function is documented on page* ??.)

5.7 Design

Increase the text width slightly so that width the standard fonts 72 columns of code may appear in a `macrocode` environment. Increase the marginpar width slightly, for long command names. And increase the left margin by a similar amount.

```
\setlength   \textwidth      { 385pt }
\addtolength \marginparwidth {  30pt }
\addtolength \oddsidemargin  {  20pt }
\addtolength \evensidemargin {  20pt }
```

(These were introduced when article was the documentclass, but I've left them here for now to remind me to do something about them later.)

\list Customise lists.

__codedoc_oldlist:nn

```
\cs_new_eq:NN \__codedoc_oldlist:nn \list
\cs_gset:Npn \list #1 #2
  { \__codedoc_oldlist:nn {#1} { #2 \dim_zero:N \listparindent } }
\setlength \parindent  { 2em }
\setlength \itemindent { 0pt }
\setlength \parskip    { 0pt plus 3pt minus 0pt }
```

(*End of definition for* `\list` *and* `\__codedoc_oldlist:nn`*. This function is documented on page* ??.)

\partname Use "File" as a name in Part titles.

```
\tl_gset:Nn \partname {File}
```

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

\l@section \l@subsection Customise the table of contents (as we have so many sections). Different design and/or structure is called for).

```
\@addtoreset{section}{part}
\cs_gset:Npn \l@section #1#2
  {
    \ifnum \c@tocdepth >\z@
      \addpenalty\@secpenalty
      \addvspace{1.0em \@plus\p@}
      \setlength\@tempdima{2.5em}  % was 1.5em
      \begingroup
        \parindent \z@ \rightskip \@pnumwidth
        \parfillskip -\@pnumwidth
        \leavevmode \bfseries
        \advance\leftskip\@tempdima
        \hskip -\leftskip
        #1\nobreak\hfil \nobreak\hb@xt@\@pnumwidth{\hss #2}\par
      \endgroup
    \fi
  }
\cs_gset:Npn \l@subsection
  { \@dottedtocline{2}{2.5em}{2.3em} }  % #2 = 1.5em
```

(*End of definition for* `\l@section` *and* `\l@subsection`. *These functions are documented on page* ??.)

5.8 Text markup

Make `|` and `"` be "short verb" characters, but not in the document preamble, where an active character may interfere with packages that are loaded. Remove these short-hands at the end of the document before reading the `.aux` file, as they may appear in labels (for instance, l3fp documents an operation `||`).

```
\AtBeginDocument
  {
    \MakeShortVerb \"
    \MakeShortVerb \|
  }
\AtEndDocument
  {
    \DeleteShortVerb \"
    \DeleteShortVerb \|
  }
```

`\eTeX` `\IniTeX` `\Lua` `\LuaTeX` `\pdfTeX` `\XeTeX` `\pTeX` `\upTeX` `\epTeX` `\eupTeX` Some commands for logos.

```
\providecommand*\eTeX{\hologo{eTeX}}
\providecommand*\IniTeX{\hologo{iniTeX}}
\providecommand*\Lua{Lua}
\providecommand*\LuaTeX{\hologo{LuaTeX}}
\providecommand*\pdfTeX{\hologo{pdfTeX}}
\providecommand*\XeTeX{\hologo{XeTeX}}
\providecommand*\pTeX{p\kern-.2em\hologo{TeX}}
\providecommand*\upTeX{up\kern-.2em\hologo{TeX}}
\providecommand*\epTeX{$\varepsilon$-\pTeX}
\providecommand*\eupTeX{$\varepsilon$-\upTeX}
\providecommand*\ConTeXt{\hologo{ConTeXt}}
```

(*End of definition for* `\eTeX` *and others. These functions are documented on page* ??.)

`\cmd` `\cs` `\tn` They rely on a common auxiliary `\__codedoc_cmd:nn` which receives as arguments the options and some tokens whose string representation starts with a backslash (to support cases such as `\cs{pkg_\ldots}`, we do not turn the whole argument into a string).

```
\DeclareDocumentCommand \cmd { O{} m }
  { \__codedoc_cmd:no {#1} { \token_to_str:N #2 } }
\DeclareDocumentCommand \cs  { O{} m }
```

```
  { \__codedoc_cmd:no {#1} { \c_backslash_str #2 } }
\DeclareDocumentCommand \tn  { O{} m }
  {
    \__codedoc_cmd:no
      { module = TeX , replace = false , #1 }
      { \c_backslash_str #2 }
  }
```

(*End of definition for* `\cmd`*,* `\cs`*, and* `\tn`*. These functions are documented on page 7.*)

`\meta` A document-level command.

```
\DeclareDocumentCommand \meta { m }
  { \__codedoc_meta:n {#1} }
```

(*End of definition for* `\meta`*. This function is documented on page 8.*)

`\__codedoc_pdfstring_cmd:w` `\__codedoc_pdfstring_cs:w` `\__codedoc_pdfstring_meta:w` To work within a bookmark, these commands must be expandable.

```
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_cmd:w } { o m } { \token_to_str:N #2 }
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_cs:w }  { o m } { \textbackslash \tl_to_str:n {#2} }
\cs_new:Npn \__codedoc_pdfstring_meta:w #1
  { < \tl_to_str:n {#1} > }
\pdfstringdefDisableCommands
  {
    \cs_set_eq:NN \cmd  \__codedoc_pdfstring_cmd:w
    \cs_set_eq:NN \cs   \__codedoc_pdfstring_cs:w
    \cs_set_eq:NN \tn   \__codedoc_pdfstring_cs:w
    \cs_set_eq:NN \meta \__codedoc_pdfstring_meta:w
  }
```

(*End of definition for* `\__codedoc_pdfstring_cmd:w`*,* `\__codedoc_pdfstring_cs:w`*, and* `\__codedoc_pdfstring_meta:w`*.*)

`\Arg` `\marg` `\oarg` `\parg` `\marg{text}` prints {⟨*text*⟩}, "mandatory argument". `\oarg{text}` prints [⟨*text*⟩], "optional argument". `\parg{te,xt}` prints (⟨*te,xt*⟩), "picture mode argument". Finally, `\Arg` is the same as `\marg`.

```
\newcommand\Arg[1]
  { \texttt{\char`\{} \meta{#1} \texttt{\char`\}} }
\providecommand\marg[1]{ \Arg{#1} }
\providecommand\oarg[1]{ \texttt[ \meta{#1} \texttt] }
\providecommand\parg[1]{ \texttt( \meta{#1} \texttt) }
```

(*End of definition for* `\Arg` *and others. These functions are documented on page 8.*)

`\file` `\env` `\pkg` `\cls`

This list may change...this is just my preference for markup.

```
\DeclareRobustCommand \file {\nolinkurl}
\DeclareRobustCommand \env {\texttt}
\DeclareRobustCommand \pkg {\textsf}
\DeclareRobustCommand \cls {\textsf}
```

(*End of definition for* `\file` *and others. These functions are documented on page 8.*)

`\EnableDocumentation` `\EnableImplementation` `\DisableDocumentation` `\DisableImplementation`

Control whether to typeset the documentation/implementation or not. These simply set two switches.

```
\NewDocumentCommand \EnableDocumentation { }
  { \bool_gset_true:N \g__codedoc_typeset_documentation_bool }
\NewDocumentCommand \EnableImplementation { }
  { \bool_gset_true:N \g__codedoc_typeset_implementation_bool }
\NewDocumentCommand \DisableDocumentation { }
  { \bool_gset_false:N \g__codedoc_typeset_documentation_bool }
\NewDocumentCommand \DisableImplementation { }
  { \bool_gset_false:N \g__codedoc_typeset_implementation_bool }
```

(*End of definition for* `\EnableDocumentation` *and others. These functions are documented on page* ??.)

`documentation` (*env.*) `implementation` (*env.*)

If the documentation/implementation should be typeset, then simply set the boolean `\l__codedoc_in_implementation_bool` which indicates whether we are within the implementation section. Otherwise use `\comment` (and a paired `\endcomment`).

```
\NewDocumentEnvironment { documentation } { }
  {
    \bool_if:NTF \g__codedoc_typeset_documentation_bool
      { \bool_set_false:N \l__codedoc_in_implementation_bool }
      { \comment }
  }
  { \bool_if:NF \g__codedoc_typeset_documentation_bool { \endcomment } }
\NewDocumentEnvironment { implementation } { }
  {
    \bool_if:NTF \g__codedoc_typeset_implementation_bool
      { \bool_set_true:N \l__codedoc_in_implementation_bool }
      { \comment }
  }
  { \bool_if:NF \g__codedoc_typeset_implementation_bool { \endcomment } }
```

`variable` (*env.*) The `variable` environment behaves as a `function` or `macro` environment depending on the part of the document.

```
\DeclareDocumentEnvironment { variable } { O{} +v }
  {
    \bool_if:NTF \l__codedoc_in_implementation_bool
      { \__codedoc_macro:nnw { var , #1 } {#2} }
      { \__codedoc_function:nnw {#1} {#2} }
  }
  {
    \bool_if:NTF \l__codedoc_in_implementation_bool
      { \__codedoc_macro_end: }
      { \__codedoc_function_end: }
  }
```

`function` (*env.*) `macro` (*env.*) Environment for documenting function(s), and environment for documenting the implementation of a macro.

```
\DeclareDocumentEnvironment { function } { O{} +v }
  { \__codedoc_function:nnw {#1} {#2} }
  { \__codedoc_function_end: }
\DeclareDocumentEnvironment { macro } { O{} +v }
  { \__codedoc_macro:nnw {#1} {#2} }
  { \__codedoc_macro_end: }
```

`syntax` (*env.*) Syntax block placed next to the list of functions to illustrate their use. TODO: test that the `syntax` environment is only used inside the `function` environment, and that it only appears once.

```
\NewDocumentEnvironment { syntax } { }
  { \__codedoc_syntax:w }
  {
    \__codedoc_syntax_end:
    \ignorespacesafterend
  }
```

`texnote` (*env.*) Used to describe information destined to TEX experts only.

```
\NewDocumentEnvironment { texnote } { }
  {
    \endgraf
    \vspace{3mm}
    \small\textbf{\TeX~hackers~note:}
  }
  {
```

```
    \vspace{3mm}
  }
```

`arguments` *(env.)* This environment is designed to be used within a `macro` environment to describe the arguments of the macro/function.

```
\NewDocumentEnvironment { arguments } { }
  {
    \enumerate [
      nolistsep ,
      label = \texttt{\#\arabic*} ~ : ,
      labelsep = * ,
    ]
  }
  {
    \endenumerate
  }
```

`\CodedocExplain` `\CodedocExplainEXP` `\CodedocExplainREXP` `\CodedocExplainTF` Explanation of stars and `TF` notations, for use in third-party packages.

```
\NewDocumentCommand { \CodedocExplain } { }
  { \CodedocExplainEXP \ \CodedocExplainREXP \ \CodedocExplainTF }
\NewDocumentCommand { \CodedocExplainEXP } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{expstar}{}}%
    \write \@auxout { \def \string \Codedoc@expstar { } }
    \__codedoc_typeset_exp:\ indicates~fully~expandable~functions,~which~
    can~be~used~within~an~\texttt{e}-type~argument~(inside~an~\tn{expanded}),~
    \texttt{x}-type~argument~(in~plain~\TeX{}~terms,~inside~an~\tn{edef}),~
    as~well~as~within~an~\texttt{f}-type~argument.
  }
\NewDocumentCommand { \CodedocExplainREXP } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{rexpstar}{}}%
    \write \@auxout { \def \string \Codedoc@rexpstar { } }
    \__codedoc_typeset_rexp:\ indicates~
    restricted~expandable~functions,~which~can~be~used~within~an~
    \texttt{x}-type~argument~or~an~\texttt{e}-type~argument,~
    but~cannot~be~fully~expanded~within~an~\texttt{f}-type~argument.
  }
\NewDocumentCommand { \CodedocExplainTF } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{explTF}{}}%
    \write \@auxout { \def \string \Codedoc@explTF { } }
```

```
    \__codedoc_typeset_TF:\ indicates~conditional~(\texttt{if})~functions~
    whose~variants~with~\texttt{T},~\texttt{F}~and~\texttt{TF}~
    argument~specifiers~expect~different~
    \enquote{true}/\enquote{false}~branches.
  }
```

(*End of definition for* `\CodedocExplain` *and others. These functions are documented on page* ??.)

5.9 Implementing text markup

Keys for `\cmd`, `\cs` and `\tn`.

```
\keys_define:nn { l3doc/cmd }
  {
    index     .tl_set:N     = \l__codedoc_cmd_index_tl        ,
    module    .tl_set:N     = \l__codedoc_cmd_module_tl       ,
    no-index  .bool_set:N   = \l__codedoc_cmd_noindex_bool    ,
    replace   .bool_set:N   = \l__codedoc_cmd_replace_bool    ,
  }
```

`\__codedoc_cmd:nn` `\__codedoc_cmd:no`

Apply the key–value ⟨*options*⟩ `#1` after setting some default values. Then (unless `replace=false`) replace `@@` in `#2`, which is a bit tricky: the `_` must be given the catcode expected by `\__codedoc_replace_at_at:N`, but should be reverted to their original catcode (normally active, needed for line-breaking) without rescanning the whole argument. Then typeset the command in `\verbatim@font`, after turning it to harmless characters if needed (and keeping the underscore breakable); in any case, spaces must be turned into `\@xobeysp` and we must use `\@` to avoid longer spaces after a control sequence that ends for instance with a colon (empty signature). Finally, produce an index entry. Indexing is suppressed when `\l__codedoc_cmd_noindex_bool` is true.

```
\cs_new_protected:Npn \__codedoc_cmd:nn #1#2
  {
    \bool_set_false:N \l__codedoc_cmd_noindex_bool
    \bool_set_true:N \l__codedoc_cmd_replace_bool
    \tl_set:Nn \l__codedoc_cmd_index_tl { \q_no_value }
    \tl_set:Nn \l__codedoc_cmd_module_tl { \q_no_value }
    \keys_set:nn { l3doc/cmd } {#1}
    \tl_set:Nn \l__codedoc_cmd_tl {#2}
    \bool_if:NT \l__codedoc_cmd_replace_bool
      {
        \tl_set_rescan:Nnn \l__codedoc_tmpb_tl { } { _ }
        \tl_replace_all:NVn \l__codedoc_cmd_tl \l__codedoc_tmpb_tl { _ }
```

```
      \__codedoc_replace_at_at:N \l__codedoc_cmd_tl
      \tl_replace_all:NnV \l__codedoc_cmd_tl { _ } \l__codedoc_tmpb_tl
    }
```

Typesetting. Note the replacement for the underscore is to permit linebreaks. The `underscore` package adds the linebreak, and the regex results in applying the breakable underscore only to the *last* of a run of underscores, and not if the underscore follows a backslash.

```
  \mode_if_math:T { \mbox }
    {
      \bool_if:NT \l__codedoc_allow_indexing_bool { \__codedoc_target: }
      \verbatim@font
      \__codedoc_if_almost_str:VT \l__codedoc_cmd_tl
        {
          \__kernel_tl_set:Ne \l__codedoc_cmd_tl { \tl_to_str:N \l__codedoc_cmd_tl }
          \bool_if:NT \g__codedoc_cs_break_bool
            {
              \regex_replace_all:nnN
                { ([^\\\_]\_*) \_ ([^\_]) }
                { \1 \c{BreakableUnderscore} \2 }
                \l__codedoc_cmd_tl
            }
        }
      \tl_replace_all:Nnn \l__codedoc_cmd_tl { ~ } { \@xobeysp }
      \l__codedoc_cmd_tl
      \@
    }
```

Indexing.

```
  \bool_if:NT \l__codedoc_allow_indexing_bool
   {
    \bool_if:NF \l__codedoc_cmd_noindex_bool
     {
      \quark_if_no_value:NF \l__codedoc_cmd_index_tl
        {
          \__kernel_tl_set:Ne \l__codedoc_cmd_tl
            { \c_backslash_str \exp_not:o { \l__codedoc_cmd_index_tl } }
        }
      \exp_args:No \__codedoc_key_get:n { \l__codedoc_cmd_tl }
      \quark_if_no_value:NF \l__codedoc_cmd_module_tl
        {
          \__kernel_tl_set:Ne \l__codedoc_index_module_tl
            { \tl_to_str:N \l__codedoc_cmd_module_tl }
```

```
          }
        \__codedoc_special_index_module:ooonN
          { \l__codedoc_index_key_tl }
          { \l__codedoc_index_macro_tl }
          { \l__codedoc_index_module_tl }
          { usage }
          \l__codedoc_index_internal_bool
      }
    }
  }
\cs_generate_variant:Nn \__codedoc_cmd:nn { no }
```

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

`\__codedoc_meta:n`
`\__codedoc_ensuremath_sb:n`
`\__codedoc_meta_original:n`

Store `#1` in `\l__codedoc_tmpa_tl` and replaces every underscore, regardless of its category ("math toggle", "alignment", "superscript", "subscript", "letter", "other", or "active") by `\__codedoc_ensuremath_sb:n` (which creates math subscripts), then runs the code used for `\meta` in doc.sty.

```
\cs_new_protected:Npn \__codedoc_meta:n #1
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#1}
    \tl_map_inline:nn
      { { 3 } { 4 } { 7 } { 8 } { 11 } { 12 } { 13 } }
      {
        \tl_set_rescan:Nnn \l__codedoc_tmpb_tl
          { \char_set_catcode:nn { `_ } {##1} } { _ }
        \tl_replace_all:NVn \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
          { \__codedoc_ensuremath_sb:n }
      }
    \exp_args:NV \__codedoc_meta_original:n \l__codedoc_tmpa_tl
  }
\cs_new_protected:Npn \__codedoc_ensuremath_sb:n #1
  { \ensuremath { \sb {#1} } }
\cs_new_protected:Npn \__codedoc_meta_original:n #1
  {
    \ensuremath \langle
    \mode_if_math:T { \nfss@text }
    {
      \meta@font@select
      \edef \meta@hyphen@restore
        { \hyphenchar \the \font \the \hyphenchar \font }
      \hyphenchar \font \m@ne
```

```
      \language \l@nohyphenation
      #1 \/
      \meta@hyphen@restore
    }
    \ensuremath \rangle
  }
```

(*End of definition for* `\__codedoc_meta:n`, `\__codedoc_ensuremath_sb:n`, *and* `\__codedoc_meta_original:n`.)

5.9.1 Common between `macro` and `function`

`\__codedoc_typeset_exp:`
`\__codedoc_typeset_rexp:`
`\__codedoc_typeset_TF:`
`\__codedoc_typeset_aux:n`

Used by `\__codedoc_macro_single:nNN` and in the `function` environment to typeset conditionals and auxiliary functions.

```
\cs_new_protected:Npn \__codedoc_typeset_exp:
  {
    \cs_if_exist:NTF \Codedoc@expstar
      { \hyperlink { expstar } }
      { \mbox }
    {$\star$}
  }
\cs_new_protected:Npn \__codedoc_typeset_rexp:
  {
    \cs_if_exist:NTF \Codedoc@rexpstar
      { \hyperlink { rexpstar } }
      { \mbox }
    { \ding { 73 } } % hollow star
  }
\cs_new_protected:Npn \__codedoc_typeset_TF:
  {
    \cs_if_exist:NTF \Codedoc@explTF
      { \hyperlink { explTF } }
      { \mbox }
      {
        \color{black}
        \itshape TF
        \makebox[0pt][r]
          {
            \cs_if_exist:NT \Codedoc@explTF { \color{red} }
            \underline { \phantom{\itshape TF} \kern-0.1em }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_typeset_aux:n #1
```

```
  {
    { \color[gray]{0.5} #1 }
  }
```

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

`\__codedoc_get_hyper_target:nN`
`\__codedoc_get_hyper_target:oN`
`\__codedoc_get_hyper_target:eN`

Create a hyperref anchor from a macro name `#1` and stores it in the token list variable `#2`. For instance, `\prg_replicate:nn` gives `doc/function//prg/replicate:nn`.

```
\cs_new_protected:Npn \__codedoc_get_hyper_target:nN #1#2
  {
    \__kernel_tl_set:Ne #2 { \tl_to_str:n {#1} }
    \tl_replace_all:NVn #2 \c_underscore_str { / }
    \tl_remove_all:NV   #2 \c_backslash_str
    \tl_put_left:Nn #2 { doc/function// }
  }
\cs_generate_variant:Nn \__codedoc_get_hyper_target:nN { o , e }
```

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

`\__codedoc_names_get_seq:nN`

The argument `#1` (argument of a `function` or `macro` environment) has catcodes 10 (space), 12 (other) and 13 (active). Sanitize catcodes. If the `verb` option was used, output a one-item sequence. Otherwise, remove any "%" character at the beginning of a line. Remove tabs and newlines. Finally, convert `_@@` and `@@` to `__`⟨*module name*⟩ (if it is non-empty). At this point, `\l__codedoc_tmpa_tl` contains a comma-delimited list of names, where `@` and `_` have category code letter. Turn it to a string, parse it as a comma-delimited list (in particular this removes spaces), and output a sequence of function/macro names.

```
\cs_new_protected:Npn \__codedoc_names_get_seq:nN #1#2
  {
    \__kernel_tl_set:Ne \l__codedoc_tmpa_tl { \tl_to_str:n {#1} }
    \bool_if:NTF \l__codedoc_names_verb_bool
      {
        \seq_clear:N #2
        \seq_put_right:NV #2 \l__codedoc_tmpa_tl
      }
      {
        \tl_remove_all:Ne \l__codedoc_tmpa_tl
          { \iow_char:N \^^M \c_percent_str }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \tl_to_str:n { ^ ^ A } }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \iow_char:N \^^I }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \iow_char:N \^^M }
        \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
```

```
        \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
        \exp_args:NNe \seq_set_from_clist:Nn #2
          { \tl_to_str:N \l__codedoc_tmpa_tl }
      }
  }
```

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

`\__codedoc_names_parse:`
`\__codedoc_names_parse_one:n`

The goal is to group variants together. We populate `\l__codedoc_names_block_tl` with local sequence variable named with `\__codedoc_lseq_name:n` after the base forms. When encountering a new base form, set the corresponding local sequence to hold the ⟨*base name*⟩ (stripped of the signature) and add the local sequence to the list `\l__codedoc_names_block_tl`. In all cases append the signature to the local sequence, which thus takes the form ⟨*base name*⟩, ⟨*signature*$_1$⟩, ⟨*signature*$_2$⟩ and so on. If the original function had no signature (no colon) then use `\scan_stop:` as the signature (there can be no variant). We special case commands `#1` starting with `\::`, namely weird functions named `\::N` and the like.

```
\cs_new_protected:Npn \__codedoc_names_parse:
  {
    \tl_clear:N \l__codedoc_names_block_tl
    \seq_map_function:NN
      \l__codedoc_names_seq
      \__codedoc_names_parse_one:n
  }
\cs_new_protected:Npn \__codedoc_names_parse_one:n #1
  {
    \__codedoc_split_function_do:nn {#1}
      { \__codedoc_names_parse_one_aux:nnNn }
    {#1}
  }
\cs_new_protected:Npn \__codedoc_names_parse_one_aux:nnNn #1#2#3#4
  {
    \bool_if:NTF #3
      {
        \tl_if_head_eq_charcode:nNTF {#2} :
          { \__codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: } }
          {
            \exp_args:Ne \__codedoc_names_parse_aux:nnn
              { \__codedoc_base_form_aux:nnN {#1} {#2} #3 }
              {#1} {#2}
          }
```

```
      }
      {
        \bool_if:NT \l__codedoc_macro_TF_bool
          { \msg_error:nne { l3doc } { no-signature-TF } {#4} }
        \__codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: }
      }
  }
\cs_new_protected:Npn \__codedoc_names_parse_aux:nnn #1
  { \exp_args:Nc \__codedoc_names_parse_aux:Nnn { \__codedoc_lseq_name:n {#1} } }
\cs_new_protected:Npn \__codedoc_names_parse_aux:Nnn #1#2#3
  {
    \tl_if_in:NnF \l__codedoc_names_block_tl {#1}
      {
        \tl_put_right:Nn \l__codedoc_names_block_tl {#1}
        \seq_clear_new:N #1
        \seq_put_right:Nn #1 {#2}
      }
    \seq_put_right:Nn #1 {#3}
  }
```

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

`\__codedoc_names_typeset:`
`\__codedoc_names_typeset_auxi:n`

This code is in particular used when typesetting function names in a `function` environment. The mapping over `\l__codedoc_names_block_tl` cannot use `\tl_map_inline:Nn` because the code following `\\` would not be expandable, thus breaking `\bottomrule`.

Call `\__codedoc_names_typeset_auxi:n` on each local sequence (which holds a set of variants). The first step is to pop the base form and change spaces to category other so that they get displayed eventually. Then store the variants in `\g__codedoc_variants_seq`, remove the first, which will be displayed more prominently, and reconstruct the actual name, passing it to `\__codedoc_names_typeset_auxii:n`.

```
\cs_new_protected:Npn \__codedoc_names_typeset:
  {
    \tl_map_function:NN \l__codedoc_names_block_tl
      \__codedoc_names_typeset_auxi:n
  }
\cs_new_protected:Npn \__codedoc_names_typeset_auxi:n #1
  {
    \seq_pop:NN #1 \l__codedoc_tmpa_tl
    \tl_gset_eq:NN \g__codedoc_base_name_tl \l__codedoc_tmpa_tl
```

```
    \tl_greplace_all:NnV \g__codedoc_base_name_tl
      { ~ } \c_catcode_other_space_tl
    \seq_get:NN #1 \l__codedoc_tmpa_tl
    \str_if_eq:VnTF \l__codedoc_tmpa_tl { \scan_stop: }
      {
        \seq_gclear:N \g__codedoc_variants_seq
        \__codedoc_names_typeset_auxii:e { \g__codedoc_base_name_tl }
      }
      {
        \seq_gset_eq:NN \g__codedoc_variants_seq #1
        \seq_gpop:NN \g__codedoc_variants_seq \l__codedoc_tmpb_tl
        \__codedoc_names_typeset_auxii:e
          { \g__codedoc_base_name_tl : \l__codedoc_tmpb_tl }
      }
  }
```

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

`\__codedoc_names_typeset_auxii:n`
`\__codedoc_names_typeset_auxii:e`

In case the option `pTF` was given, typeset predicates before the `TF` functions. In case the option `noTF` was given, typeset the non-`TF` function as well. Pass the relevant boolean in both cases to control whether to append `TF`.

```
\cs_new_protected:Npn \__codedoc_names_typeset_auxii:n #1
  {
    \bool_if:NT \l__codedoc_macro_pTF_bool
      {
        \__codedoc_names_typeset_block:eN
          { \__codedoc_predicate_from_base:n {#1} }
          \c_false_bool
      }
    \bool_if:NT \l__codedoc_macro_noTF_bool
      { \__codedoc_names_typeset_block:nN {#1} \c_false_bool }
    \__codedoc_names_typeset_block:nN {#1} \l__codedoc_macro_TF_bool
  }
\cs_generate_variant:Nn \__codedoc_names_typeset_auxii:n { e }
```

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

`\__codedoc_names_typeset_block:nN`
`\__codedoc_names_typeset_block:eN`

Names in `function` and `macro` environments are typeset differently. To distinguish the two note that `\l__codedoc_nested_macro_int` is at least one when in an `macro` environment (we assume `function` is not nested inside it). A block is a function with all its variants.

```
\cs_new_protected:Npn \__codedoc_names_typeset_block:nN
```

```
  {
    \int_compare:nNnTF \l__codedoc_nested_macro_int = 0
      { \__codedoc_typeset_function_block:nN }
      { \__codedoc_macro_typeset_block:nN }
  }
\cs_generate_variant:Nn \__codedoc_names_typeset_block:nN { e }
```

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

`\__codedoc_if_macro_internal_p:n`
`\__codedoc_if_macro_internal:nTF`
`\__codedoc_if_macro_internal_aux:w`

Determines whether the given macro should be considered internal or public. If an option such as `int` was given then the answer is `\l__codedoc_macro_internal_bool`, otherwise check for whether the macro name contains `__`.

```
\prg_new_conditional:Npnn \__codedoc_if_macro_internal:n #1 { p , T , F , TF }
  {
    \bool_if:NTF \l__codedoc_macro_internal_bool
      { \prg_return_true: }
      {
        \tl_if_empty:eTF
          {
            \exp_after:wN \__codedoc_if_macro_internal_aux:w
            \tl_to_str:n { #1 ~ __ }
          }
          { \prg_return_false: } { \prg_return_true: }
      }
  }
\exp_last_unbraced:NNNNo
  \cs_new:Npn \__codedoc_if_macro_internal_aux:w #1 { \tl_to_str:n { __ } } { }
```

(*End of definition for* `\__codedoc_if_macro_internal:nTF` *and* `\__codedoc_if_macro_internal_aux:w`.)

`\__codedoc_names_block_base_map:N`

The `\l__codedoc_names_block_tl` contains sequence variables corresponding to different base functions and their variants. For each such sequence, put the first and second items in `\l__codedoc_tmpa_tl` and `\l__codedoc_tmpb_tl` and build the base function's name.

```
\cs_new_protected:Npn \__codedoc_names_block_base_map:N #1
  {
    \tl_map_inline:Nn \l__codedoc_names_block_tl
      {
        \group_begin:
          \seq_set_eq:NN \l__codedoc_tmpa_seq ##1
          \seq_pop:NN \l__codedoc_tmpa_seq \l__codedoc_tmpa_tl
          \seq_get:NN \l__codedoc_tmpa_seq \l__codedoc_tmpb_tl
```

```
          \exp_args:NNe
        \group_end:
        #1
          {
            \l__codedoc_tmpa_tl
            \str_if_eq:VnF \l__codedoc_tmpb_tl { \scan_stop: }
              { : \l__codedoc_tmpb_tl }
            \bool_if:NT \l__codedoc_macro_TF_bool { TF }
          }
      }
  }
```

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

5.9.2 The `function` environment

```
\keys_define:nn { l3doc/function }
  {
    TF .value_forbidden:n = true ,
    TF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
      } ,
    EXP .value_forbidden:n = true ,
    EXP .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    rEXP .value_forbidden:n = true ,
    rEXP .code:n =
      {
        \bool_set_false:N \l__codedoc_macro_EXP_bool
        \bool_set_true:N \l__codedoc_macro_rEXP_bool
      } ,
    pTF .value_forbidden:n = true ,
    pTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_pTF_bool
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
```

```
    noTF .value_forbidden:n = true ,
    noTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_noTF_bool
        \bool_set_true:N \l__codedoc_macro_TF_bool
      } ,
    added .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_added_tl {#1} },
    updated .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_updated_tl {#1} } ,
    deprecated .bool_set:N = \l__codedoc_macro_deprecated_bool ,
    no-user-doc .bool_set:N = \l__codedoc_macro_nodoc_bool ,
    tested .code:n = { } ,
    label .code:n =
      {
        \clist_set:Nn \l__codedoc_function_label_clist {#1}
        \bool_set_true:N \l__codedoc_no_label_bool
      } ,
    verb .value_forbidden:n = true ,
    verb .bool_set:N = \l__codedoc_names_verb_bool ,
    module .tl_set:N = \l__codedoc_override_module_tl ,
  }
```

`\__codedoc_date_set:Nn`
`\__codedoc_date_set_past:Nn`

Normalize the date into the format `YYYY-MM-DD`; more precisely month and day are allowed to be single digits. The `\__codedoc_date_set_past:Nn` function only allows dates in the past (or same day).

```
\cs_new_protected:Npn \__codedoc_date_set:Nn #1#2
  {
    \tl_set:Nn #1 {#2}
    \regex_replace_once:nnNF
      { \A(\d\d\d\d)[-/](\d\d?)[-/](\d\d?)\Z } { \1-\2-\3 } #1
      {
        \msg_error:nnn { l3doc } { date-format } {#2}
        \tl_set:Nn #1 { 1970-01-01 }
      }
  }
\cs_new_protected:Npn \__codedoc_date_set_past:Nn #1#2
  {
    \__codedoc_date_set:Nn #1 {#2}
    \exp_args:No \__codedoc_date_compare:nNnT
      {#1} > { \c_sys_year_int - \c_sys_month_int - \c_sys_day_int }
      {
        \msg_error:nnee { l3doc } { future-date }
          { \tl_to_str:N \l__codedoc_macro_argument_tl }
```

```
            {#1}
        }
    }
```

(*End of definition for* `\__codedoc_date_set:Nn` *and* `\__codedoc_date_set_past:Nn`.)

`\__codedoc_function:nnw`

`#1` : Key–value list.

`#2` : Comma-separated list of functions; input has already been sanitised by catcode changes before reading the argument.

`\__codedoc_function_end:`

Make sure any paragraph is finished, and similar safe practices at the beginning of an environment which will typeset material. Initialize some variables. Parse the key–value list. Clean up the list of functions, then go through them to extract some data. After this, typeset the function names in the coffin `\l__codedoc_functions_coffin` and measure it to know if it fits in the margin. Finally, start a vertical coffin for the main part of the environment. This coffin stops when the environment ends, then all the pieces are assembled into a single coffin, which is typeset.

```
\cs_new_protected:Npn \__codedoc_function:nnw #1#2
  {
    \__codedoc_function_typeset_start:
    \__codedoc_function_init:
    \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
    \keys_set:nn { l3doc/function } {#1}
    \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
    \__codedoc_names_parse:
    \__codedoc_function_typeset:
    \__codedoc_function_reset:
    \__codedoc_function_descr_start:w
  }
\cs_new_protected:Npn \__codedoc_function_end:
  {
    \__codedoc_function_descr_stop:
    \__codedoc_function_assemble:
    \__codedoc_function_typeset_stop:
  }
```

(*End of definition for* `\__codedoc_function:nnw` *and* `\__codedoc_function_end:`.)

`\__codedoc_function_typeset_start:`
`\__codedoc_function_typeset_stop:`

At the start of the `function` environment, before performing any assignment, close the last paragraph, and set up the typesetting scene. Further code typesets a coffin, so we end the paragraph and allow a page break.

```
\cs_new_protected:Npn \__codedoc_function_typeset_start:
```

```
  {
    \par \bigskip \noindent
  }
\cs_new_protected:Npn \__codedoc_function_typeset_stop:
  {
    \par
    \dim_set:Nn \prevdepth { \box_dp:N \l__codedoc_descr_coffin }
    \allowbreak
  }
```

(*End of definition for* `\__codedoc_function_typeset_start:` *and* `\__codedoc_function_typeset_stop:`.)

`\__codedoc_function_init:` Complain if `function` environments are nested. Clear various variables.

```
\cs_new_protected:Npn \__codedoc_function_init:
  {
    \box_if_empty:NF \g__codedoc_syntax_box
      { \msg_error:nn { l3doc } { syntax-nested-function } }
    \coffin_clear:N \l__codedoc_descr_coffin
    \box_gclear:N \g__codedoc_syntax_box
    \coffin_clear:N \l__codedoc_syntax_coffin
    \coffin_clear:N \l__codedoc_functions_coffin
    \bool_set_false:N \l__codedoc_macro_TF_bool
    \bool_set_false:N \l__codedoc_macro_pTF_bool
    \bool_set_false:N \l__codedoc_macro_noTF_bool
    \bool_set_false:N \l__codedoc_macro_EXP_bool
    \bool_set_false:N \l__codedoc_macro_rEXP_bool
    \bool_set_false:N \l__codedoc_no_label_bool
    \bool_set_false:N \l__codedoc_names_verb_bool
    \bool_set_true:N \l__codedoc_in_function_bool
    \clist_clear:N \l__codedoc_function_label_clist
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
    \char_set_active_eq:NN \< \__codedoc_shorthand_meta:
    \char_set_catcode_active:N \<
  }
```

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

`\__codedoc_shorthand_meta:` `\__codedoc_shorthand_meta:w` Allow `<...>` to be used as markup for `\meta{...}`.

```
\cs_new_protected:Npn \__codedoc_shorthand_meta:
  { \mode_if_math:TF { < } { \__codedoc_shorthand_meta:w } }
\cs_new_protected_nopar:Npn \__codedoc_shorthand_meta:w #1 > { \meta {#1} }
```

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

__codedoc_function_reset: Clear some variables.

```
\cs_new_protected:Npn \__codedoc_function_reset:
  {
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
  }
```

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

__codedoc_function_typeset: Typeset in the coffin `\l__codedoc_functions_coffin` the functions listed in `\l__codedoc_names_block_tl` and the relevant dates, then set `\l__codedoc_long_name_bool` to be `true` if this coffin is larger than the available width in the margin. The function `\__codedoc_typeset_functions:` is quite involved hence given later.

```
\cs_new_protected:Npn \__codedoc_function_typeset:
  {
    \dim_zero:N \l__codedoc_trial_width_dim
    \hcoffin_set:Nn \l__codedoc_functions_coffin { \__codedoc_typeset_functions: }
    \dim_set:Nn \l__codedoc_trial_width_dim
      { \box_wd:N \l__codedoc_functions_coffin }
    \bool_set:Nn \l__codedoc_long_name_bool
      { \dim_compare_p:nNn \l__codedoc_trial_width_dim > \marginparwidth }
  }
```

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

__codedoc_function_descr_start:w
__codedoc_function_descr_stop: The last step in `\__codedoc_function:nnw` (the beginning of a `function` environment) is to open a coffin which will capture the description of the function, namely the body of the `function` environment. This is closed by `\__codedoc_function_end:` (the end of a `function` environment).

```
\cs_new_protected:Npn \__codedoc_function_descr_start:w
  {
    \vcoffin_set:Nnw \l__codedoc_descr_coffin { \textwidth }
      \noindent \ignorespaces
  }
\cs_new_protected:Npn \__codedoc_function_descr_stop:
  { \vcoffin_set_end: }
```

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

__codedoc_function_assemble: The box `\g__codedoc_syntax_box` contains the contents of the syntax environment if it was used. Now that we have all the pieces, join together the syntax coffin, the names coffin, and the description coffin. The relative positions depend on whether the names coffin fits in the margin. Then typeset the combination.

```
\cs_new_protected:Npn \__codedoc_function_assemble:
  {
    \hcoffin_set:Nn  \l__codedoc_syntax_coffin
      { \box_use_drop:N \g__codedoc_syntax_box }
    \bool_if:NTF \l__codedoc_long_name_bool
      {
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {hc} {vc}
          \l__codedoc_syntax_coffin {l} {T}
          {0pt} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {t}
          \l__codedoc_functions_coffin  {r} {t}
          {-\marginparsep} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {b}
          \l__codedoc_descr_coffin  {l} {t}
          {0.75\marginparwidth + \marginparsep} {-\medskipamount}
        \coffin_typeset:Nnnnn \l__codedoc_output_coffin
          {\l__codedoc_descr_coffin-l} {\l__codedoc_descr_coffin-t}
          {0pt} {0pt}
      }
      {
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {hc} {vc}
          \l__codedoc_syntax_coffin {l} {t}
          {0pt} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {b}
          \l__codedoc_descr_coffin  {l} {t}
          {0pt} {-\medskipamount}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {t}
          \l__codedoc_functions_coffin  {r} {t}
          {-\marginparsep} {0pt}
        \coffin_typeset:Nnnnn \l__codedoc_output_coffin
          {\l__codedoc_syntax_coffin-l} {\l__codedoc_syntax_coffin-T}
          {0pt} {0pt}
      }
  }
```

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

__codedoc_typeset_functions:

This function builds the `\l__codedoc_functions_coffin` by typesetting the function names (with variants) and the relevant dates in a `tabular` environment. The use of rules `\toprule`, `\midrule` and `\bottomrule` requires whatever lies between the last `\\` and the rule to be expandable, making our lives a bit complicated.

```
\cs_new_protected:Npn \__codedoc_typeset_functions:
  {
    \small\ttfamily
    \__codedoc_target:
    \Hy@MakeCurrentHref { HD. \int_use:N \c@HD@hypercount }
    \begin{tabular} [t] { @{} l @{} >{\hspace{\tabcolsep}} r @{} }
      \toprule
      \__codedoc_function_extra_labels:
      \__codedoc_names_typeset:
      \__codedoc_typeset_dates:
      \bottomrule
    \end{tabular}
    \normalfont\normalsize
  }
```

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

__codedoc_typeset_function_block:nN
__codedoc_typeset_function_block:eN
__codedoc_function_index:n
__codedoc_function_index:e

`#1` is a csname, `#2` a boolean indicating whether to add `TF` or not.

```
\cs_new_protected:Npn \__codedoc_typeset_function_block:nN #1#2
  {
    \__codedoc_function_index:e
      { #1 \bool_if:NT #2 { \tl_to_str:n {TF} } }
    \__codedoc_function_label:eN {#1} #2
    #1
    \bool_if:NT #2 { \__codedoc_typeset_TF: }
    \__codedoc_typeset_expandability:
    \seq_if_empty:NF \g__codedoc_variants_seq
      { \__codedoc_typeset_variant_list:nN {#1} #2 }
    \\
  }
\cs_generate_variant:Nn \__codedoc_typeset_function_block:nN { e }
\cs_new_protected:Npn \__codedoc_function_index:n #1
  {
    \seq_gput_right:Nn \g_doc_functions_seq {#1}
    \__codedoc_special_index:nn {#1} { usage }
  }
\cs_generate_variant:Nn \__codedoc_function_index:n { e }
```

```
\cs_new_protected:Npn \__codedoc_typeset_expandability:
  {
    &
    \bool_if:NT \l__codedoc_macro_EXP_bool  { \__codedoc_typeset_exp: }
    \bool_if:NT \l__codedoc_macro_rEXP_bool { \__codedoc_typeset_rexp: }
  }
```

`#1` is the function, `#2` whether to add `TF`.

```
\cs_new_protected:Npn \__codedoc_typeset_variant_list:nN #1#2
  {
    \\
    \__codedoc_typeset_aux:n { \__codedoc_get_function_name:n {#1} }
    :
    \int_compare:nTF { \seq_count:N \g__codedoc_variants_seq == 1 }
      { \seq_use:Nn \g__codedoc_variants_seq { } }
      {
        \hbox_set:Nn \l_tmpa_box
          { \seq_use:Nn \g__codedoc_variants_seq { \textrm| \nolinebreak[2] } }
        \textrm(
```

Set long variant lists in a parbox, short lists set natural length.

```
        \dim_compare:nNnTF { \box_wd:N \l_tmpa_box } > { .4\columnwidth }
          {
            \parbox[t]{.4\columnwidth}
              {
                \raggedright
                \hbox_unpack_drop:N \l_tmpa_box
                \textrm)
                \bool_if:NT #2 { \__codedoc_typeset_TF: }
              }
          }
          {
            \hbox_unpack_drop:N \l_tmpa_box
            \textrm)
            \bool_if:NT #2 { \__codedoc_typeset_TF: }
          }
      }
    \__codedoc_typeset_expandability:
  }
```

`#1` is the function name, `#2` whether to add `TF`.

```
\cs_new_protected:Npn \__codedoc_function_extra_labels:
  {
    \bool_if:NT \l__codedoc_no_label_bool
```

```
      {
        \clist_map_inline:Nn \l__codedoc_function_label_clist
          {
            \__codedoc_get_hyper_target:oN { \token_to_str:N ##1 }
              \l__codedoc_tmpa_tl
            \exp_args:No \label { \l__codedoc_tmpa_tl }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_function_label:nN #1#2
  {
    \bool_if:NF \l__codedoc_no_label_bool
      {
        \__codedoc_get_hyper_target:eN
          {
            \exp_not:n {#1}
            \bool_if:NT #2 { \tl_to_str:n {TF} }
          }
          \l__codedoc_tmpa_tl
        \exp_args:No \label { \l__codedoc_tmpa_tl }
      }
  }
\cs_generate_variant:Nn \__codedoc_function_label:nN { e }
```

(*End of definition for* `\__codedoc_typeset_function_block:nN` *and* `\__codedoc_function_index:n`.)

`\__codedoc_typeset_dates:` To display metadata for when functions are added/modified. This function must be expandable since it produces rules for use in alignments.

```
\cs_new:Npn \__codedoc_typeset_dates:
  {
    \bool_lazy_and:nnF
      { \tl_if_empty_p:N \l__codedoc_date_added_tl }
      { \tl_if_empty_p:N \l__codedoc_date_updated_tl }
      { \midrule }
    \tl_if_empty:NF \l__codedoc_date_added_tl
      {
        \multicolumn { 2 } { @{} r @{} }
          { \scriptsize New: \, \l__codedoc_date_added_tl } \\
      }

    \tl_if_empty:NF \l__codedoc_date_updated_tl
      {
        \multicolumn { 2 } { @{} r @{} }
```

```
          { \scriptsize Updated: \, \l__codedoc_date_updated_tl } \\
      }
  }
```

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

`\__codedoc_syntax:w` Implement the `syntax` environment.

`\__codedoc_syntax_end:`

```
\dim_new:N \l__codedoc_syntax_dim
\cs_new_protected:Npn \__codedoc_syntax:w
  {
    \box_if_empty:NF \g__codedoc_syntax_box
      { \msg_error:nn { l3doc } { multiple-syntax } }
    \dim_set:Nn \l__codedoc_syntax_dim
      {
        \textwidth
        \bool_if:NT \l__codedoc_long_name_bool
          { + 0.75 \marginparwidth - \l__codedoc_trial_width_dim }
      }
    \hbox_gset:Nw \g__codedoc_syntax_box
      \small \ttfamily
      \arrayrulecolor{white}
      \begin{tabular} { @{} l @{} }
        \toprule
        \begin{minipage}[t]{\l__codedoc_syntax_dim}
          \raggedright
          \obeyspaces
          \obeylines
  }
\cs_new_protected:Npn \__codedoc_syntax_end:
  {
        \end{minipage}
      \end{tabular}
      \arrayrulecolor{black}
    \hbox_gset_end:
    \bool_if:NF \l__codedoc_in_function_bool
      {
        \begin{quote}
          \mode_leave_vertical:
          \box_use_drop:N \g__codedoc_syntax_box
        \end{quote}
      }
  }
```

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

5.9.3 The `macro` environment

Keyval for the `macro` environment. TODO: provide document command for documenting keys.

```
\keys_define:nn { l3doc/macro }
  {
    aux .value_forbidden:n = true ,
    aux .code:n =
      {
        \msg_warning:nnnn { l3doc } { deprecated-option }
          { aux } { function/macro }
      } ,
    deprecated .bool_set:N = \l__codedoc_macro_deprecated_bool ,
    internal .value_forbidden:n = true ,
    internal .code:n =
      { \bool_set_true:N \l__codedoc_macro_internal_bool } ,
    int .value_forbidden:n = true ,
    int .code:n =
      { \bool_set_true:N \l__codedoc_macro_internal_bool } ,
    no-user-doc .bool_set:N = \l__codedoc_macro_nodoc_bool ,
    var .value_forbidden:n = true ,
    var .code:n =
      { \bool_set_true:N \l__codedoc_macro_var_bool } ,
    TF .value_forbidden:n = true ,
    TF .code:n =
      { \bool_set_true:N \l__codedoc_macro_TF_bool } ,
    pTF .value_forbidden:n = true ,
    pTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_pTF_bool
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    noTF .value_forbidden:n = true ,
    noTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_noTF_bool
      } ,
    EXP .value_forbidden:n = true ,
    EXP .code:n =
```

```
      {
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    rEXP .value_forbidden:n = true ,
    rEXP .code:n =
      {
        \bool_set_false:N \l__codedoc_macro_EXP_bool
        \bool_set_true:N \l__codedoc_macro_rEXP_bool
      } ,
    tested .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_tested_bool
      } ,
    added .code:n = {} , % TODO
    updated .code:n = {} , % TODO
    verb .bool_set:N = \l__codedoc_names_verb_bool ,
    module .tl_set:N = \l__codedoc_override_module_tl ,
    documented-as .tl_set:N = \l__codedoc_macro_documented_tl ,
    do-not-index .value_required:n = true ,
    do-not-index .tl_set:N = \l__codedoc_macro_do_not_index_tl ,
    % do-not-index .default:n = \q_no_value ,
  }
```

`\__codedoc_macro:nnw` The arguments are a key–value list of ⟨*options*⟩ and a comma-list of ⟨*names*⟩, read verbatim by ltcmd. First initialize some variables before applying the ⟨*options*⟩, then parse the ⟨*names*⟩ to get a sequence of macro names, then apply `\__codedoc_macro_single:nNN` to each (this step is more subtle than `\seq_map_function:NN` because of `TF`/`pTF`/`noTF`). Finally typeset the macro names in the margin.

```
\cs_new_protected:Npn \__codedoc_macro:nnw #1#2
  {
    \__codedoc_macro_init:
    \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
    \keys_set:nn { l3doc/macro } {#1}
    \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
    \__codedoc_names_parse:
    \__codedoc_macro_exclude_index:
    \__codedoc_macro_save_names:
    \__codedoc_names_typeset:
    \__codedoc_macro_dump:
    \__codedoc_macro_reset:
  }
```

(*End of definition for* `\__codedoc_macro:nnw`.)

`\__codedoc_macro_init:` The booleans hold various key–value options, `\l__codedoc_nested_macro_int` counts the number of `macro` environments around the current point (is 0 outside).

```
\cs_new_protected:Npn \__codedoc_macro_init:
  {
    \int_incr:N \l__codedoc_nested_macro_int
    \bool_set_false:N \l__codedoc_macro_deprecated_bool
    \bool_set_false:N \l__codedoc_macro_internal_bool
    \bool_set_false:N \l__codedoc_macro_TF_bool
    \bool_set_false:N \l__codedoc_macro_pTF_bool
    \bool_set_false:N \l__codedoc_macro_noTF_bool
    \bool_set_false:N \l__codedoc_macro_EXP_bool
    \bool_set_false:N \l__codedoc_macro_rEXP_bool
    \bool_set_false:N \l__codedoc_macro_var_bool
    \bool_set_false:N \l__codedoc_macro_tested_bool
    \bool_set_false:N \l__codedoc_names_verb_bool
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
    \tl_clear:N \l__codedoc_macro_documented_tl
    \cs_set_eq:NN \testfile \__codedoc_print_testfile:n
    \box_clear:N \l__codedoc_macro_index_box
    \vbox_set:Nn \l__codedoc_macro_box
      {
        \hbox:n
          {
            \strut
            \int_compare:nNnT \l__codedoc_macro_int = 0 { \__codedoc_target: }
          }
        \vskip \int_eval:n { \l__codedoc_macro_int - 1 } \baselineskip
      }
  }
```

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

`\__codedoc_macro_reset:` We ensure that `\cs` commands nested inside a macro whose module is imposed are not affected.

```
\cs_new_protected:Npn \__codedoc_macro_reset:
  {
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
  }
```

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

__codedoc_macro_save_names:

The list of names defined in a set of `macro` environments is eventually used to display on which page they are documented. If the `documented-as` key is given, use that, otherwise find names in `\l__codedoc_names_block_tl`.

```
\cs_new_protected:Npn \__codedoc_macro_save_names:
  {
    \tl_if_empty:NTF \l__codedoc_macro_documented_tl
      { \__codedoc_names_block_base_map:N \__codedoc_macro_save_names_aux:n }
      {
        \seq_gput_right:Ne \g__codedoc_nested_names_seq
          { \tl_to_str:N \l__codedoc_macro_documented_tl }
      }
  }
\cs_new_protected:Npn \__codedoc_macro_save_names_aux:n #1
  { \seq_gput_right:Nn \g__codedoc_nested_names_seq {#1} }
```

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

__codedoc_macro_exclude_index:

Some control sequences in a `macrocode` environment shouldn't be indexed, for different reasons. This macro parses the argument of the `do-not-index` option and locally removes the given macros from the index.

The optional argument to `macro` is not scanned with verbatim catcodes, so we use `\tl_set_rescan:NnV` to rescan the commands with the same catcodes as `\DoNotIndex`. The scanned token list contains spaces after control sequences, which are not there when `\DoNotIndex` is used. Since `\seq_set_from_clist:Nn` removes spaces around the items, we can abuse that and `\seq_use:Nn` to normalise each item. After that `\DoNotIndex` can do its thing.

```
\cs_new_protected:Npn \__codedoc_macro_exclude_index:
  {
    \tl_if_empty:NF \l__codedoc_macro_do_not_index_tl
      {
        \tl_set_rescan:NnV \l__codedoc_macro_do_not_index_tl
          { \MakePrivateLetters \catcode`\\12 }
          \l__codedoc_macro_do_not_index_tl
        \exp_args:NNV \seq_set_from_clist:Nn
          \l__codedoc_tmpa_seq \l__codedoc_macro_do_not_index_tl
        \__kernel_tl_set:Ne \l__codedoc_macro_do_not_index_tl
          { \seq_use:Nn \l__codedoc_tmpa_seq { , } }
        \exp_args:NV \DoNotIndex \l__codedoc_macro_do_not_index_tl
      }
  }
```

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

`\__codedoc_macro_dump:` This calls `\makelabel{}`

```
\cs_new_protected:Npn \__codedoc_macro_dump:
  {
    \topsep\MacroTopsep
    \trivlist
    \cs_set:Npn \makelabel ##1
      {
        \llap
          {
            \hbox_unpack_drop:N \l__codedoc_macro_index_box
            \vtop to \baselineskip
              {
                \vbox_unpack_drop:N \l__codedoc_macro_box
                \vss
              }
          }
      }
    \item [ ]
  }
```

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

`\__codedoc_macro_typeset_block:nN` Used to typeset a macro and its variants. `#1` is the macro name, `#2` is a boolean controlling whether to add `TF`.

```
\cs_new_protected:Npn \__codedoc_macro_typeset_block:nN #1#2
  {
    \__codedoc_macro_single:nNN {#1} \c_true_bool #2
    \seq_if_empty:NF \g__codedoc_variants_seq
      {
        \__codedoc_macro_typeset_variant_list:eN
          { \__codedoc_get_function_name:n {#1} } #2
      }
  }
\cs_new_protected:Npn \__codedoc_macro_typeset_variant_list:nN #1#2
  {
    \seq_map_inline:Nn \g__codedoc_variants_seq
      { \__codedoc_macro_single:nNN { #1 : ##1 } \c_false_bool #2 }
  }
\cs_generate_variant:Nn \__codedoc_macro_typeset_variant_list:nN { e }
```

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

__codedoc_macro_single:nNN

The arguments are #1 a macro name (without TF), #2 a boolean determining whether or not to index, and #3 whether or not to add TF. Let's start to mess around with doc's macro environment. See doc.dtx for a full explanation of the original environment. It's rather *enthusiastically* commented.

#1 : Macro/function/whatever name; input has already been sanitised.

The assignments to \saved@macroname and \saved@indexname are used by doc's \changes mechanism.

```
\cs_new_protected:Npn \__codedoc_macro_single:nNN #1#2#3
  {
    \tl_set:Nn \saved@macroname {#1}
    \__codedoc_macro_typeset_one:nN {#1} #3
    \bool_if:NT #3 { \DoNotIndex {#1} }
    \exp_args:Ne \__codedoc_macro_index:nN
      { #1 \bool_if:NT #3 { \tl_to_str:n { TF } } }
      #2
  }
\cs_new_protected:Npn \__codedoc_macro_index:nN #1#2
  {
    \DoNotIndex {#1}
    \bool_if:NT #2
      {
        \bool_lazy_any:nF
          {
            { \__codedoc_if_macro_internal_p:n {#1} }
            { \l__codedoc_macro_deprecated_bool }
            { \l__codedoc_macro_nodoc_bool }
          }
          { \seq_gput_right:Nn \g_doc_macros_seq {#1} }
        \hbox_set:Nw \l__codedoc_macro_index_box
          \hbox_unpack_drop:N \l__codedoc_macro_index_box
          \int_gincr:N \c@CodelineNo
          \__codedoc_special_index:nn {#1} { main }
          \int_gdecr:N \c@CodelineNo
        \exp_args:NNNo \hbox_set_end:
          \tl_set:Nn \saved@indexname { \l__codedoc_index_key_tl }
      }
  }
```

(*End of definition for* __codedoc_macro_single:nNN.)

__codedoc_macro_typeset_one:nN

For a long time, l3doc collected the macro names as labels in the first items of nested \trivlist, but these were not closed properly with \endtrivlist. Also,

it interacted in surprising ways with hyperref targets. Now, we collect typeset macro names by hand in the box `\l__codedoc_macro_box`. The fixed-size space `\MacroFont\ ` could be replaced by an customizable horizontal space; it is important for it to be the same for all macros. `#1` is the macro name, `#2` whether to add `TF`.

```
\cs_new_protected:Npn \__codedoc_macro_typeset_one:nN #1#2
  {
    \vbox_set:Nn \l__codedoc_macro_box
      {
        \vbox_unpack_drop:N \l__codedoc_macro_box
        \hbox { \llap { \__codedoc_print_macroname:nN {#1} #2
            \MacroFont \
        } }
      }
    \int_incr:N \l__codedoc_macro_int
  }
```

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

`\__codedoc_print_macroname:nN` In the name, spaces are replaced by other spaces to ensure they get displayed in case there are any.

```
\cs_new_protected:Npn \__codedoc_print_macroname:nN #1#2
  {
    \strut
    \__codedoc_get_hyper_target:eN
      {
        \exp_not:n {#1}
        \bool_if:NT #2 { \tl_to_str:n {TF} }
      }
      \l__codedoc_tmpa_tl
    \cs_if_exist:cTF { r@ \l__codedoc_tmpa_tl }
      { \exp_last_unbraced:NNo \hyperref [ \l__codedoc_tmpa_tl ] }
      { \use:n }
      {
        \int_compare:nTF { \str_count:n {#1} <= 28 }
          { \MacroFont } { \MacroLongFont }
        \tl_set:Nn \l__codedoc_tmpa_tl {#1}
        \tl_replace_all:NnV \l__codedoc_tmpa_tl
          { ~ } \c_catcode_other_space_tl
        \__codedoc_macroname_prefix:o \l__codedoc_tmpa_tl
        \__codedoc_macroname_suffix:N #2
      }
```

```
  }
\cs_new_protected:Npn \__codedoc_macroname_prefix:n #1
  {
    \__codedoc_if_macro_internal:nTF {#1}
      { \__codedoc_typeset_aux:n {#1} } {#1}
  }
\cs_generate_variant:Nn \__codedoc_macroname_prefix:n { o }
\cs_new_protected:Npn \__codedoc_macroname_suffix:N #1
  { \bool_if:NTF #1 { \__codedoc_typeset_TF: } { } }
```

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

`\MacroLongFont`

```
\providecommand \MacroLongFont
  {
    \fontfamily{lmtt}\fontseries{lc}\small
  }
```

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

`\__codedoc_print_testfile:n` Used to show that a macro has a test, somewhere.

`\__codedoc_print_testfile_aux:n`

```
\cs_new_protected:Npn \__codedoc_print_testfile:n #1
  {
    \bool_set_true:N \l__codedoc_macro_tested_bool
    \tl_if_eq:nnF {#1} {*}
      {
        \seq_if_in:NnF \g__codedoc_testfiles_seq {#1}
          {
            \seq_gput_right:Nn \g__codedoc_testfiles_seq {#1}
            \par
            \__codedoc_print_testfile_aux:n {#1}
          }
      }
  }
\cs_new_protected:Npn \__codedoc_print_testfile_aux:n #1
  {
    \footnotesize
    (
    \textit
      {
        The~ test~ suite~ for~ this~ command,~
        and~ others~ in~ this~ file,~ is~ \textsf{#1}
      }.
```

```
    )\par
  }
```

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

`\TestFiles`

```
\DeclareDocumentCommand \TestFiles {m}
  {
    \par
    \textit
      {
        The~ following~ test~ files~ are~
        used~ for~ this~ code:~ \textsf{#1}.
      }
    \par \ignorespaces
  }
```

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

`\UnitTested`

```
\DeclareDocumentCommand \UnitTested { } { \testfile* }
```

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

`\TestMissing`

```
\DeclareDocumentCommand \TestMissing { m }
  { \__codedoc_test_missing:n {#1} }
```

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

`\__codedoc_test_missing:n` Keys in `\g__codedoc_missing_tests_prop` are lists of macros given as arguments of one `macro` environment. Values are pairs of a file name and a comment about the missing tests.

```
\cs_new_protected:Npn \__codedoc_test_missing:n #1
  {
    \__codedoc_test_missing_aux:Nen
      \g__codedoc_missing_tests_prop
      { \seq_use:Nn \l__codedoc_names_seq { , } }
      { { \g_file_curr_name_str \c_space_tl (#1) } }
  }
\cs_new_protected:Npn \__codedoc_test_missing_aux:Nnn #1#2#3
  {
    \prop_get:NnNTF #1 {#2} \l__codedoc_tmpa_tl
```

```
      { \tl_put_right:Nn \l__codedoc_tmpa_tl { , #3 } }
      { \tl_set:Nn \l__codedoc_tmpa_tl {#3} }
    \prop_put:Nno #1 {#2} \l__codedoc_tmpa_tl
  }
\cs_generate_variant:Nn \__codedoc_test_missing_aux:Nnn { Ne }
```

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

`\__codedoc_macro_end:` It is too late for anyone to declare a test file for this macro, so we can check now whether the macro is tested. If the `macro` environment which is being ended is the outermost one, then wrap each macro in `\texttt` (with the addition of `TF` if relevant) and typeset two informations: that this ends the definition of some macros, and that they are documented on some page.

```
\cs_new_protected:Npn \__codedoc_macro_end:
  {
    \endtrivlist
    \__codedoc_macro_end_check_tested:
    \int_compare:nNnT \l__codedoc_nested_macro_int = 1
      { \__codedoc_macro_end_style:n { \__codedoc_print_end_definition: } }
  }
```

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

`\__codedoc_macro_end_check_tested:` If the `checktest` option was issued and the macro is not an auxiliary nor a variable (and it does not have a test), then add it to the sequence of non-tested macros.

```
\cs_new_protected:Npn \__codedoc_macro_end_check_tested:
  {
    \bool_lazy_all:nT
     {
       { \g__codedoc_checktest_bool }
       { ! \l__codedoc_macro_var_bool }
       { ! \l__codedoc_macro_tested_bool }
     }
     {
       \seq_set_filter:NNn \l__codedoc_tmpa_seq \l__codedoc_names_seq
         { ! \__codedoc_if_macro_internal_p:n {##1} }
       \seq_gput_right:Ne \g__codedoc_not_tested_seq
         {
           \seq_use:Nn \l__codedoc_tmpa_seq { , }
           \bool_if:NTF \l__codedoc_macro_pTF_bool {~(pTF)}
             { \bool_if:NT \l__codedoc_macro_TF_bool {~(TF)} }
         }
```

```
    }
  }
```

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

`\__codedoc_macro_end_style:n` Style for the extra information at the end of a top-level `macro` environment.

```
\cs_new_protected:Npn \__codedoc_macro_end_style:n #1
  {
    \nobreak \noindent
    { \footnotesize ( \emph{#1} ) \par }
  }
```

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

`\__codedoc_print_end_definition:`
`\__codedoc_macro_end_wrap_item:n`
`\__codedoc_print_documented:`

Surround each item by `\texttt`, replacing `_` by `\_` as well. Then list the macro names through `\seq_use:Nnnn`, unless there are too many. Finally, if the macro is neither auxiliary nor internal, add a link to where it is documented.

```
\cs_new_protected:Npn \__codedoc_macro_end_wrap_item:n #1
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#1}
    \tl_replace_all:NVn \l__codedoc_tmpa_tl
      \c_underscore_str { \_ }
    \texttt { \l__codedoc_tmpa_tl }
  }
\cs_new_protected:Npn \__codedoc_print_end_definition:
  {
    \seq_set_map:NNn \l__codedoc_tmpa_seq
      \g__codedoc_nested_names_seq
      { \__codedoc_macro_end_wrap_item:n {##1} }
    End~ of~ definition~ for~
    \int_compare:nTF { \seq_count:N \l__codedoc_tmpa_seq <= 3 }
      {
        \seq_use:Nnnn \l__codedoc_tmpa_seq
          { \,~and~ } { \,,~ } { \,,~and~ }
      }
      { \seq_item:Nn \l__codedoc_tmpa_seq {1}\,~and~others }
    \@.
    \__codedoc_print_documented:
  }
\cs_new_protected:Npn \__codedoc_print_documented:
  {
    \seq_gset_filter:NNn \g__codedoc_nested_names_seq
      \g__codedoc_nested_names_seq
```

```
      {
        ! \bool_lazy_any_p:n
          {
            { \__codedoc_if_macro_internal_p:n {##1} }
            { \l__codedoc_macro_deprecated_bool }
            { \l__codedoc_macro_nodoc_bool }
          }
      }
    \seq_if_empty:NF \g__codedoc_nested_names_seq
      {
        \int_set:Nn \l__codedoc_tmpa_int
          { \seq_count:N \g__codedoc_nested_names_seq }
        \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~This~} {~These~}
        \bool_if:NTF \l__codedoc_macro_var_bool {variable} {function}
        \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~is~} {s~are~}
        documented~on~page~
        \__codedoc_get_hyper_target:eN
          { \seq_item:Nn \g__codedoc_nested_names_seq { 1 } }
          \l__codedoc_tmpa_tl
        \exp_args:Ne \pageref { \l__codedoc_tmpa_tl } .
      }
    \seq_gclear:N \g__codedoc_nested_names_seq
  }
```

(*End of definition for* `\__codedoc_print_end_definition:`, `\__codedoc_macro_end_wrap_item:n`, *and* `\__codedoc_print_documented:`.)

5.9.4 Misc

`\DescribeOption` For describing package options: retained for consistency, but updated for doc v3.

```
\NewDocElement[idxtype = option, idxgroup = options]{Option}{optionenv}
```

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

Here are some definitions for additional markup that helps to structure your documentation.

`danger` (*env.*)
`ddanger` (*env.*)

```
\begin{[d]danger}
dangerous code
\end{[d]danger}
```

Provides a danger bend, as known from the TEXbook.

The actual character from the font `manfnt`:

```
\font \manual = manfnt \scan_stop:
\cs_gset:Npn \dbend { {\manual\char127} }
```

Defines the single danger bend. Use it whenever there is a feature in your package that might be tricky to use. FIXME: Has to be fixed when in combination with a macro-definition.

```
\newenvironment {danger}
  {
    \begin{trivlist}\item[]\noindent
    \begingroup\hangindent=2pc\hangafter=-2
    \cs_set:Npn \par{\endgraf\endgroup}
    \hbox to0pt{\hskip-\hangindent\dbend\hfill}\ignorespaces
  }
  {
    \par\end{trivlist}
  }
```

Use the double danger bend if there is something which could cause serious problems when used in a wrong way. Better the normal user does not know about such things.

```
\newenvironment {ddanger}
  {
    \begin{trivlist}\item[]\noindent
    \begingroup\hangindent=3.5pc\hangafter=-2
    \cs_set:Npn \par{\endgraf\endgroup}
    \hbox to0pt{\hskip-\hangindent\dbend\kern2pt\dbend\hfill}\ignorespaces
  }{
      \par\end{trivlist}
  }
```

5.9.5 NB and NOTE

These macros are intended for additional notes added to the source that are not typeset.

\NB `\NB{wspr}{this is what I think about this!}`

```
\bool_if:NTF \g__codedoc_show_notes_bool
  {
    \NewDocumentCommand\NB{mm}
      {
        (\emph{Note}\footnote{\ttfamily [#1]:~\detokenize{#2}})
      }
  }
```

```
  {
    \NewDocumentCommand\NB{mm}{}
  }
```

(*End of definition for* `\NB`. *This function is documented on page* 8.)

NOTE (*env.*)

```
\begin{NOTE}{wspr}
  this is what I #$%& think about this!
\end{NOTE}
```

```
\bool_if:NTF \g__codedoc_show_notes_bool
  {
    \NewDocumentEnvironment{NOTE}{m}
      {
        \par\noindent (\emph{Note}~[\texttt{#1}]:\par
        \verbatim
      }
      {
        \endverbatim
        \par\noindent \emph{Note~end})\par
      }
  }
  {
    \NewDocumentEnvironment{NOTE}{m}{\comment}{\endcomment}
  }
```

5.10 Footnote support

The environments `function` and `variable` are boxes and so looses footnotes. The following implements support. It relies currently on an internal from hyperref to get the correct targets.

```
\providecommand\Hy@footnote@currentHref{}
\prop_new:N\g__codedoc_fnmark_prop
\cs_new_protected:Npn \__codedoc_fn_store:
 {
   \prop_gput:Nee\g__codedoc_fnmark_prop
     {fn\int_use:N\c@footnote}{{\Hy@footnote@currentHref}{\int_use:N\c@footnote}}
 }
\cs_new_protected:Npn \__codedoc_fn_restore:n  #1
 {
   \prop_get:NnN \g__codedoc_fnmark_prop {fn#1}\l__codedoc_tmpa_tl
   \tl_gset:Ne\Hy@footnote@currentHref
```

```
      {\exp_last_unbraced:NV\use_i:nn \l__codedoc_tmpa_tl }
    \setcounter{footnote}{\exp_last_unbraced:NV\use_ii:nn \l__codedoc_tmpa_tl}
  }

\cs_generate_variant:Nn \hook_gput_next_code:nn {ne}
\cs_new_protected:Npn \__codedoc_fn_footnote:nn #1 #2
  {
    \footnotemark
    \__codedoc_fn_store:
    \hook_gput_next_code:ne {env/#1/after}
      {\exp_not:N\__codedoc_fn_restore:n{\int_use:N\c@footnote}{\exp_not:n{\footnotetext{#2}
\AddToHook{env/function/begin}{\def\footnote{\__codedoc_fn_footnote:nn{function}}}
\AddToHook{env/variable/begin}{\def\footnote{\__codedoc_fn_footnote:nn{variable}}}
```

5.11 Documenting templates

```
\newenvironment{TemplateInterfaceDescription}[1]
  {
    \subsection{The~object~type~`#1'}
    \begingroup
    \@beginparpenalty\@M
    \description
    \def\TemplateArgument##1##2{\item[Arg:~##1]##2\par}
    \def\TemplateSemantics
      {
        \enddescription\endgroup
        \subsubsection*{Semantics:}
      }
  }
  {
    \par\bigskip
  }
\newenvironment{TemplateDescription}[2]
  {
    \subsection{The~template~`#2'~(object~type~#1)}
    \subsubsection*{Attributes:}
    \begingroup
    \@beginparpenalty\@M
    \description
    \def\TemplateKey##1##2##3##4
      {
```

```
        \item[##1~(##2)]##3%
        \ifx\TemplateKey##4\TemplateKey\else
%         \hskip0ptplus3em\penalty-500\hskip 0pt plus 1filll Default:~##4%
          \hfill\penalty500\hbox{}\hfill Default:~##4%
          \nobreak\hskip-\parfillskip\hskip0pt\relax
        \fi
        \par
      }
    \def\TemplateSemantics
      {
        \enddescription\endgroup
        \subsubsection*{Semantics~\&~Comments:}
      }
  }
  { \par \bigskip }
\newenvironment{InstanceDescription}[4][xxxxxxxxxxxxxx]
  {
    \subsubsection{The~instance~`#3'~(template~#2/#4)}
    \subsubsection*{Attribute~values:}
    \begingroup
    \@beginparpenalty\@M
    \def\InstanceKey##1##2{\>\textbf{##1}\>##2\\}
    \def\InstanceSemantics{\endtabbing\endgroup
      \vskip-30pt\vskip0pt
      \subsubsection*{Layout~description~\&~Comments:}}
    \tabbing
    xxxx\=#1\=\kill
  }
  { \par \bigskip }
```

5.12 Inheriting doc

Code here is taken from doc, stripped of comments and translated into expl3 syntax. New features are added in various places.

\StopEventually
\MaybeStop
\Finale
\AlsoImplementation
\OnlyDescription
\g__codedoc_finale_tl

TODO: remove these four commands altogether, document that it is better to use the `documentation` and `implementation` environments.

```
\DeclareDocumentCommand \OnlyDescription { }
  { \bool_gset_false:N \g__codedoc_typeset_implementation_bool }
\DeclareDocumentCommand \AlsoImplementation { }
  { \bool_gset_true:N \g__codedoc_typeset_implementation_bool }
\DeclareDocumentCommand \StopEventually { m }
```

```
  {
    \bool_if:NTF \g__codedoc_typeset_implementation_bool
      {
        \@bsphack
        \tl_gset:Nn \g__codedoc_finale_tl { #1 \check@checksum }
        \init@checksum
        \@esphack
      }
      { #1 \endinput }
  }
```

We also need to support doc V3 `\MaybeStop` if it is around (which may not be the case).

```
\cs_if_exist:NT \MaybeStop
   { \RenewCommandCopy \MaybeStop \StopEventually }

\DeclareDocumentCommand \Finale { }
  { \tl_use:N \g__codedoc_finale_tl }
\tl_new:N \g__codedoc_finale_tl
```

(*End of definition for* `\StopEventually` *and others. These functions are documented on page* **??**.)

`\__codedoc_input:n` Inputting a file, with some setup: the module name should be empty before the first `<@@=`⟨*module*⟩`>` line in the file.

```
\cs_new_protected:Npn \__codedoc_input:n #1
  {
    \tl_gclear:N \g__codedoc_module_name_tl
    \MakePercentIgnore
    \input{#1}
    \MakePercentComment
  }
```

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

`\DocInput` Modified from doc to accept comma-list input (who has commas in filenames?).

```
\DeclareDocumentCommand \DocInput { m }
  {
    \clist_map_inline:nn {#1}
      {
        \clist_put_right:Nn \g_docinput_clist {##1}
        \__codedoc_input:n {##1}
      }
  }
```

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

`\DocInputAgain` Uses `\g_docinput_clist` to re-input whatever's already been `\DocInput`-ed until now. May be used multiple times.

```
\DeclareDocumentCommand \DocInputAgain { }
  { \clist_map_function:NN \g_docinput_clist \__codedoc_input:n }
```

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

`\DocInclude` More or less exactly the same as `\include`, but uses `\DocInput` on a `.dtx` file, not `\input` on a `.tex` file.

```
\NewDocumentCommand \DocInclude { m }
  {
    \relax\clearpage
    \docincludeaux
    \IfFileExists{#1.fdd}
      { \cs_set:Npn \currentfile{#1.fdd} }
      { \cs_set:Npn \currentfile{#1.dtx} }
    \int_compare:nNnTF \@auxout = \@partaux
      { \@latexerr{\string\include\space cannot~be~nested}\@eha }
      { \@docinclude {#1} }
  }

\cs_gset:Npn \@docinclude #1
  {
    \clearpage
    \immediate\write\@mainaux{\string\@input{#1.aux}}
    \@tempswatrue
    \if@partsw
      \@tempswafalse
      \cs_set:Npe \@tempb {#1}
      \clist_map_inline:Nn \@partlist
        {
          \if_meaning:w \@tempa \@tempb
            \@tempswatrue
          \fi:
        }
    \fi
    \if@tempswa
      \cs_set_eq:NN \@auxout                \@partaux
      \immediate\openout\@partaux #1.aux
      \immediate\write\@partaux{\relax}
      \cs_set_eq:NN \@ltxdoc@PrintIndex     \PrintIndex
```

```
      \cs_set_eq:NN \PrintIndex              \relax
      \cs_set_eq:NN \@ltxdoc@PrintChanges    \PrintChanges
      \cs_set_eq:NN \PrintChanges            \relax
      \cs_set_eq:NN \@ltxdoc@theglossary     \theglossary
      \cs_set_eq:NN \@ltxdoc@endtheglossary  \endtheglossary
      \part{\currentfile}
      {
        \cs_set_eq:NN \ttfamily\relax
        \cs_gset:Npe \filekey
          { \filekey,~ \thepart = { \ttfamily \currentfile } }
      }
      \DocInput{\currentfile}
      \cs_set_eq:NN \PrintIndex              \@ltxdoc@PrintIndex
      \cs_set_eq:NN \PrintChanges            \@ltxdoc@PrintChanges
      \cs_set_eq:NN \theglossary             \@ltxdoc@theglossary
      \cs_set_eq:NN \endtheglossary          \@ltxdoc@endtheglossary
      \clearpage
      \@writeckpt{#1}
      \immediate \closeout \@partaux
    \else
      \@nameuse{cp@#1}
    \fi
    \cs_set_eq:NN \@auxout \@mainaux
  }
```

Here, `MMMMI` (for page references) and `MMMMV` (for codeline references) are interpreted by `makeindex` as an uppercase Roman number pages, and should be large enough to avoid collisions with other uses of uppercase Roman number pages. Two subtle differences between `\@wrindex` and `\codeline@wrindex` are that the first must be a delayed write because the page number is not known yet, and it must close a group and finish some space-hack.

We also provide versions for our use that refer

```
\cs_gset_protected:Npn \@wrindex #1
  {
    \protected@write \@indexfile {}
      { \string \indexentry {#1} { MMMMI - \thepage } }
    \endgroup \@esphack
  }
\cs_gset_protected:Npn \codeline@wrindex #1
  {
    \immediate\write\@indexfile
      {
```

```
        \string\indexentry{#1}
          { MMMMV - \filesep \int_use:N \c@CodelineNo }
      }
  }
\tl_gclear:N \filesep
\cs_new_protected:Npn \__codedoc_index_page_hc:nn #1#2
  {
    \protected@write \@indexfile {}
      {
        \string \indexentry { #1 \encapchar hdpindex{#2} }
          { MMMMI - \thepage }
      }
  }
\cs_new_protected:Npn \__codedoc_index_codeline_hc:nn #1#2
  {
    \immediate\write\@indexfile
      {
        \string \indexentry { #1 \encapchar hdclindex{\the\c@HD@hypercount}{#2} }
          { MMMMV - \filesep \int_use:N \c@CodelineNo - MMMD - \the\c@HD@hypercount - M }
      }
  }
```

We already have a single `HD.xx` target per code line. It would be better to have a target `CL.\the\c@CodelineNo` per code line and change `hdclindex{\the\c@HD@hypercount}` to a mechanism closer to `hdpindex`, but we need to understand better the different types of indexings, and there are subtleties with indexing `\{` and `\}`.

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

`\docincludeaux`

```
\cs_gset:Npn \docincludeaux
  {
    \tl_set:Nn \thepart { \alphalph { part } }
    \tl_set:Nn \filesep { \thepart - }
    \cs_set_eq:NN \filekey \use_none:n
    \tl_gput_right:Nn \index@prologue
      {
        \cs_gset:Npn \@oddfoot
          {
            \parbox { \textwidth }
              {
                \strut \footnotesize
                \raggedright { \bfseries File~Key: } ~ \filekey
```

```
            }
          }
        \cs_set_eq:NN \@evenfoot \@oddfoot
      }
    \cs_gset_eq:NN \docincludeaux \relax
    \cs_gset:Npn \@oddfoot
      {
        \cs_if_exist:cTF { ver @ \currentfile }
          { File~\thepart :~{\ttfamily\currentfile}~ }
          {
            \GetFileInfo{\currentfile}
            File~\thepart :~{\ttfamily\filename}~
            Date:~\ExplFileDate\ % space
            Version~\ExplFileVersion
          }
        \hfill \thepage
      }
    \cs_set_eq:NN \@evenfoot \@oddfoot
  }
```

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

5.12.1 The `macrocode` environment

`\xmacro@code`
`\__codedoc_xmacro_code:n`
`\__codedoc_xmacro_code:w`

Hook into the `macrocode` environment in a dirty way: `\xmacro@code` is responsible for grabbing (and tokenizing) the body of the environment. Redefine it to pass what it grabs to `\__codedoc_xmacro_code:n`. This new macro replaces all `@@` by the appropriate module name. One exceptional case is the `<@@=`⟨*module*⟩`>` lines themselves, where `@@` should not be modified. Actually, we search for such lines, to set the module name automatically. We need to be careful: no `<@@=` should appear as such in the code below since l3doc is also typeset using this code. At each `<@@=` found, replace the ⟨*module*⟩ in the code behind it, update the ⟨*module*⟩, and loop to check for further occurrences of `<@@=`.

```
\group_begin:
  \char_set_catcode_other:N \^^A
  \char_set_catcode_active:N \^^S
  \char_set_catcode_active:N \^^B
  \char_set_catcode_other:N \^^L
  \char_set_catcode_other:N \^^R
  \char_set_lccode:nn { `\^^A } { `\% }
  \char_set_lccode:nn { `\^^S } { `\  }
```

```
  \char_set_lccode:nn { `\^^B } { `\\ }
  \char_set_lccode:nn { `\^^L } { `\{ }
  \char_set_lccode:nn { `\^^R } { `\} }
  \tex_lowercase:D
    {
      \group_end:
      \cs_set_protected:Npn \xmacro@code
          #1 ^^A ^^S^^S^^S^^S ^^Bend ^^Lmacrocode^^R
        { \__codedoc_xmacro_code:n {#1} \end{macrocode} }
    }
\group_begin:
  \char_set_catcode_active:N \<
  \char_set_catcode_active:N \>
  \cs_new_protected:Npn \__codedoc_xmacro_code:n #1
    {
      \tl_clear:N \l__codedoc_tmpa_tl
      \tl_if_in:nnTF {#1} { < @ @ = }
        { \__codedoc_xmacro_code:w #1 < @ @ = \q_recursion_tail > \q_recursion_stop }
        {
          \tl_set:Nn \l__codedoc_tmpa_tl {#1}
          \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
          \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
          \tl_use:N \l__codedoc_tmpa_tl
        }
    }
  \cs_new_protected:Npn \__codedoc_xmacro_code:w #1 < @ @ = #2 >
    {
      % Add code before <@@=...>
      \tl_set:Nn \l__codedoc_tmpb_tl {#1}
      \__codedoc_detect_internals:N \l__codedoc_tmpb_tl
      \__codedoc_replace_at_at:N \l__codedoc_tmpb_tl
      \tl_put_right:NV \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
      % Check for \q_recursion_tail
      \quark_if_recursion_tail_stop_do:nn {#2}
        { \tl_use:N \l__codedoc_tmpa_tl }
      % Change module name and add <@@=#2> to typeset output
      \tl_gset:Nn \g__codedoc_module_name_tl {#2}
      \tl_put_right:Nn \l__codedoc_tmpa_tl { < \text { \verbatim@font @ @ = #2 } > }
      % Loop
      \__codedoc_xmacro_code:w
    }
\group_end:
```

(*End of definition for* `\xmacro@code`, `\__codedoc_xmacro_code:n`, *and* `\__codedoc_xmacro_code:w`. *This function is documented on page* **??**.)

5.13 At end document

Print all defined and documented macros/functions.

```
\iow_new:N \g__codedoc_func_iow

\tl_new:N \l__codedoc_doc_def_tl
\tl_new:N \l__codedoc_doc_undef_tl
\tl_new:N \l__codedoc_undoc_def_tl
\tl_const:Nn \c__codedoc_iow_separator_tl { ---- }
\tl_const:Nn \c__codedoc_iow_midrule_tl  { -- }

\cs_new_protected:Npn \__codedoc_show_functions_defined:
  {
    \bool_lazy_and:nnT
      { \g__codedoc_typeset_implementation_bool } { \g__codedoc_checkfunc_bool }
      {
        \iow_term:e { \c__codedoc_iow_separator_tl \iow_newline: }
        \iow_open:Nn \g__codedoc_func_iow { \c_sys_jobname_str .cmds }

        \tl_clear:N \l__codedoc_doc_def_tl
        \tl_clear:N \l__codedoc_doc_undef_tl
        \tl_clear:N \l__codedoc_undoc_def_tl
        \seq_gremove_duplicates:N \g_doc_functions_seq
        \seq_gremove_duplicates:N \g_doc_macros_seq
        \seq_map_inline:Nn \g_doc_functions_seq
          {
            \seq_if_in:NnTF \g_doc_macros_seq {##1}
              {
                \tl_put_right:Ne \l__codedoc_doc_def_tl
                  { \iow_newline: > ~ ##1 }
              }
              {
                \tl_put_right:Ne \l__codedoc_doc_undef_tl
                  { \iow_newline: ! ~ ##1 }
              }
          }
        \seq_map_inline:Nn \g_doc_macros_seq
          {
            \seq_if_in:NnF \g_doc_functions_seq {##1}
              {
```

```
                \tl_put_right:Ne \l__codedoc_undoc_def_tl
                  { \iow_newline: ? ~ ##1 }
              }
          }
        \__codedoc_functions_typeout:nN
          {
            Functions~both~documented~and~defined: \iow_newline:
            (In~order~of~being~documented)
          }
          \l__codedoc_doc_def_tl
        \__codedoc_functions_typeout:nN
          { Functions~documented~but~not~defined: }
          \l__codedoc_doc_undef_tl
        \__codedoc_functions_typeout:nN
          { Functions~defined~but~not~documented: }
          \l__codedoc_undoc_def_tl

        \iow_close:N \g__codedoc_func_iow
        \iow_term:e { \c__codedoc_iow_separator_tl }
      }
  }
\AtEndDocument { \__codedoc_show_functions_defined: }
```

TODO: use `\iow_term:e`.

```
\cs_new_protected:Npn \__codedoc_functions_typeout:nN #1#2
  {
    \tl_if_empty:NF #2
      {
        \iow_now:Ne \g__codedoc_func_iow
          {
            \c__codedoc_iow_midrule_tl \iow_newline:
            #1 \iow_newline:
            \c__codedoc_iow_midrule_tl
            #2
          }
        \tl_clear:N #2
      }
  }

\cs_new_protected:Npn \__codedoc_show_not_tested:
  {
    \bool_if:NT \g__codedoc_checktest_bool
      {
```

```
\tl_clear:N \l__codedoc_tmpa_tl
\prop_if_empty:NF \g__codedoc_missing_tests_prop
  {
    \cs_set:Npn \__codedoc_tmpa:w ##1##2
      {
        \iow_newline:
        \space\space\space\space \exp_not:n {##1}
        \clist_map_function:nN {##2} \__codedoc_tmpb:w
      }
    \cs_set:Npn \__codedoc_tmpb:w ##1
      {
        \iow_newline:
        \space\space\space\space\space\space * ~ ##1
      }
    \tl_put_right:Ne \l__codedoc_tmpa_tl
      {
        \iow_newline: \iow_newline:
        The~ following~ macro(s)~ have~ incomplete~ tests:
        \iow_newline:
        \prop_map_function:NN
          \g__codedoc_missing_tests_prop \__codedoc_tmpa:w
      }
  }
\seq_if_empty:NF \g__codedoc_not_tested_seq
  {
    \cs_set:Npn \__codedoc_tmpa:w ##1
      { \clist_map_function:nN {##1} \__codedoc_tmpb:w }
    \cs_set:Npn \__codedoc_tmpb:w ##1
      {
        \iow_newline:
        \space\space\space\space ##1
      }
    \tl_put_right:Ne \l__codedoc_tmpa_tl
      {
        \iow_newline:
        \iow_newline:
        The~ following~ macro(s)~ do~ not~ have~ any~ tests:
        \iow_newline:
        \seq_map_function:NN
          \g__codedoc_not_tested_seq \__codedoc_tmpa:w
      }
  }
```

```
        \tl_if_empty:NF \l__codedoc_tmpa_tl
          {
            \int_set:Nn \l__codedoc_tmpa_int { \tex_interactionmode:D }
            \errorstopmode
            \ClassError { l3doc } { \l__codedoc_tmpa_tl } { }
            \int_set:Nn \tex_interactionmode:D { \l__codedoc_tmpa_int }
          }
      }
  }
\AtEndDocument { \__codedoc_show_not_tested: }
```

5.14 Indexing

5.14.1 Necessary patching

The following is useful to set up hyperref targets, for instance for the purpose of indexing. Contrarily to hypdoc we do not try to save pdf destinations, as this leads to too many pdfTEX warnings on early runs.

```
\cs_new_protected:Npn \__codedoc_target:
  {
    \mode_leave_vertical:
    \group_begin:
      \HD@savedestfalse \HD@target
    \group_end:
  }
```

Force targets on every code line.

```
\cs_set_nopar:Npe \theCodelineNo
  {
    \group_begin:
      \exp_not:N \HD@savedestfalse
      \exp_not:o \theCodelineNo
    \group_end:
  }
```

Inside the table of contents (and other similar lists introduced by `\@starttoc`), we suppress indexing. This is because `\cmd`, `\cs`, or `\tn` appearing in titles only gets typeset in the second run, and getting their indexing right would require even more runs than we already need. Besides, it is not useful to index uses of some command in the table of contents.

```
\bool_new:N \l__codedoc_allow_indexing_bool
\bool_set_true:N \l__codedoc_allow_indexing_bool
```

```
\use:e
  {
    \exp_not:n { \cs_set_nopar:Npn \@starttoc #1 }
      {
        \group_begin:
          \bool_set_false:N \l__codedoc_allow_indexing_bool
          \exp_not:o { \@starttoc {#1} }
        \group_end:
      }
  }
```

5.14.2 Userspace commands

Fix index (for now):

```
\g@addto@macro \theindex { \MakePrivateLetters }
\cs_gset:Npn \verbatimchar {&}

\setcounter { IndexColumns } { 2 }
```

Set up the Index to use `\part`

```
\IndexPrologue
  {
    \part*{Index}
    \markboth{Index}{Index}
    \addcontentsline{toc}{part}{Index}
    The~italic~numbers~denote~the~pages~where~the~
    corresponding~entry~is~described,~
    numbers~underlined~point~to~the~definition,~
    all~others~indicate~the~places~where~it~is~used.
  }
```

`\SpecialIndex` An attempt at affecting how commands which appear within the `macrocode` environment are treated in the index.

```
\cs_gset_protected:Npn \SpecialIndex #1
  {
    \@bsphack
    \__codedoc_special_index:nn {#1} { }
    \@esphack
  }
```

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

```
\msg_new:nnn { l3doc } { print-index-howto }
```

```
  {
    Generate~the~index~by~executing\\
    \iow_indent:n
      { makeindex~-s~gind.ist~-o~\c_sys_jobname_str.ind~\c_sys_jobname_str.idx }
  }
\tl_gput_right:Nn \PrintIndex
  { \AtEndDocument { \msg_info:nn { l3doc } { print-index-howto } } }
```

5.14.3 Internal index commands

\it@is@a

The index of one-character commands within the `macrocode` environment is produced using `\it@is@a` ⟨*char*⟩. Alter that command.

```
\cs_gset_protected:Npn \it@is@a #1
  {
    \use:e
      {
        \__codedoc_special_index_module:nnnnN
          {#1}
          { \bslash #1 }
          { }
          { }
          \c_false_bool
      }
  }
```

(*End of definition for* `\it@is@a`. *This function is documented on page* **??**.)

__codedoc_special_index:nn

```
\cs_new_protected:Npn \__codedoc_special_index:nn #1#2
  {
    \__codedoc_key_get:n {#1}
    \quark_if_no_value:NF \l__codedoc_override_module_tl
      { \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_override_module_tl }
    \__codedoc_special_index_module:ooonN
      { \l__codedoc_index_key_tl }
      { \l__codedoc_index_macro_tl }
      { \l__codedoc_index_module_tl }
      {#2}
      \l__codedoc_index_internal_bool
  }
\cs_generate_variant:Nn \__codedoc_special_index:nn { o }
```

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

__codedoc_special_index_module:nnnnN
__codedoc_special_index_module:ooonN
__codedoc_special_index_aux:nnnnnn
__codedoc_special_index_set:Nn

Remotely based on Heiko's replacement to play nicely with hypdoc. We use `\verb` or a `\verbatim@font` construction depending on whether the number of tokens in `#2` is equal to its number of characters: if it is not then that suggests that there is a construct such as `\meta{...}`.

```
\tl_new:N \l__codedoc_index_escaped_macro_tl
\tl_new:N \l__codedoc_index_escaped_key_tl

\cs_new_protected:Npn \__codedoc_special_index_module:nnnnN #1#2#3#4#5
```

`#1` : key
`#2` : macro
`#3` : module
`#4` : index 'type' (`main`/`usage`/*etc.*)
`#5` : boolean whether internal command

```
{
  \use:e
    {
      \exp_not:n { \__codedoc_special_index_aux:nnnnnn {#1} {#2} }
        \tl_if_empty:nTF {#3}
          { { } { } { } }
          {
            \str_if_eq:nnTF {#3} { TeX }
              {
                { TeX~and~LaTeX2e }
                { \string\TeX{}~and~\string\LaTeXe{} }
              }
              {
                {#3}
                { \string\pkg{#3} }
              }
            { \bool_if:NT #5 { ~internal } ~commands: }
          }
    }
        {#4}
}

\cs_generate_variant:Nn \__codedoc_special_index_module:nnnnN { ooo }

\cs_new_protected:Npn \__codedoc_special_index_aux:nnnnnn #1#2#3#4#5#6
```

`#1` : key
`#2` : macro

#3 : index subheading string

#4 : index subheading text

#5 : index subheading suffix (appended to both arg 3 and 4)

#6 : index 'type' (`main`/`usage`/*etc.*)

```
{
  \tl_set:Nn \l__codedoc_index_escaped_key_tl {#1}
  \__codedoc_quote_special_char:N \l__codedoc_index_escaped_key_tl
  \__codedoc_special_index_set:Nn \l__codedoc_index_escaped_macro_tl {#2}
  \str_if_eq:onTF { \@currenvir } { macrocode }
    { \__codedoc_index_codeline_hc:nn }
    {
      \str_case:nnF {#6}
        {
          { main }  { \__codedoc_index_codeline_hc:nn }
          { usage } { \__codedoc_index_page_hc:nn }
        }
        { \__codedoc_target: \__codedoc_index_page_hc:nn }
    }
    {
      \tl_if_empty:nF { #3 #4 #5 }
        { #3 #5 \actualchar #4 #5 \levelchar }
      \l__codedoc_index_escaped_key_tl
      \actualchar
      {
        \token_to_str:N \verbatim@font \c_space_tl
        \l__codedoc_index_escaped_macro_tl
      }
    }
    {#6}
}
```

Note that `#3` here could contain `MMMMI-` or `MMMMV-` more than once if several successive code lines have been merged into a range somehow. Note incidentally that the dash is active in some of our sources, like `interface3.tex` or `source2e.tex`.

```
\group_begin:
\char_set_active_eq:NN - \scan_stop:
\tl_const:Ne \c__codedoc_active_minus_tl { \char_generate:nn { `- } { 13 } }
\group_end:
\cs_new_eq:NN \__codedoc_old_hdpindex:nn \hdpindex
\cs_new_eq:NN \__codedoc_old_hdclindex:nnn \hdclindex
\cs_gset_protected:Npn \hdpindex #1
```

\hdpindex
__codedoc_old_hdpindex:nn
\hdclindex
__codedoc_old_hdclindex:nnn
__codedoc_hdindex:nn
\c__codedoc_active_minus_tl
__codedoc_hdindex_aux:nn
__codedoc_hdindex_aux:w

```
  { \__codedoc_hdindex:nn { \__codedoc_old_hdpindex:nn {#1} } }
\cs_gset_protected:Npn \hdclindex #1#2
  { \__codedoc_hdindex:nn { \__codedoc_old_hdclindex:nnn {#1} {#2} } }
\cs_new_protected:Npn \__codedoc_hdindex:nn #1#2
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#2}
    \tl_replace_all:Nen \l__codedoc_tmpa_tl
      { \exp_not:V \c__codedoc_active_minus_tl \exp_not:V \c__codedoc_active_minus_tl }
      { -- }
    \seq_set_split:NnV \l__codedoc_tmpa_seq { -- } \l__codedoc_tmpa_tl
    \seq_set_map:NNn \l__codedoc_tmpa_seq \l__codedoc_tmpa_seq
      { \__codedoc_hdindex_aux:nn {#1} {##1} }
    \seq_use:Nn \l__codedoc_tmpa_seq { -- }
  }
\cs_new_protected:Npn \__codedoc_hdindex_aux:nn #1#2
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#2}
    \tl_replace_all:Nnn \l__codedoc_tmpa_tl { MMMM } { \use_none:nn }
    \tl_if_in:NnT \l__codedoc_tmpa_tl { MMMD }
      {
        \tl_replace_all:Nen \l__codedoc_tmpa_tl
          { \exp_not:V \c__codedoc_active_minus_tl MMMD } { - MMMD }
        \tl_replace_all:Nnn \l__codedoc_tmpa_tl { - MMMD } { \__codedoc_hdindex_aux:w }
      }
    \use:e { \exp_not:n {#1} { \exp_not:V \l__codedoc_tmpa_tl } }
  }
\cs_new_protected:Npn \__codedoc_hdindex_aux:w #1 M { }

\cs_new_protected:Npn \__codedoc_special_index_set:Nn #1#2
  {
    \__kernel_tl_set:Ne #1 { \tl_to_str:n {#2} }
    \__codedoc_if_almost_str:nTF {#2}
      {
        \tl_replace_all:Nen #1 { \tl_to_str:n { __ } }
          {
            \verbatimchar
            \token_to_str:N \_ \token_to_str:N \_
            \token_to_str:N \verb * \verbatimchar
          }
        \exp_args:Ne \tl_map_inline:nn
          { \tl_to_str:N \verbatimchar \token_to_str:N _ }
          {
            \tl_replace_all:Nnn #1 {##1}
```

```
              {
                \verbatimchar \c_backslash_str ##1
                \token_to_str:N \verb * \verbatimchar
              }
          }
        \__kernel_tl_set:Ne #1
          {
            \token_to_str:N \verb * \verbatimchar
            #1 \verbatimchar
          }
      }
      {
        \tl_set:Nn #1 {#2}
        \tl_replace_all:NVn #1
          \c_backslash_str
          { \token_to_str:N \bslash \c_space_tl }
      }
    \__codedoc_quote_special_char:N #1
  }
```

(*End of definition for* `\__codedoc_special_index_module:nnnnN` *and others. These functions are documented on page* **??**.)

`\__codedoc_quote_special_char:N` Quote some special characters.

```
\cs_new_protected:Npn \__codedoc_quote_special_char:N #1
  {
    \tl_map_inline:nn { \quotechar \actualchar \encapchar \levelchar \bslash }
      {
        \tl_replace_all:Nen #1
          { \tl_to_str:N ##1 } { \quotechar \tl_to_str:N ##1 }
      }
  }
```

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

5.14.4 Finding sort-key and module

`\__codedoc_key_get:n` Sets `\l__codedoc_index_macro_tl`, `\l__codedoc_index_key_tl`, and `\l__codedoc_index_module_tl` from `#1`. The base function is stored by `\__codedoc_key_get_base:nN` in `\l__codedoc_index_macro_tl`, falling back to `#1` if it contains markup or has no signature.

The starting point for the ⟨*key*⟩ is `\l__codedoc_index_key_tl` as a string. If it the first character is a backslash, remove it. Then recognize `expl` functions and

variables by the presence of `:` or `_` and TEX/LATEX 2ε commands by the presence of `@`. For `expl` names, we call `\__codedoc_key_func:` or `\__codedoc_key_var:`, which are responsible for removing some characters and finding the module name, while for TEX/LATEX 2ε commands the module name is `TeX`, and others have an empty module name.

```
\cs_new_protected:Npe \__codedoc_key_get:n #1
  {
    \exp_not:N \__codedoc_key_get_base:nN {#1} \exp_not:N \l__codedoc_index_macro_tl
    \__kernel_tl_set:Ne \exp_not:N \l__codedoc_index_key_tl
      { \exp_not:N \tl_to_str:N \exp_not:N \l__codedoc_index_macro_tl }
    \tl_clear:N \exp_not:N \l__codedoc_index_module_tl
    \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \tl_to_str:n { __ } }
      { \bool_set_true:N \exp_not:N \l__codedoc_index_internal_bool }
      { \bool_set_false:N \exp_not:N \l__codedoc_index_internal_bool }
    \exp_not:N \tl_if_head_eq_charcode:VNT
      \exp_not:N \l__codedoc_index_key_tl \c_backslash_str
      { \exp_not:N \__codedoc_key_pop: }
    \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N : }
      { \exp_not:N \__codedoc_key_func: }
      {
        \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N _ }
          { \exp_not:N \__codedoc_key_var: }
          {
            \tl_if_in:NnT \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N @ }
              { \tl_set:Nn \exp_not:N \l__codedoc_index_module_tl { TeX } }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_key_pop:
  {
    \__kernel_tl_set:Ne \l__codedoc_index_key_tl
      { \tl_tail:N \l__codedoc_index_key_tl }
  }
```

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

`\__codedoc_key_trim_module:n`
`\__codedoc_key_drop_underscores:`

Helper that removes from `\l__codedoc_index_module_tl` everything after the first occurence of `#1`. Helper that removes any leading underscore from `\l__codedoc_index_key_tl`.

```
\cs_new_protected:Npn \__codedoc_key_trim_module:n #1
  {
    \cs_set:Npn \__codedoc_tmpa:w ##1 #1 ##2 \q_stop
```

```
      { \exp_not:n {##1} }
    \__kernel_tl_set:Ne \l__codedoc_index_module_tl
      { \exp_after:wN \__codedoc_tmpa:w \l__codedoc_index_module_tl #1 \q_stop }
  }
\cs_new_protected:Npn \__codedoc_key_drop_underscores:
  {
    \tl_if_head_eq_charcode:VNT \l__codedoc_index_key_tl _
      { \__codedoc_key_pop: \__codedoc_key_drop_underscores: }
  }
```

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

`\__codedoc_key_func:` The function `\__codedoc_key_func:` is used if there is a colon, so either for usual expl3 functions or for keys from l3keys. After removing from the key a leading dot (for the latter case), and any leading underscore, the module name is the part before any colon or underscore.

```
\cs_new_protected:Npn \__codedoc_key_func:
  {
    \tl_if_head_eq_charcode:VNT \l__codedoc_index_key_tl .
      { \__codedoc_key_pop: }
    \__codedoc_key_drop_underscores:
    \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
    \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N : }
    \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
  }
```

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

`\__codedoc_key_var:` `\__codedoc_key_get_module:` The function `\__codedoc_key_var:` covers cases with no `:` but with `_`, typically variables but occasionally non-expl3 functions such as Lua function with underscores. First test the second character: if that is `_` then assume we have a proper variable, otherwise use the part before any underscore as the module name. For variables, distinguish quarks and scan marks (starting with `q` and `s`), then drop the first letter (local/global/constant marker) and underscores to improve the index sorting. Then get the module as the first (underscore-delimited) "word". In the past, we distinguished according to how many such words there were, to detect commands like `\c_zero`, which should be sorted as `int` variables, and `\l_tmpa_dim`, which should be sorted in the `dim` and not the `tmpa` module. Now the first case has been deprecated for some time, while `tmpa` and similar are special-cased through an explicit list given below. The way it works is that if the module is in a list of special names that

are not valid modules, then we try the last word, and if that also fails (for instance in the deprecated `\c_one_hundred`) we empty the module completely.

```
\cs_new_protected:Npn \__codedoc_key_var:
  {
    \exp_args:Ne \tl_if_head_eq_charcode:nNTF
      { \exp_args:No \str_tail:n \l__codedoc_index_key_tl } _
      {
        \str_case:en { \str_head:N \l__codedoc_index_key_tl }
          {
            { q } { \tl_set:Nn \l__codedoc_index_module_tl { quark } }
            { s } { \tl_set:Nn \l__codedoc_index_module_tl { scan } }
          }
        \__codedoc_key_pop:
        \__codedoc_key_pop:
        \__codedoc_key_drop_underscores:
        \tl_if_empty:NT \l__codedoc_index_module_tl
          {
            \seq_set_split:NoV \l__codedoc_tmpa_seq
              { \token_to_str:N _ } \l__codedoc_index_key_tl
            \seq_get_left:NN \l__codedoc_tmpa_seq \l__codedoc_index_module_tl
            \clist_if_in:NoT \g__codedoc_non_modules_clist \l__codedoc_index_module_tl
              {
                \seq_get_right:NN \l__codedoc_tmpa_seq \l__codedoc_index_module_tl
                \clist_if_in:NoT \g__codedoc_non_modules_clist \l__codedoc_index_module_tl
                  {
                    \tl_clear:N \l__codedoc_index_module_tl
                  }
              }
          }
      }
      {
        \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
        \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
      }
  }
```

(*End of definition for* `\__codedoc_key_var:` *and* `\__codedoc_key_get_module:`.)

`\g__codedoc_non_modules_clist` List of names that appear as the first word in an expl3 command, but that are not true modules, so that they should be sorted differently in an index.

```
\clist_new:N \g__codedoc_non_modules_clist
\clist_gset:Ne \g__codedoc_non_modules_clist
```

```
  {
    \tl_to_str:n
      {

        alignment, ampersand, atsign, backslash, catcode, circumflex,
        code, colon, document, dollar, e, empty, false, hash, inf,
        initex, job, left, log, math, mark, max, minus, nan, nil, no,
        novalue, other, parameter, percent, pi, recursion, right, space,
        stop, term, tilde, tmpa, tmpb, true, underscore, zero, one, two,
        three, four, five, six, seven, eight, nine, ten, eleven, twelve,
        thirteen, fourteen, fifteen, sixteen, thirty, hundred

      }
  }
```

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

5.15 Change history

Set the change history to use `\part`. Allow control names to be hyphenated in here...

```
\GlossaryPrologue
  {
    \part*{Change~History}
    {\GlossaryParms\ttfamily\hyphenchar\font=`\-}
    \markboth{Change~History}{Change~History}
    \addcontentsline{toc}{part}{Change~History}
  }

\msg_new:nnn { l3doc } { print-changes-howto }
  {
    Generate~the~change~list~by~executing\\
    \iow_indent:n
      { makeindex~-s~gglo.ist~-o~\c_sys_jobname_str.gls~\c_sys_jobname_str.glo }
  }
\tl_gput_right:Nn \PrintChanges
  { \AtEndDocument { \msg_info:nn { l3doc } { print-changes-howto } } }
```

5.16 Default configuration

```
\bool_if:NTF \g__codedoc_typeset_implementation_bool
  {
    \RecordChanges
```

```
    \CodelineIndex
    \EnableCrossrefs
    \AlsoImplementation
  }
  {
    \CodelineNumbered
    \DisableCrossrefs
    \OnlyDescription
  }
```

⟨/class⟩

5.17 Internal macros for LaTeX3 sources

These definitions are only used by the LaTeX3 documentation; they are not necessary for third-party users of l3doc. In time this will be broken into a separate package that is specifically loaded in the various expl3 modules, *etc.*

⟨*cfg⟩

The Guilty Parties.

```
\tl_const:Nn \Team
  {
    The~\LaTeX3~Project\thanks
      {\url{https://www.latex-project.org/latex3/}}
  }

\NewDocumentCommand{\ExplMakeTitle}{mm}
  {
    \title
      {
       The~\pkg{#1}~package \\ #2
      }
    \author
      {
       The~\LaTeX3~Project\thanks{E-mail:~
       \href{mailto:latex-l@listserv.uni-heidelberg.de}
            {latex-l@listserv.uni-heidelberg.de}}
      }
    \date{Released~\ExplFileDate}
    \maketitle
  }
```

5.18 Math extras

For l3fp.

```
\AtBeginDocument
  {
    \clist_map_inline:nn
      {
       asin, acos, atan, acot,
       asinh, acosh, atanh, acoth, round, floor, ceil
      }
      { \exp_args:Nc \DeclareMathOperator{#1}{#1} }
  }
```

\nan

```
\NewDocumentCommand { \nan } { } { \text { \texttt { nan } } }
```

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

```
⟨/cfg⟩
```

5.19 Makeindex configuration

```
⟨*docist⟩
```

The makeindex style `l3doc.ist` is used in place of the usual `gind.ist` to ensure that `I` is used in the sequence `I J K` not `I II II`, which would be the default makeindex behaviour.

Will: Do we need this?

Frank: at the moment we do not distribute or generate this file. `gind.ist` is used instead.

```
actual '='
quote '!'
level '>'
preamble
"\n \\begin{theindex} \n \\makeatletter\\scan@allowedfalse\n"
postamble
"\n\n \\end{theindex}\n"
item_x1   "\\efill \n \\subitem "
item_x2   "\\efill \n \\subsubitem "
delim_0   "\\pfill "
delim_1   "\\pfill "
```

```
delim_2   "\\pfill "
% The next lines will produce some warnings when
% running Makeindex as they try to cover two different
% versions of the program:
lethead_prefix   "{\\bfseries\\hfil "
lethead_suffix   "\\hfil}\\nopagebreak\n"
lethead_flag       1
heading_prefix   "{\\bfseries\\hfil "
heading_suffix   "\\hfil}\\nopagebreak\n"
headings_flag      1

% and just for source3:
% Remove R so I is treated in sequence I J K not I II III
page_precedence "rnaA"
```

(*End of definition for .*)

⟨/docist⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

\" 486, 491
\# 597
\% 1895
\& 1716, 1729
\, ... 1194, 1200, 1575, 1577
\- 2373
\/ 724
\: 433
\< 1025, 1026, 1908
\= 1731
\> 1726, 1909
\\ 453, 1115, 1132, 1194, 1200, 1359, 1726, 1897, 2113, 2379, 2408
\{ 531, 1898
\} 531, 1899
\␣ 605, 610, 619, 628, 1437, 1882, 1896
\^ 135, 780, 782, 783, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899
_ 432, 665, 1563, 2238
\| 487, 492

A

\A 962
\actualchar 2186, 2188, 2266
\addcontentsline 2099, 2375
\addpenalty 469
\AddToHook 1678, 1679
\addtolength 455, 456, 457
\addvspace 470
\advance 476

\allowbreak 1005
\alphalph 1859
\AlsoImplementation *6*, 1734, 2390
\arabic 439, 597
\Arg *8*, *33*, 530
arguments (env.) *11*, 593
\arrayrulecolor 1216, 1228
\AtBeginDocument 484, 2419
\AtEndDocument 489, 1995, 2065, 2118, 2384
\author 2410

B

\baselineskip . . . 608, 617, 626, 1336, 1377
\begin . . 1096, 1217, 1219, 1232, 1613, 1623
\begingroup 472, 1614, 1624, 1683, 1700, 1724
\bfseries 475, 1869
\bigskip 999, 1694, 1719, 1733
bool commands:
\bool_gset_false:N 371, 544, 546, 1735
\bool_gset_true:N 41, 365, 366, 370, 540, 542, 1737
\bool_if:NTF . . . 131, 169, 261, 278, 419, 549, 553, 556, 560, 563, 568, 648, 657, 662, 674, 676, 773, 805, 816, 859, 865, 879, 907, 1055, 1108, 1111, 1127, 1128, 1148, 1154, 1161, 1173, 1178, 1211, 1230, 1405, 1407, 1413, 1448, 1471, 1549, 1550, 1598, 1630, 1640, 1740, 2012, 2163, 2385
\bool_lazy_all:nTF 1537
\bool_lazy_and:nnTF 1187, 1947
\bool_lazy_any:nTF 1415
\bool_lazy_any_p:n 1586
\bool_lazy_or:nnTF 264
\bool_new:N 6, 15, 16, 19, 23, 24, 25, 26, 27, 28, 29, 30, 31, 35, 36, 37, 38, 39, 40, 48, 55, 67, 68, 69, 70, 71, 77, 2080
\bool_set:Nn 1041
\bool_set_false:N . 550, 642, 922, 927, 936, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1266, 1278, 1283, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 2087, 2280
\bool_set_true:N 7, 72, 73, 557, 643, 916, 921, 928, 933, 934, 935, 941, 942, 952, 1022, 1249, 1252, 1256, 1259, 1263, 1264, 1265, 1271, 1272, 1277, 1284, 1288, 1478, 2081, 2279
\c_false_bool 216, 863, 866, 1398, 2128
\c_true_bool 215, 1388
\bottomrule 1101
box commands:
\box_clear:N 1328
\box_dp:N 1004
\box_gclear:N 1012
\box_if_empty:NTF 1009, 1206
\box_new:N 14, 61, 62
\box_use_drop:N 1054, 1234
\box_wd:N 1040, 1141
\l_tmpa_box 1138, 1141, 1146, 1152
\bslash 2125, 2260, 2266

C

\c 666
\catcode 1359
\changes *61*
\char 531, 1610
char commands:
\char_generate:nn 2198
\char_set_active_eq:NN . . . 1025, 2197
\char_set_catcode:nn 706
\char_set_catcode_active:N 135, 1026, 1891, 1892, 1908, 1909
\char_set_catcode_letter:N 431, 432, 433
\char_set_catcode_other:N 1890, 1893, 1894
\char_set_lccode:nn 1895, 1896, 1897, 1898, 1899
check (option) *6*
checktest (option) *6*
\ClassError 2060
\clearpage 1773, 1784, 1818

clist commands:
\clist_clear:N 1023
\clist_count:N *15*
\clist_count:n *15*
\clist_gset:Nn 2355
\clist_if_in:NnTF 2339, 2342
\clist_map_function:NN 1770
\clist_map_function:nN ... 2021, 2040
\clist_map_inline:Nn 1163, 1790
\clist_map_inline:nn 1763, 2421
\clist_new:N 3, 76, 2354
\clist_put_right:Nn 1765
\clist_set:Nn 951
\closeout 1820
\cls *8*, 535
\cmd *6*, *7*, *17*, *37*, *81*, 505, 525
codedoc internal commands:
\c__codedoc_active_minus_tl ... 2196
\l__codedoc_allow_indexing_bool 657, 674, 2080, 2081, 2087
__codedoc_base_form_aux:nnN 234, 258, 286, 811
__codedoc_base_form_aux:nnnnnN 274, 276
__codedoc_base_form_signature_do:nnn 271, 271
\g__codedoc_base_name_tl 74, 841, 842, 848, 854
\g__codedoc_checkfunc_bool 35, 373, 1948
\g__codedoc_checktest_bool 35, 374, 1539, 2012
__codedoc_cmd:nn *32*, 506, 508, 511, 640, 640, 698
\l__codedoc_cmd_index_tl 64, 635, 644, 678, 681
\l__codedoc_cmd_module_tl 64, 636, 645, 684, 687
\l__codedoc_cmd_noindex_bool *37*, 64, 637, 642, 676
\l__codedoc_cmd_replace_bool 64, 638, 643, 648
\l__codedoc_cmd_tl 64, 647, 651, 652, 653, 659, 661, 667, 670, 671, 680, 683
\g__codedoc_cs_break_bool 35, 378, 662
\l__codedoc_date_added_tl 78, 944, 1188, 1191, 1194
__codedoc_date_compare:nNn 293
__codedoc_date_compare:nNnTF 293, 971
__codedoc_date_compare_aux:nnnNnnn 293, 298, 307
__codedoc_date_compare_aux:w 293, 294, 295
__codedoc_date_compare_p:nNn .. 293
__codedoc_date_set:Nn . 958, 958, 970
__codedoc_date_set_past:Nn *47*, 944, 945, 958, 968
\l__codedoc_date_updated_tl 78, 945, 1189, 1197, 1200
\l__codedoc_descr_coffin 11, 1004, 1011, 1046, 1067, 1070, 1080
__codedoc_detect_internals:N 129, 129, 784, 1917, 1926
__codedoc_detect_internals_aux:N 129, 132, 136
\l__codedoc_detect_internals_bool 6, 131
\l__codedoc_detect_internals_cs_tl 9, 149, 154
\l__codedoc_detect_internals_tl 6, 138, 139, 140, 141, 142, 144, 146, 147, 149, 150, 151, 152, 155
\l__codedoc_doc_def_tl 1940, 1953, 1962, 1983
\l__codedoc_doc_undef_tl 1941, 1954, 1966, 1986
__codedoc_ensuremath_sb:n *39*, 699, 708, 712
\g__codedoc_finale_tl 1734
__codedoc_fn_footnote:nn 1671, 1678, 1679
__codedoc_fn_restore:n .. 1662, 1676

__codedoc_fn_store: 1657, 1674
\g__codedoc_fnmark_prop 1656, 1659, 1664
\g__codedoc_func_iow 1939, 1951, 1991, 2000
__codedoc_function:nnw *50*, 565, 573, 979, 979
__codedoc_function_assemble: .. 994, 1051, 1051
__codedoc_function_descr_start:w 989, 1044, 1044
__codedoc_function_descr_stop: 993, 1044, 1049
__codedoc_function_end: *50*, 570, 574, 979, 991
__codedoc_function_extra_labels: 1098, 1159
__codedoc_function_index:n 1105, 1107, 1118, 1123
__codedoc_function_init: 982, 1007, 1007
__codedoc_function_label:nN ... 1109, 1171, 1184
\l__codedoc_function_label_clist 76, 951, 1023, 1163
__codedoc_function_reset: 988, 1031, 1031
__codedoc_function_typeset: ... 987, 1035, 1035
__codedoc_function_typeset_start: 981, 997, 997
__codedoc_function_typeset_stop: 995, 997, 1001
\l__codedoc_functions_coffin ... *14*, *48*, *50*, *52*, 11, 1014, 1038, 1040, 1063, 1084
__codedoc_functions_typeout:nN 1978, 1984, 1987, 1996
__codedoc_get_function_name:n 202, 206, 206, 1133, 1392
__codedoc_get_function_signature:n 204, 206, 208
__codedoc_get_hyper_target:nN . 762, 762, 769, 1165, 1175, 1445, 1601
__codedoc_gprop_name:n ... 328, 328
__codedoc_hdindex:nn 2196, 2203, 2205, 2206
__codedoc_hdindex_aux:nn 2196, 2214, 2217
__codedoc_hdindex_aux:w 2196, 2225, 2229
__codedoc_if_almost_str:n .. 86, 94
__codedoc_if_almost_str:nTF ... 86, 230, 659, 2233
__codedoc_if_detect_internals_ok:N 161
__codedoc_if_detect_internals_ok:NTF 129, 147
__codedoc_if_macro_internal:n . 877
__codedoc_if_macro_internal:nTF 877, 1466
__codedoc_if_macro_internal_aux:w 877, 884, 891
__codedoc_if_macro_internal_p:n 877, 1417, 1545, 1588
\l__codedoc_in_function_bool ... 15, 1022, 1230
\l__codedoc_in_implementation_bool *34*, 69, 550, 557, 563, 568
__codedoc_index_codeline_hc:nn 1849, 2175, 2179
\l__codedoc_index_escaped_key_tl 2145, 2171, 2172, 2187
\l__codedoc_index_escaped_macro_tl 2144, 2173, 2191
\l__codedoc_index_internal_bool *16*, 51, 694, 2141, 2279, 2280
\l__codedoc_index_key_tl *16*, *87*, *88*, 51, 690, 1428, 2137, 2275, 2278, 2282, 2284, 2287, 2290, 2297, 2298, 2309, 2314, 2317, 2324, 2326, 2337, 2350
\l__codedoc_index_macro_tl *16*, *87*, 51, 691, 2138, 2274, 2276

\l__codedoc_index_module_tl 16, 87, 88, 51, 686, 692, 2135, 2139, 2277, 2291, 2304, 2305, 2317, 2328, 2329, 2334, 2338, 2339, 2341, 2342, 2344, 2350
__codedoc_index_page_hc:nn 1841, 2180, 2182
__codedoc_input:n 1754, 1754, 1766, 1770
\c__codedoc_iow_mid_rule_tl 59
\c__codedoc_iow_midrule_tl 57, 1944, 2002, 2004
\c__codedoc_iow_rule_tl 57
\c__codedoc_iow_separator_tl ... 1943, 1950, 1992
\g__codedoc_kernel_bool 35, 169, 375, 376
__codedoc_key_drop_underscores: 2300, 2307, 2310, 2316, 2333
__codedoc_key_func: 88, 89, 2285, 2312, 2312
__codedoc_key_get:n 683, 2133, 2272, 2272
__codedoc_key_get_base:nN 87, 228, 228, 2274
__codedoc_key_get_base_TF:nN .. 232, 238
__codedoc_key_get_module: ... 2321
__codedoc_key_pop: 2283, 2295, 2310, 2315, 2331, 2332
__codedoc_key_trim_module:n 19, 2300, 2300, 2318, 2319, 2351
__codedoc_key_var: 88, 89, 2288, 2321, 2321
\g__codedoc_lmodern_bool 35, 377, 419
\l__codedoc_long_name_bool 14, 50, 16, 1041, 1055, 1211
__codedoc_lseq_name:n 15, 42, 328, 329, 822
__codedoc_macro:nnw 564, 576, 1299, 1299
\l__codedoc_macro_argument_tl .. 80, 975, 983, 1302
\l__codedoc_macro_box 62, 61, 1329, 1379, 1433, 1435
\l__codedoc_macro_deprecated_-bool 23, 946, 1246, 1315, 1418, 1589
\l__codedoc_macro_do_not_index_-tl 16, 51, 1296, 1356, 1358, 1360, 1362, 1363, 1365
\l__codedoc_macro_documented_tl 23, 1294, 1326, 1345, 1349
__codedoc_macro_dump: 1309, 1368, 1368
__codedoc_macro_end: 569, 577, 1528, 1528
__codedoc_macro_end_check_-tested: 1531, 1535, 1535
__codedoc_macro_end_style:n ... 1533, 1554, 1554
__codedoc_macro_end_wrap_item:n 1559, 1559, 1570
__codedoc_macro_exclude_index: 1306, 1354, 1354
\l__codedoc_macro_EXP_bool 23, 921, 927, 935, 1018, 1127, 1265, 1277, 1283, 1320
__codedoc_macro_index:nN 1406, 1410
\l__codedoc_macro_index_box 61, 1328, 1376, 1422, 1423
__codedoc_macro_init: 1301, 1312, 1312
\l__codedoc_macro_int 61, 1334, 1336, 1440
\l__codedoc_macro_internal_bool 45, 23, 879, 1249, 1252, 1316
\l__codedoc_macro_nodoc_bool ... 23, 947, 1253, 1419, 1590
\l__codedoc_macro_noTF_bool 23, 865, 941, 1017, 1272, 1319
\l__codedoc_macro_pTF_bool 23, 859, 933, 1016, 1264, 1318, 1549

__codedoc_macro_reset: 1310, 1339, 1339
\l__codedoc_macro_rEXP_bool 23, 922, 928, 936, 1019, 1128, 1266, 1278, 1284, 1321
__codedoc_macro_save_names: . . . 1307, 1343, 1343
__codedoc_macro_save_names_-aux:n 1346, 1352
__codedoc_macro_single:nNN *40*, *57*, 1388, 1398, 1401, 1401
\l__codedoc_macro_tested_bool . . 19, 1288, 1323, 1478, 1541
\l__codedoc_macro_TF_bool 23, 816, 867, 907, 916, 934, 942, 1015, 1259, 1263, 1271, 1317, 1550
__codedoc_macro_typeset_-block:nN 874, 1386, 1386
__codedoc_macro_typeset_one:nN 1404, 1431, 1431
__codedoc_macro_typeset_-variant_list:nN . . 1391, 1395, 1400
\l__codedoc_macro_var_bool 23, 1256, 1322, 1540, 1598
__codedoc_macroname_prefix:n 1460, 1464, 1469
__codedoc_macroname_suffix:N 1461, 1470
__codedoc_meta:n 516, 699, 699
__codedoc_meta_original:n 699, 710, 714
\g__codedoc_missing_tests_prop *64*, 19, 1516, 2015, 2034
\g__codedoc_module_name_tl . . 56, 113, 116, 156, 163, 166, 1756, 1933
__codedoc_names_block_base_-map:N 892, 892, 1346
\l__codedoc_names_block_tl *42*, *43*, *45*, *50*, *59*, 46, 792, 825, 827, 835, 894
__codedoc_names_get_seq:nN 770, 770, 985, 1304
__codedoc_names_parse: 790, 790, 986, 1305
__codedoc_names_parse_aux:Nnn 822, 823
__codedoc_names_parse_aux:nnn 808, 810, 818, 821
__codedoc_names_parse_one:n 790, 795, 797
__codedoc_names_parse_one_-aux:nnNn 800, 803
\l__codedoc_names_seq 49, 794, 985, 1304, 1517, 1544
__codedoc_names_typeset: 833, 833, 1099, 1308
__codedoc_names_typeset_auxi:n *43*, 833, 836, 838
__codedoc_names_typeset_auxii:n *43*, 848, 853, 857, 857, 869
__codedoc_names_typeset_-block:nN 861, 866, 867, 870, 870, 876
\l__codedoc_names_verb_bool 48, 773, 955, 1021, 1292, 1324
\l__codedoc_nested_macro_int *44*, *58*, 18, 872, 1314, 1532
\g__codedoc_nested_names_seq 50, 1348, 1353, 1569, 1583, 1584, 1593, 1596, 1602, 1606
\l__codedoc_no_label_bool 76, 952, 1020, 1161, 1173
\g__codedoc_non_modules_clist 2339, 2342, 2354
\g__codedoc_not_tested_seq 19, 1546, 2037, 2053
__codedoc_old_hdclindex:nnn 2196, 2201, 2205
__codedoc_old_hdpindex:nn 2196, 2200, 2203
__codedoc_oldlist:nn . . 458, 458, 460
\l__codedoc_output_coffin 10, 1058, 1062, 1066, 1069, 1075, 1079, 1083, 1086
\l__codedoc_override_module_tl .

. . . . . *14*, 23, 956, 1024, 1033, 1293, 1325, 1341, 2134, 2135
__codedoc_pdfstring_cmd:w 517, 518, 525
__codedoc_pdfstring_cs:w 517, 520, 526, 527
__codedoc_pdfstring_meta:w 517, 521, 528
__codedoc_pdfstring_newline:w 449, 451, 453
__codedoc_predicate_from_base:n 200, 200, 862
__codedoc_print_documented: 1559, 1579, 1581
__codedoc_print_end_definition: 1533, 1559, 1566
__codedoc_print_macroname:nN 1436, 1442, 1442
__codedoc_print_testfile:n 1327, 1476, 1476
__codedoc_print_testfile_aux:n 1476, 1485, 1489
__codedoc_quote_special_char:N 2172, 2262, 2264, 2264
__codedoc_replace_at_at:N *37*, 111, 111, 652, 785, 1918, 1927
__codedoc_replace_at_at_aux:Nn 111, 115, 119
__codedoc_shorthand_meta: 1025, 1028, 1028
__codedoc_shorthand_meta:w 1028, 1029, 1030
__codedoc_show_functions_-defined: 1945, 1995
__codedoc_show_not_tested: 2010, 2065
\g__codedoc_show_notes_bool 35, 380, 381, 1630, 1640
__codedoc_signature_base_form:n 179, 179, 268
__codedoc_signature_base_form_-aux:n 179, 180, 181, 195
__codedoc_signature_base_form_-aux:w 179, 196, 198
__codedoc_special_index:nn 1121, 1425, 2108, 2131, 2131, 2143
__codedoc_special_index_-aux:nnnnnn 2144, 2150, 2169
__codedoc_special_index_-module:nnnnN 689, 2123, 2136, 2144, 2146, 2168
__codedoc_special_index_set:Nn 2144, 2173, 2230
__codedoc_split_function_auxi:w 206, 214, 219
__codedoc_split_function_-auxii:w 206, 221, 222
__codedoc_split_function_do:nn 206, 207, 209, 212, 227, 234, 273, 799
__codedoc_str_if_begin:nn 101, 109
__codedoc_str_if_begin:nnTF 101, 163, 166, 171
__codedoc_syntax:w . . 579, 1203, 1204
\g__codedoc_syntax_box *50*, 14, 1009, 1012, 1054, 1206, 1214, 1234
\l__codedoc_syntax_coffin *14*, 11, 1013, 1053, 1059, 1076, 1087
\l__codedoc_syntax_dim 1203, 1208, 1219
__codedoc_syntax_end: 581, 1203, 1224
__codedoc_target: 657, 1094, 1334, 2066, 2182
__codedoc_test_missing:n 1512, 1513, 1513
__codedoc_test_missing_aux:Nnn 1515, 1520, 1527
\g__codedoc_testfiles_seq 19, 1481, 1483
__codedoc_tmp:w 97, 98
__codedoc_tmpa:w . . . 82, 82, 210, 226, 2017, 2034, 2039, 2053, 2302, 2305
\l__codedoc_tmpa_int 42, 1595, 1597, 1599, 2058, 2061

\l__codedoc_tmpa_seq 42, 140, 141, 142, 897, 898, 899, 1362, 1364, 1544, 1548, 1568, 1572, 1574, 1577, 2212, 2213, 2215, 2336, 2338, 2341
\l__codedoc_tmpa_tl 39, 41, 45, 42, 232, 234, 701, 707, 710, 772, 776, 779, 781, 782, 783, 784, 785, 787, 840, 841, 844, 845, 898, 904, 1166, 1167, 1180, 1181, 1450, 1451, 1452, 1457, 1458, 1460, 1522, 1523, 1524, 1525, 1561, 1562, 1564, 1603, 1604, 1664, 1666, 1667, 1912, 1916, 1917, 1918, 1919, 1928, 1931, 1934, 2014, 2028, 2046, 2056, 2060, 2208, 2209, 2212, 2219, 2220, 2221, 2223, 2225, 2227
__codedoc_tmpb:w 82, 83, 2021, 2023, 2040, 2041
\l__codedoc_tmpb_tl 45, 42, 650, 651, 653, 705, 707, 852, 854, 899, 905, 906, 1925, 1926, 1927, 1928
\l__codedoc_trial_width_dim 14, 16, 1037, 1039, 1042, 1212
__codedoc_trim_right:Nn 95, 95, 100, 144, 146, 150, 151, 152
__codedoc_typeset_aux:n 729, 758, 1133, 1467
__codedoc_typeset_dates: 1100, 1185, 1185
\g__codedoc_typeset_documentation_bool 70, 365, 370, 540, 544, 549, 553
__codedoc_typeset_exp: 610, 729, 729, 1127
__codedoc_typeset_expandability: 1112, 1124, 1157
__codedoc_typeset_function_block:nN 873, 1105, 1105, 1117
__codedoc_typeset_functions: .. 50, 1038, 1091, 1091
\g__codedoc_typeset_implementation_bool 70, 366, 371, 542, 546, 556, 560, 1735, 1737, 1740, 1948, 2385
__codedoc_typeset_rexp: 619, 729, 736, 1128
__codedoc_typeset_TF: 628, 729, 743, 1111, 1148, 1154, 1471
__codedoc_typeset_variant_list:nN 1114, 1130
\l__codedoc_undoc_def_tl 1942, 1955, 1974, 1989
\l__codedoc_variants_prop 74
\g__codedoc_variants_seq 43, 47, 847, 851, 852, 1113, 1135, 1136, 1139, 1389, 1397
__codedoc_xmacro_code:n 76, 1889, 1905, 1910
__codedoc_xmacro_code:w 1889, 1914, 1922, 1936
\CodedocExplain 604
\CodedocExplainEXP 604
\CodedocExplainREXP 604
\CodedocExplainTF 604
\CodelineIndex 2388
CodelineNo 435
\CodelineNumbered 2393
coffin commands:
\coffin_clear:N 1011, 1013, 1014
\coffin_join:NnnNnnnn 1057, 1061, 1065, 1074, 1078, 1082
\coffin_new:N 10, 11, 12, 13
\coffin_typeset:Nnnnn 1069, 1086
\color 749, 753, 760
\columnwidth 1141, 1143
\comment 34, 551, 558, 1653
\ConTeXt 504
\cs 6–8, 13, 17, 37, 58, 81, 505, 526
cs commands:
\cs_generate_variant:Nn 84, 85, 100, 227, 698, 769, 869, 876, 1117, 1123, 1184, 1400, 1469, 1527, 1670, 2143, 2168
\cs_gset:Npe 1810
\cs_gset:Npn 429, 459, 466, 482, 1610, 1782, 1857, 1864, 1875, 2093

\cs_gset_eq:NN 441, 442, 1874
\cs_gset_protected:Npn 1826, 1832, 2105, 2119, 2202, 2204
\cs_if_exist:NTF 393, 731, 738, 745, 753, 1451, 1749, 1877
\cs_new:Npe 238
\cs_new:Npn 179, 181, 198, 200, 206, 208, 212, 219, 222, 258, 295, 307, 328, 329, 449, 521, 891, 1185
\cs_new_eq:NN . 82, 83, 458, 2200, 2201
\cs_new_protected:Npe 119, 2272
\cs_new_protected:Npn 95, 111, 129, 136, 228, 271, 276, 640, 699, 712, 714, 729, 736, 743, 758, 762, 770, 790, 797, 803, 821, 823, 833, 838, 857, 870, 892, 958, 968, 979, 991, 997, 1001, 1007, 1028, 1031, 1035, 1044, 1049, 1051, 1091, 1105, 1118, 1124, 1130, 1159, 1171, 1204, 1224, 1299, 1312, 1339, 1343, 1352, 1354, 1368, 1386, 1395, 1401, 1410, 1431, 1442, 1464, 1470, 1476, 1489, 1513, 1520, 1528, 1535, 1554, 1559, 1566, 1581, 1657, 1662, 1671, 1754, 1841, 1849, 1910, 1922, 1945, 1996, 2010, 2066, 2131, 2146, 2169, 2206, 2217, 2229, 2230, 2264, 2295, 2300, 2307, 2312, 2321
\cs_new_protected_nopar:Npn . . . 1030
\cs_set:Npe 1789
\cs_set:Npn 97, 1372, 1615, 1625, 1776, 1777, 2017, 2023, 2039, 2041, 2302
\cs_set_eq:NN . . 453, 525, 526, 527, 528, 1327, 1798, 1801, 1802, 1803, 1804, 1805, 1806, 1809, 1814, 1815, 1816, 1817, 1824, 1861, 1872, 1887
\cs_set_nopar:Npe 2073
\cs_set_nopar:Npn 2084
\cs_set_protected:Npn . 210, 443, 1903
cs-break (option) *6*
cs-break-nohyphen (option) *6*
\currentfile 1776, 1777, 1807, 1811, 1813, 1877, 1878, 1880
\CurrentOption 384

D

\d 962
danger (env.) 1609
\date 2416
\dbend 1610, 1616, 1626
ddanger (env.) 1609
\DeclareDocumentCommand 505, 507, 509, 515, 1500, 1510, 1511, 1734, 1736, 1738, 1751, 1761, 1769
\DeclareDocumentEnvironment 561, 572, 575
\DeclareExpandableDocumentCommand 450, 517, 519
\DeclareFontShape 425
\DeclareKeys 360
\DeclareMathOperator 2426
\DeclareRobustCommand . 535, 536, 537, 538
\DeclareUnknownKeyHandler 383
\def 609, 618, 627, 1678, 1679, 1686, 1687, 1703, 1713, 1726, 1727
\DeleteShortVerb 491, 492
\Describe *4*
\DescribeOption 1608
\DescribeRoutine *3*
\DescribeVariable *4*
\description 1685, 1702
\detokenize 1634
dim commands:
\dim_compare:nNnTF 1141
\dim_compare_p:nNn 1042
\dim_new:N 17, 1203
\dim_set:Nn 1004, 1039, 1208
\dim_zero:N 460, 1037
\ding 741
\DisableCrossrefs 2394
\DisableDocumentation *7*, 539
\DisableImplementation *7*, 539
doc commands:
\g_doc_functions_seq 4, 1120, 1956, 1958, 1972

\g_doc_macros_seq 4, 1421, 1957, 1960, 1970
\DocInclude 1771
\docincludeaux 1774, 1857
\DocInput 7, 13, 1761, 1813
docinput commands:
\g_docinput_clist . . 73, 3, 1765, 1770
\DocInputAgain 7, 1769
documentation (env.) 547
\DoNotIndex 59, 1365, 1405, 1412

E

\edef 720
\else 1706, 1821
\emph 1557, 1634, 1644, 1649
\EnableCrossrefs 2389
\EnableDocumentation 6, 539
\EnableImplementation 6, 539
\encapchar 1845, 1853, 2266
\end 1102, 1226, 1227, 1235, 1619, 1628, 1905
\endcomment 34, 553, 560, 1653
\enddescription 1689, 1715
\endenumerate 602
\endgraf 586, 1615, 1625
\endgroup 479, 1615, 1625, 1689, 1715, 1727, 1830
\endinput 1747
\endtabbing 1727
\endtheglossary 1806, 1817
\endtrivlist 1530
\endVerbatim 442
\endverbatim 440, 1648
\enquote 631
\ensuremath 713, 716, 727
\enumerate 595
\env 8, 535
environments:
arguments 11, 593
danger 1609
ddanger 1609
documentation 547
function 8, 572
implementation 547
macro 10, 572
NOTE 1640
syntax 8, 578
texnote 10, 584
variable 8, 11, 561
\epTeX 494
\errorstopmode 2059
\eTeX 494
\eupTeX 494
\evensidemargin 457
exp commands:
\exp_after:wN 98, 104, 105, 214, 884, 2305
\exp_args 265
\exp_args:Nc 822, 2426
\exp_args:Ne 810, 1406, 1604, 2241, 2323
\exp_args:Nee 445
\exp_args:NNe 786, 900
\exp_args:NNNo 1427
\exp_args:NNo 115
\exp_args:NNV 1361
\exp_args:No 226, 683, 971, 1167, 1181, 2318, 2319, 2324, 2351
\exp_args:NV 447, 710, 1365
\exp_last_unbraced 266
\exp_last_unbraced:NNNNo 890
\exp_last_unbraced:NNo 1452
\exp_last_unbraced:NV 1666, 1667
\exp_not:N 240, 242, 244, 248, 254, 256, 1676, 2076, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2287, 2288, 2290, 2291
\exp_not:n 22, 97, 199, 260, 267, 285, 681, 1177, 1447, 1676, 2020, 2077, 2084, 2088, 2150, 2210, 2224, 2227, 2303
\ExplFileDate 1882, 2416
\ExplFileVersion 1883
\ExplMakeTitle 28, 2404

F

\fi 480, 1710, 1796, 1823

fi commands:
\fi: ... 1794
\file ... 8, 535
file commands:
\g_file_curr_name_str ... 1518
\file_if_exist:nTF ... 390
\file_input:n ... 392
\filekey ... 1810, 1811, 1861, 1869
\filename ... 1881
\filesep ... 1837, 1840, 1854, 1860
\Finale ... 1734
\font ... 721, 722, 1609, 2373
\fontfamily ... 1474
\fontseries ... 1474
\foo ... 7
\footnote ... 1634, 1678, 1679
\footnotemark ... 1673
\footnotesize ... 1491, 1557, 1868
\footnotetext ... 1676
full (option) ... 5
function (env.) ... 8, 572
\fvset ... 440

G

\GetFileInfo ... 1880
\GlossaryParms ... 2373
\GlossaryPrologue ... 2370
group commands:
\group_begin: ... 134, 423, 896, 1889, 1907, 2069, 2075, 2086, 2196
\group_end: ... 160, 426, 901, 1902, 1938, 2071, 2078, 2089, 2199

H

\hangafter ... 1614, 1624
\hangindent ... 1614, 1616, 1624, 1626
\hbox ... 1436, 1616, 1626, 1708
hbox commands:
\hbox:n ... 1331
\hbox_gset:Nw ... 1214
\hbox_gset_end: ... 1229
\hbox_set:Nn ... 1138
\hbox_set:Nw ... 1422
\hbox_set_end: ... 1427
\hbox_unpack_drop:N ... 1146, 1152, 1376, 1423
hcoffin commands:
\hcoffin_set:Nn ... 1038, 1053
\hdclindex ... 2196
\hdpindex ... 2196
\hfil ... 478
\hfill ... 1616, 1626, 1708, 1885
hide-notes (option) ... 6
\hologo 494, 495, 497, 498, 499, 500, 501, 504
hook commands:
\hook_gput_next_code:nn ... 1670, 1675
\href ... 2413
\hskip ... 477, 1616, 1626, 1707, 1709
\hspace ... 1096
\hss ... 478
\hyperlink ... 732, 739, 746
\hyperref ... 1452
\hypertarget ... 608, 617, 626
\hyphenchar ... 721, 722, 2373

I

if commands:
\if_meaning:w ... 1792
\IfFileExists ... 1775
\ifnum ... 468
\ifx ... 1706
\ignorespaces ... 1047, 1508, 1616, 1626
\ignorespacesafterend ... 582
\immediate 1785, 1799, 1800, 1820, 1834, 1851
implementation (env.) ... 547
\include ... 1779
\indexentry ... 1829, 1836, 1845, 1853
\IndexPrologue ... 2095
\IniTeX ... 494
\input ... 1758
\InstanceKey ... 1726
\InstanceSemantics ... 1727
int commands:
\int_compare:nNnTF ... 88, 309, 311, 313, 317, 322, 872, 1334, 1532, 1597, 1599, 1778

\int_compare:nTF ... 1135, 1455, 1572
\int_eval:n ... 1336
\int_gdecr:N ... 1426
\int_gincr:N ... 1424
\int_incr:N ... 1314, 1440
\int_new:N ... 18, 44, 45, 63, 81
\int_set:Nn ... 1595, 2058, 2061
\int_use:N 1095, 1660, 1676, 1837, 1854
iow commands:
\iow_char:N ... 780, 782, 783
\iow_close:N ... 1991
\iow_indent:n ... 2114, 2380
\iow_new:N ... 1939
\iow_newline: ... 1950, 1963, 1967, 1975, 1980, 2002, 2003, 2019, 2025, 2030, 2032, 2043, 2048, 2049, 2051
\iow_now:Nn ... 2000
\iow_open:Nn ... 1951
\iow_term:n ... *79*, 1950, 1992
\item ... 1384, 1613, 1623, 1686, 1705
\itemindent ... 462
\itshape ... 750, 754

K

\kern ... 500, 501, 754, 1626
kernel (option) ... *5*
kernel internal commands:
__kernel_tl_set:Nn ... 98, 233, 240, 661, 680, 686, 764, 772, 1363, 2232, 2250, 2275, 2297, 2304
keys commands:
\keys_define:nn ... 633, 911, 1238
\keys_set:nn ... 646, 984, 1303
\kill ... 1731

L

\label ... 1167, 1181
\langle ... 716
\language ... 723
\LaTeX ... 2401, 2412
\LaTeXe ... 2157
\leavevmode ... 475
\leftskip ... 476, 477
\levelchar ... 2186, 2266
\list ... 458
\listparindent ... 460
\llap ... 1374, 1436
lm-default (option) ... *5*
\LoadClass ... 397
\Lua ... 494
\LuaTeX ... 494

M

macro (env.) ... *10*, 572
\MacroFont ... 1437, 1456
\MacroLongFont ... 1456, 1472
\MacroTopsep ... 1370
\makebox ... 751
\makelabel ... 1372
\MakePercentComment ... 1759
\MakePercentIgnore ... 1757
\MakePrivateLetters ... 429, 1359, 2092
\MakeShortVerb ... 486, 487
\maketitle ... 2417
\manual ... 1609, 1610
\marg ... *8*, *33*, 530
\marginparsep ... 1064, 1068, 1085
\marginparwidth ... 455, 1042, 1068, 1212
\markboth ... 2098, 2374
\MaybeStop ... *72*, 1734
\mbox ... 655, 733, 740, 747
\medskipamount ... 1068, 1081
\meta ... *8*, 515, 528, 531, 533, 534, 1030
\midrule ... 1190
mode commands:
\mode_if_math:TF ... 655, 717, 1029
\mode_leave_vertical: ... 1233, 2068
msg commands:
\msg_error:nn ... 1010, 1207
\msg_error:nnn ... 817, 964
\msg_error:nnnn ... 974
\msg_info:nn ... 394, 2118, 2384
\msg_new:nnn ... 338, 340, 342, 347, 352, 354, 388, 2111, 2377
\msg_new:nnnn ... 330
\msg_warning:nnnn ... 1243

\msg_warning:nnnnn 153
\multicolumn 1193, 1199

N

\n 2435, 2437, 2438, 2439, 2447, 2450
\nan 2428
\NB *6*, *8*, 1630
\newcommand 530
\NewDescribeEnvironment *4*
\NewDocElement 1608
\NewDocumentCommand 539, 541, 543, 545, 604, 606, 615, 624, 1632, 1638, 1771, 2404, 2428
\NewDocumentEnvironment 547, 554, 578, 584, 593, 1642, 1653
\newenvironment 1611, 1621, 1680, 1696, 1720
\NewMacroEnvironment *4*
\nobreak 478, 1556, 1709
\noindent 999, 1047, 1556, 1613, 1623, 1644, 1649
\nolinebreak 1139
\nolinkurl 535
\normalfont 1103
\normalsize 1103
NOTE (env.) 1640
\NOTE *6*, *8*

O

\oarg *8*, 530
\obeylines 1222
\obeyspaces 1221
\oddsidemargin 456
\OnlyDescription 1734, 2395
onlydoc (option) *5*
\OnlyDocumentation *6*
\openout 1799
options:
check *6*
checktest *6*
cs-break *6*
cs-break-nohyphen *6*
full *5*
hide-notes *6*
kernel *5*
lm-default *5*
onlydoc *5*
show-notes *6*

P

package commands:
\package_function_one:N *9*
\package_function_two:n *9*
\pageref 1604
\par 478, 999, 1003, 1484, 1498, 1502, 1508, 1557, 1615, 1619, 1625, 1628, 1644, 1649, 1686, 1694, 1711, 1719, 1733
\parbox 1143, 1866
\parfillskip 474, 1709
\parg *8*, 530
\parindent 461, 473
\parskip 463
\part 1807, 2097, 2372
\partname 464
\PassOptionsToClass 384, 387
\PassOptionsToPackage 379
\pdfstringdefDisableCommands . 452, 523
\pdfstringnewline 449
\pdfTeX 494
\penalty 1707, 1708
\phantom 754
\pkg *8*, 535, 2161, 2408
\prevdepth 1004
prg commands:
\prg_break: 242, 244, 248, 254
\prg_break_point: 256
\prg_generate_conditional_variant:Nnn 94, 109
\prg_new_conditional:Npnn . 293, 877
\prg_new_protected_conditional:Npnn 86, 101, 161
\prg_return_false: 91, 107, 173, 175, 314, 318, 323, 887
\prg_return_true: 92, 106, 164, 167, 172, 314, 318, 323, 880, 887

\PrintChanges 1803, 1804, 1815, 2383
\PrintIndex 1801, 1802, 1814, 2117
\ProcessKeyOptions 396
prop commands:
\prop_get:NnN 1664
\prop_get:NnNTF 1522
\prop_gput:Nnn 1659
\prop_if_empty:NTF 2015
\prop_map_function:NN 2033
\prop_new:N 20, 75, 1656
\prop_put:Nnn 1525
\providecommand 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 532, 533, 534, 1472, 1655
\pTeX 494

Q

quark commands:
\q_mark 215, 216, 220, 221, 223, 294, 296
\q_no_value 33, 644, 645, 1024, 1033, 1297, 1325, 1341
\quark_if_no_value:NTF 678, 684, 2134
\quark_if_recursion_tail_stop_do:nn 1930
\q_recursion_stop 1914
\q_recursion_tail 1914, 1929
\q_stop 97, 98, 180, 198, 217, 220, 221, 223, 294, 296, 326, 2302, 2305
\quotechar 2266, 2269

R

\raggedbottom 418
\raggedright 1145, 1220, 1869
\raisebox 608, 617, 626
\rangle 727
\RecordChanges 2387
regex commands:
\regex_replace_all:nnN 664
\regex_replace_once:nnNTF 961
\relax 1709, 1773, 1800, 1802, 1804, 1809, 1874
\RenewCommandCopy 1750
\RequirePackage . . 398, 399, 421, 422, 428
\rightskip 473

S

\sb 713
scan commands:
\scan_stop: *15*, *42*, 104, 105, 808, 818, 845, 905, 1609, 2197
\scriptsize 1194, 1200
seq commands:
\seq_clear:N 775
\seq_clear_new:N 828
\seq_count:N 1135, 1572, 1596
\seq_gclear:N 847, 1606
\seq_get:NN 844, 899
\seq_get_left:NN 2338
\seq_get_right:NN 2341
\seq_gpop:NN 852
\seq_gput_right:Nn 1120, 1348, 1353, 1421, 1483, 1546
\seq_gremove_duplicates:N 1956, 1957
\seq_gset_eq:NN 851
\seq_gset_filter:NNn 1583
\seq_if_empty:NTF 1113, 1389, 1593, 2037
\seq_if_in:NnTF 1481, 1960, 1972
\seq_item:Nn 1577, 1602
\seq_map_function:NN . . . *57*, 793, 2052
\seq_map_inline:Nn . 1397, 1958, 1970
\seq_map_variable:NNn 142
\seq_new:N 4, 5, 21, 22, 47, 49, 50
\seq_pop:NN 840, 898
\seq_pop_left:NN 141
\seq_put_right:Nn 776, 829, 831
\seq_set_eq:NN 897
\seq_set_filter:NNn 1544
\seq_set_from_clist:Nn . *59*, 786, 1361
\seq_set_map:NNn 1568, 2213
\seq_set_split:Nnn 84, 84, 140, 2212, 2336
\seq_use:Nn *59*, 1136, 1139, 1364, 1517, 1548, 2215
\seq_use:Nnnn *66*, 1574
\setcounter 435, 1667, 2094

\SetKeys 385
\setlength 454, 461, 462, 463, 471
\sffamily 439
show-notes (option) *6*
\small 588, 1093, 1215, 1474
\space 1779, 2020, 2026, 2044
\SpecialIndex 2105
\star 734
\StopEventually *5*, *6*, 1734
str commands:
\c_backslash_str *19*, 446, 508, 513, 681, 766, 2246, 2259, 2282
\c_percent_str 780
\str_case:nn 2326
\str_case:nnTF 183, 2177
\str_count:n 1455
\str_head:N 2326
\str_if_eq:nnTF . . . 845, 905, 2154, 2174
\str_if_eq_p:nn 265, 266
\str_tail:n 2324
\c_underscore_str 139, 765, 1563
\string 609, 618, 627, 1779, 1785, 1829, 1836, 1845, 1853, 2157, 2161
\strut 1333, 1444, 1868
\subsection 1682, 1698
\subsubsection 1690, 1699, 1716, 1722, 1723, 1729
syntax (env.) *8*, 578
sys commands:
\c_sys_day_int 972
\c_sys_jobname_str . 1951, 2115, 2381
\c_sys_month_int 972
\c_sys_year_int 972

T

\tabbing 1730
\tabcolsep 1096
\Team 2399
\TemplateArgument 1686
\TemplateKey 1703, 1706
\TemplateSemantics 1687, 1713
\testfile 1327, 1510
\TestFiles *11*
\TestFiles *11*, 1500
\TestMissing *11*
\TestMissing *11*, 1511
\TeX 588, 612, 2157
TEX and LATEX 2ε commands:
\@ *37*, 431, 672, 1578
\@M 1684, 1701, 1725
\@addtoreset 436, 465
\@auxout 609, 618, 627, 1778, 1798, 1824
\@beginparpenalty . . 1684, 1701, 1725
\@bsphack 1742, 2107
\@currenvir 2174
\@docinclude 1780, 1782
\@dottedtocline 483
\@eha 1779
\@esphack 1745, 1830, 2109
\@evenfoot 1872, 1887
\@indexfile 1828, 1834, 1843, 1851
\@input 1785
\@latexerr 362, 1779
\@ltxdoc@PrintChanges 1803, 1815
\@ltxdoc@PrintIndex 1801, 1814
\@ltxdoc@endtheglossary . . 1806, 1817
\@ltxdoc@theglossary 1805, 1816
\@mainaux 1785, 1824
\@nameuse 1822
\@oddfoot 1864, 1872, 1875, 1887
\@partaux 1778, 1798, 1799, 1800, 1820
\@partlist 1790
\@plus 470
\@pnumwidth 473, 474, 478
\@secpenalty 469
\@starttoc *81*, 2084, 2088
\@tempa 1792
\@tempb 1789, 1792
\@tempdima 471, 476
\@tempswafalse 1788
\@tempswatrue 1786, 1793
\@wrindex *74*, 1826
\@writeckpt 1819
\@xobeysp *37*, 670
_ *66*

\bottomrule *43*, *52*
\c@CodelineNo 81, 1424, 1426, 1837, 1854
\c@footnote 1660, 1676
\c@HD@hypercount . . . 1095, 1853, 1854
\c@tocdepth 468
\check@checksum 1743
\Codedoc@explTF 627, 745, 753
\Codedoc@expstar 609, 731
\Codedoc@rexpstar 618, 738
\codeline@wrindex *74*, 1832
\DocInput *73*
\DoNotIndex *30*
\endtrivlist *61*
\expanded@notin 445
\g@addto@macro 2092
\hb@xt@ 478
\HD@savedestfalse 2070, 2076
\HD@target 2070
\HDorg@theCodelineNo 438
\Hy@footnote@currentHref 1655, 1660, 1665
\Hy@MakeCurrentHref 1095
\if@partsw 1787
\if@tempswa 1797
\ifnot@excluded 443
\include *73*
\index@excludelist *30*, 447
\index@prologue 1862
\init@checksum 1744
\input *73*
\it@is@a *83*, 2119
\l@nohyphenation 723
\l@section 465
\l@subsection 465
\m@ne 722
\macro@namepart *30*, 446
\meta *39*
\meta@font@select 719
\meta@hyphen@restore 720, 725
\midrule *52*
\nfss@text 717
\p@ 470
\part *5*, *82*, *91*
\partname *5*
\protected@write 1828, 1843
\saved@indexname *61*, 1428
\saved@macroname *61*, 1403
\texttt *65*, *66*
\toprule *52*
\trivlist *61*
\verb *84*
\verbatim@font *37*, *84*, 658, 1934, 2190
\xmacro@code *76*, 1889
\z@ 468, 473
tex commands:
\tex_interactionmode:D . . . 2058, 2061
\tex_lowercase:D 1900
texnote (env.) *10*, 584
\text 1934, 2428
\textbackslash 520
\textbf 588, 1726
\textcolor 439
\textit 1493, 1503
\textrm 1139, 1140, 1147, 1153
\textsf 537, 538, 1496, 1506
\texttt 531, 533, 534, 536, 597, 611, 612, 613, 621, 622, 628, 629, 1564, 1644, 2428
\textwidth 454, 1046, 1210, 1866
\thanks 2401, 2412
\the 721, 1853, 1854
\theCodelineNo 437, 2073, 2077
\theglossary 1805, 1816
\theindex 2092
\thepage 1829, 1846, 1885
\thepart 1811, 1859, 1860, 1878, 1881
\tiny 439
\title 2406
tl commands:
\c_catcode_active_space_tl 145
\c_catcode_other_space_tl 843, 1459
\c_space_tl 1518, 2190, 2260
\tl_clear:N . 792, 1326, 1912, 1953, 1954, 1955, 2007, 2014, 2277, 2344

\tl_const:Nn 57, 59, 1943, 1944, 2198, 2399
\tl_count:n 89, 90
\tl_gclear:N 1756, 1840
\tl_gput_right:Nn . . 1862, 2117, 2383
\tl_greplace_all:Nnn 842
\tl_gset:Nn 464, 1665, 1743, 1933
\tl_gset_eq:NN 841
\tl_if_empty:NTF *9*, *10*, 113, 1191, 1197, 1345, 1356, 1998, 2056, 2334
\tl_if_empty:nTF 299, 300, 301, 303, 304, 305, 358, 882, 2151, 2185
\tl_if_empty_p:N . . . *9*, *10*, 1188, 1189
\tl_if_eq:nnTF 1479
\tl_if_head_eq_charcode:nNTF 280, 807, 2281, 2309, 2314, 2323
\tl_if_in:NnTF 241, 825, 2221, 2278, 2284, 2287, 2290
\tl_if_in:nnTF 103, 243, 245, 250, 1913
\tl_map_function:NN 835
\tl_map_inline:Nn *43*, 894
\tl_map_inline:nn . . . 702, 2241, 2266
\tl_new:N 8, 9, 32, 34, 42, 43, 46, 51, 52, 53, 54, 56, 64, 65, 66, 74, 78, 79, 80, 1753, 1940, 1941, 1942, 2144, 2145
\tl_put_left:Nn 767
\tl_put_right:Nn 247, 252, 827, 1523, 1928, 1934, 1962, 1966, 1974, 2028, 2046
\tl_remove_all:Nn 766, 779, 781, 782, 783
\tl_replace_all:Nnn 121, 122, 123, 124, 125, 126, 127, 139, 651, 653, 670, 707, 765, 1458, 1562, 2209, 2220, 2223, 2225, 2235, 2244, 2258, 2268
\tl_replace_once:Nnn 253, 437
\tl_set:Nn 33, 236, 644, 645, 647, 701, 960, 965, 983, 1024, 1033, 1302, 1325, 1341, 1403, 1428, 1457, 1524, 1561, 1859, 1860, 1916, 1925, 2171, 2208, 2219, 2257, 2291, 2328, 2329
\tl_set_eq:NN 138, 149, 2135, 2317, 2350
\tl_set_rescan:Nnn *59*, 650, 705, 1358
\tl_tail:N 2298
\tl_to_str:N 154, 155, 156, 446, 661, 687, 787, 975, 1349, 2242, 2269, 2276
\tl_to_str:n *20*, *23*, 84, 85, 90, 104, 105, 203, 215, 240, 241, 243, 245, 247, 250, 253, 328, 329, 447, 520, 522, 764, 772, 781, 885, 891, 1108, 1178, 1407, 1448, 2232, 2235, 2278, 2357
\tl_use:N 1752, 1919, 1931
\tn *8*, *17*, *37*, *81*, 505, 527, 611, 612
token commands:
\token_to_str:N 121, 122, 123, 127, 152, 226, 263, 506, 518, 1165, 2190, 2238, 2239, 2242, 2247, 2252, 2260, 2284, 2287, 2290, 2318, 2319, 2337, 2351
\toprule 1097, 1218
\topsep 1370
\trivlist 1371
\ttfamily 424, 1093, 1215, 1634, 1809, 1811, 1878, 1881, 2373

U

\underline 754
\UnitTest *6*
\UnitTested *11*
\UnitTested *11*, 1510
\upTeX 494
\url 2402
use commands:
\use:n 283, 1453, 2082, 2121, 2148, 2227
\use_i:nn 1666
\use_i:nnn 207
\use_ii:nn 1667
\use_ii:nnn 209
\use_none:n 325, 1861
\use_none:nn 2220

V

\varepsilon 502, 503

variable (env.) 8, 11, 561
vbox commands:
\vbox_set:Nn 1329, 1433
\vbox_unpack_drop:N 1379, 1435
vcoffin commands:
\vcoffin_set:Nnw 1046
\vcoffin_set_end: 1050
\verb 2239, 2247, 2252
\Verbatim 441
\verbatim 440, 1645
\verbatimchar 2093, 2237, 2239, 2242, 2246, 2247, 2252, 2253
\vskip 1336, 1728
\vspace 587, 591
\vss 1380
\vtop 1377

W

\write 609, 618, 627, 1785, 1800, 1834, 1851

X

\XeTeX 494

Z

\Z 962