The l3doc class

The LaTeX3 Project[*]

Released 2019-11-07

Contents

[*] https://www.latex-project.org/latex3/

1 Introduction

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

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 xparse 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 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 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`/`\DisableImplementation` 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.

4.3 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 sequencies entirely, use `cs-break-off`.

```
\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.

4.4 Describing functions in the documentation

`function` `syntax` Two heavily-used environments are defined to describe the syntax of expl3 functions and variables.

```
\begin{function}{\function_one:, \function_two:}
  \begin{syntax}
    |\foo_bar:| \Arg{meta} \meta{test_1}
  \end{syntax}
\meta{description}
\end{function}
```

```
\function_one:    \foo_bar: {⟨meta⟩} ⟨test1⟩
\function_two:
                  ⟨description⟩
```

Function environments take an optional argument to indicate whether the function(s) it describes are expandable or restricted-expandable or defined in conditional forms. Use `EXP`, `rEXP`, `TF`, `pTF`, or `noTF` for this; 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`. As an example:

```
\begin{function}[pTF]{\cs_if_exist:N}
  \begin{syntax}
    \cs{cs_if_exist_p:N} \meta{cs}
  \end{syntax}
\meta{description}
\end{function}
```

```
\cs_if_exist_p:N ⋆    \cs_if_exist_p:N ⟨cs⟩
\cs_if_exist:NTF ⋆
                      ⟨description⟩
```

`variable` If you are documenting a variable instead of a function, use the `variable` environment instead; it behaves identically to the `function` environment above.

`texnote` This environment is used to call out sections within `function` and similar that are only of interest to seasoned TeX developers.

4.5 Describing functions in the implementation

macro 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`).

\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 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 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}
```

4.6 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.

4.7 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

```
⟨*class⟩
⟨@@=codedoc⟩
```

5.1 Variables

\g_docinput_clist

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

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

(*End 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.

```
\seq_new:N \g_doc_functions_seq
\seq_new:N \g_doc_macros_seq
```

(*End 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.

```
\bool_new:N \l__codedoc_detect_internals_bool
\bool_set_true:N \l__codedoc_detect_internals_bool
\tl_new:N \l__codedoc_detect_internals_tl
\tl_new:N \l__codedoc_detect_internals_cs_tl
```

(*End 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.

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

(*End 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.

```
\coffin_new:N \l__codedoc_functions_coffin
\coffin_new:N \l__codedoc_descr_coffin
\coffin_new:N \l__codedoc_syntax_coffin
```

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

\l__codedoc_macro_internal_set_bool
\l__codedoc_macro_internal_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_internal_set_bool
\bool_new:N \l__codedoc_macro_internal_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 definition for* `\l__codedoc_macro_internal_set_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 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 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 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 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 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 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 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 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 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 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 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 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 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 definition for* `\g__codedoc_typeset_documentation_bool` *and* `\g__codedoc_typeset_implementation_bool`.)

`\g__codedoc_base_name_tl`
`\l__codedoc_variants_prop`

The name of the macro which is being documented (without its signature), and a 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 definition for* `\g__codedoc_base_name_tl` *and* `\l__codedoc_variants_prop`.)

\l__codedoc_function_label_clist
\l__codedoc_no_label_bool

Option of a `function` environment which replaces the label that would normally be 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 definition for* \l__codedoc_function_label_clist *and* \l__codedoc_no_label_bool.)

\l__codedoc_date_added_tl
\l__codedoc_date_updated_tl

Values of some options of the `function` environment.

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

(*End 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 definition for* \l__codedoc_macro_argument_tl.)

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

5.2 Variants and helpers

__codedoc_tmpa:w
__codedoc_tmpb:w

Auxiliary macros for temporary use.

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

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

\seq_set_split:NoV
\str_case:fn
\tl_count:f
\tl_greplace_all:Nxn
\tl_greplace_all:Nno
\tl_if_head_eq_charcode:oNTF
\tl_if_head_eq_charcode:oNT
\tl_if_head_eq_charcode:oNF
\tl_if_head_eq_meaning:VNF
\tl_if_in:noTF
\tl_if_in:ooTF
\tl_if_in:NoTF
\tl_if_in:NoT
\tl_if_in:NoF
\tl_remove_all:Nx
\tl_replace_all:Nxn
\tl_replace_all:Nnx
\tl_replace_all:Non
\tl_replace_all:Nno
\tl_replace_once:Noo
\tl_to_str:f
\tl_to_str:o
\prop_get:NxNTF
\prop_put:Nxn
\prop_gput:NVx

A few missing variants.

```
\cs_generate_variant:Nn \seq_set_split:Nnn { NoV }
\cs_generate_variant:Nn \seq_gput_right:Nn { Nf }
\cs_generate_variant:Nn \str_case:nn { fn }
\cs_generate_variant:Nn \tl_count:n { f }
\cs_generate_variant:Nn \tl_greplace_all:Nnn { Nx , Nno }
\cs_generate_variant:Nn \tl_if_empty:nTF { f }
\cs_generate_variant:Nn \tl_if_head_eq_charcode:nNTF { o }
\cs_generate_variant:Nn \tl_if_head_eq_charcode:nNT  { o }
\cs_generate_variant:Nn \tl_if_head_eq_charcode:nNF  { o }
\cs_generate_variant:Nn \tl_if_head_eq_meaning:nNF  { V }
\cs_generate_variant:Nn \tl_if_in:nnTF { no , oo }
\cs_generate_variant:Nn \tl_if_in:NnTF { No }
\cs_generate_variant:Nn \tl_if_in:NnT  { No }
\cs_generate_variant:Nn \tl_if_in:NnF  { No }
\cs_generate_variant:Nn \tl_remove_all:Nn   { Nx }
\cs_generate_variant:Nn \tl_replace_all:Nnn { Nx , Nnx, No , Nno }
\cs_generate_variant:Nn \tl_replace_once:Nnn { Noo }
\cs_generate_variant:Nn \tl_set_rescan:Nnn { NnV }
\cs_generate_variant:Nn \tl_to_str:n { f , o }
\cs_generate_variant:Nn \prop_get:NnNTF { Nx }
\cs_generate_variant:Nn \prop_put:Nnn { Nx }
\cs_generate_variant:Nn \prop_gput:Nnn { NVx }
```

(*End definition for* \seq_set_split:NoV *and others. These functions are documented on page* **??**.)

__codedoc_if_almost_str:n*TF*

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 are 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:f { \tl_to_str:f {#1} } }
      { \prg_return_false: }
      { \prg_return_true: }
  }
\cs_generate_variant:Nn \__codedoc_if_almost_str:nT { V }
```

(*End 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} }
    \tl_set:Nx #1 { \exp_after:wN \__codedoc_tmp:w #1 #2 \q_stop }
  }
\cs_generate_variant:Nn \__codedoc_trim_right:Nn { No }
```

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

__codedoc_str_if_begin:nn*TF*
__codedoc_str_if_begin:oo*TF*

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 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:Npx \__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 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:Non \l__codedoc_detect_internals_tl { \token_to_str:N _ } { _ }
      \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:nnxxx { 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 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 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 x-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 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 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
        \tl_set:Nx #2
          { \__codedoc_split_function_do:on \l__codedoc_tmpa_tl { \__codedoc_base_form_aux:n
      }
      { \tl_set:Nn #2 {#1} }
  }
\cs_new:Npx \__codedoc_key_get_base_TF:nN #1#2
  {
    \tl_set:Nx #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 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:x
              {
                \exp_not:n {#3}
                { \__codedoc_base_form_aux:nnN {#4} {#5} #6 }
              }
                {#4} {#5}
          }
      }
      { #2 {#1} }
  }
```

(*End 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 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 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 } { deprecated-function }
  { The~deprecated~function(s)~’#1’~should~have~been~removed~on~#2. }
\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

```
\DeclareOption { a5paper } { \@latexerr { Option~not~supported } { } }
```

```
\DeclareOption { full }
  {
    \bool_gset_true:N \g__codedoc_typeset_documentation_bool
    \bool_gset_true:N \g__codedoc_typeset_implementation_bool
  }
\DeclareOption { onlydoc }
  {
    \bool_gset_true:N \g__codedoc_typeset_documentation_bool
    \bool_gset_false:N \g__codedoc_typeset_implementation_bool
  }
```

```
\DeclareOption { check }
  { \bool_gset_true:N \g__codedoc_checkfunc_bool }
\DeclareOption { nocheck }
  { \bool_gset_false:N \g__codedoc_checkfunc_bool }
```

```
\DeclareOption { checktest }
  { \bool_gset_true:N \g__codedoc_checktest_bool }
\DeclareOption { nochecktest }
  { \bool_gset_false:N \g__codedoc_checktest_bool }
```

```
\DeclareOption { kernel }
  { \bool_gset_true:N \g__codedoc_kernel_bool }
\DeclareOption { stdmodule }
  { \bool_gset_false:N \g__codedoc_kernel_bool }
```

```
\DeclareOption { cm-default }
  { \bool_gset_false:N \g__codedoc_lmodern_bool }
\DeclareOption { lm-default }
  { \bool_gset_true:N \g__codedoc_lmodern_bool }
```

```
\DeclareOption { cs-break-off }
  { \bool_gset_false:N \g__codedoc_cs_break_bool }
\DeclareOption { cs-break-nohyphen }
  { \PassOptionsToPackage{nohyphen}{underscore} }
```

```
\DeclareOption { show-notes }
  { \bool_gset_true:N  \g__codedoc_show_notes_bool }
\DeclareOption { hide-notes }
  { \bool_gset_false:N \g__codedoc_show_notes_bool }
```

```
\DeclareOption* { \PassOptionsToClass { \CurrentOption } { article } }
\ExecuteOptions { full, kernel, nocheck, nochecktest, 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 } }
  }
```

```
\ProcessOptions
```

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 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 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 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 somewhat inefficient as we could have ensured that \index@excludelist has string catcodes in the first place.

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

(*End 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 xparse 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 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
__codedoc_oldlist:nn

Customise lists.

```
\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 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 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 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 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 definition for* `\cmd`*,* `\cs`*, and* `\tn`*. These functions are documented on page 5.*)

`\meta` A document-level command.

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

(*End definition for* `\meta`*. This function is documented on page 5.*)

`\__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 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 definition for* `\Arg` *and others. These functions are documented on page 5.*)

\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 definition for* `\file` *and others. These functions are documented on page 5.*)

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

documentation
implementation

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

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
macro

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

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

Used to describe information destined to TEX experts only.

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

arguments

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{x}-type~argument~(in~plain~
    \TeX{}~terms,~inside~an~\cs{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~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 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:Non \l__codedoc_cmd_tl \l__codedoc_tmpb_tl { _ }
      \__codedoc_replace_at_at:N \l__codedoc_cmd_tl
      \tl_replace_all:Nno \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 }
    {
      \verbatim@font
      \__codedoc_if_almost_str:VT \l__codedoc_cmd_tl
        {
          \tl_set:Nx \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:NF \l__codedoc_cmd_noindex_bool
    {
      \quark_if_no_value:NF \l__codedoc_cmd_index_tl
        {
          \tl_set:Nx \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
        {
          \tl_set:Nx \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 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:Non \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 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 definition for* `\__codedoc_typeset_exp:` *and others.*)

__codedoc_get_hyper_target:nN
__codedoc_get_hyper_target:oN
__codedoc_get_hyper_target:xN

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
  {
    \tl_set:Nx #2 { \tl_to_str:n {#1} }
    \tl_replace_all:Nxn #2 { \c_underscore_str } { / }
    \tl_remove_all:Nx   #2 { \c_backslash_str }
    \tl_put_left:Nn #2 { doc/function// }
  }
\cs_generate_variant:Nn \__codedoc_get_hyper_target:nN { o , x }
```

(*End 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
  {
    \tl_set:Nx \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:Nx \l__codedoc_tmpa_tl
          { \iow_char:N \^^M \c_percent_str }
        \tl_remove_all:Nx \l__codedoc_tmpa_tl { \tl_to_str:n { ^ ^ A } }
        \tl_remove_all:Nx \l__codedoc_tmpa_tl { \iow_char:N \^^I }
        \tl_remove_all:Nx \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:NNx \seq_set_from_clist:Nn #2
          { \tl_to_str:N \l__codedoc_tmpa_tl }
      }
  }
```

(*End 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:Nx \__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:nnx { 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 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 to `\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:Nno \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:x { \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:x
          { \g__codedoc_base_name_tl : \l__codedoc_tmpb_tl }
      }
  }
```

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

__codedoc_names_typeset_auxii:n
__codedoc_names_typeset_auxii:x

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:xN
          { \__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 { x }
```

(*End definition for __codedoc_names_typeset_auxii:n.*)

__codedoc_names_typeset_block:nN
__codedoc_names_typeset_block:xN

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 { x }
```

(*End 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_set_bool
      {
        \bool_if:NTF \l__codedoc_macro_internal_bool
          { \prg_return_true: } { \prg_return_false: }
      }
      {
        \tl_if_empty:fTF
          {
            \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 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:NNx
        \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 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 .code:n = { \__codedoc_deprecated_on:n {#1} } ,
    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:nnxx { l3doc } { future-date }
          { \tl_to_str:N \l__codedoc_macro_argument_tl }
          {#1}
      }
  }
```

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

__codedoc_deprecated_on:n

The date comparison function expects two dates in the YYYY-MM-DD format (- is not subtraction here). Complain if a deprecated function should have been removed earlier. In any case, mark it as internal to suppress the text “documented on page . . . ”.

```
\cs_new_protected:Npn \__codedoc_deprecated_on:n #1
```

```
  {
    \__codedoc_date_set:Nn \l__codedoc_tmpa_tl {#1}
    \exp_args:No \__codedoc_date_compare:nNnT
      { \l__codedoc_tmpa_tl } < { \c_sys_year_int - \c_sys_month_int - \c_sys_day_int }
      {
        \msg_error:nnxx { l3doc } { deprecated-function }
          { \tl_to_str:N \l__codedoc_macro_argument_tl }
          { \l__codedoc_tmpa_tl }
      }
    \bool_set_true:N \l__codedoc_macro_internal_bool
    \bool_set_true:N \l__codedoc_macro_internal_set_bool
  }
```

(*End definition for __codedoc_deprecated_on:n.*)

__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 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 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 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 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 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 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 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 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
    \HD@savedestfalse
    \HD@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 definition for* `\__codedoc_typeset_functions:`.)

__codedoc_typeset_function_block:nN
__codedoc_typeset_function_block:xN
__codedoc_function_index:n
__codedoc_function_index:x

`#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:x
      { #1 \bool_if:NT #2 { \tl_to_str:n {TF} } }
    \__codedoc_function_label:xN {#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 { x }
\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 { x }
```

```
\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 { } }
      {
        \textrm(
          \seq_use:Nn \g__codedoc_variants_seq { \textrm| }
        \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:xN
          {
            \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 { x }
```

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

`\__codedoc_syntax:w`
`\__codedoc_syntax_end:`

Implement the `syntax` environment.

```
\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 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 }
      } ,
    internal .value_forbidden:n = true ,
    internal .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_internal_bool
        \bool_set_true:N \l__codedoc_macro_internal_set_bool
      } ,
    int .value_forbidden:n = true ,
    int .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_internal_bool
        \bool_set_true:N \l__codedoc_macro_internal_set_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
    deprecated .code:n = { \__codedoc_deprecated_on:n {#1} } ,
    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 xparse. 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 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_internal_bool
    \bool_set_false:N \l__codedoc_macro_internal_set_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
              { \HD@target }
          }
        \vskip \int_eval:n { \l__codedoc_macro_int - 1 } \baselineskip
      }
  }
```

(*End 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 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:Nf \g__codedoc_nested_names_seq
          { \exp_after:wN \token_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 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
        \tl_set:Nx \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 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 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:xN
          { \__codedoc_get_function_name:n {#1} } #2
      }
  }
\cs_generate_variant:Nn \__codedoc_macro_typeset_block:nN { x }
\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 { x }
```

(*End 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:Nx \__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
      {
        \__codedoc_if_macro_internal:nF {#1}
          { \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 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. Note the space \ . #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 \ } }
      }
    \int_incr:N \l__codedoc_macro_int
  }
```

(*End 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:xN
      {
        \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:Nno \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 definition for __codedoc_print_macroname:nN.*)

\MacroLongFont

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

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

`\__codedoc_print_testfile:n`
`\__codedoc_print_testfile_aux:n`

Used to show that a macro has a test, somewhere.

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

`\UnitTested`

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

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

`\TestMissing`

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

(*End 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:Nxn
      \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 { Nx }
```

(*End 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 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:Nx \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 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 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:Non \l__codedoc_tmpa_tl
      { \token_to_str:N _ } { \_ }
    \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
      { \exp_not:n { \__codedoc_macro_end_wrap_item:n {##1} } }
    End~ 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
      { ! \__codedoc_if_macro_internal_p:n {##1} }
    \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:xN
```

```
     { \seq_item:Nn \g__codedoc_nested_names_seq { 1 } }
     \l__codedoc_tmpa_tl
    \exp_args:Nx \pageref { \l__codedoc_tmpa_tl } .
   }
  \seq_gclear:N \g__codedoc_nested_names_seq
 }
```

(*End 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. Due to Joseph Wright. Name/usage might change soon.

```
\newcommand*{\DescribeOption}
 {
  \leavevmode
  \@bsphack
  \begingroup
   \MakePrivateLetters
   \Describe@Option
 }
```

```
\newcommand*{\Describe@Option}[1]
 {
  \endgroup
  \marginpar{
   \raggedleft
   \PrintDescribeEnv{#1}
  }
  \SpecialOptionIndex{#1}
  \@esphack
  \ignorespaces
 }
```

```
\newcommand*{\SpecialOptionIndex}[1]
 {
  \@bsphack
  \begingroup
   \HD@target
   \let\HDorg@encapchar\encapchar
   \edef\encapchar usage
    {
     \HDorg@encapchar hdclindex{\the\c@HD@hypercount}{usage}
    }
   \index
    {
     #1\actualchar{\protect\ttfamily#1}~(option)
     \encapchar usage
    }
   \index
    {
     options:\levelchar#1\actualchar{\protect\ttfamily#1}
     \encapchar usage
    }
  \endgroup
```

```
    \@esphack
  }
```

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

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

danger
ddanger

```
\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 definition for* \NB*. This function is documented on page 6.*)

NOTE

```
\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 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][xxxxxxxxxxxxxxx]
  {
    \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 }
```

5.11 Inheriting doc

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

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

(*End 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 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 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 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:Npx \@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:Npx \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
  }
\cs_gset:Npn \codeline@wrindex #1
  {
    \immediate\write\@indexfile
      {
        \string\indexentry{#1}
          { \filesep \int_use:N \c@CodelineNo }
      }
  }
\tl_gclear:N \filesep
```

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

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

5.12 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
\cs_new_protected:Npn \__codedoc_show_functions_defined:
  {
    \bool_lazy_and:nnT
      { \g__codedoc_typeset_implementation_bool } { \g__codedoc_checkfunc_bool }
      {
        \iow_term:x { \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_map_inline:Nn \g_doc_functions_seq
          {
            \seq_if_in:NnTF \g_doc_macros_seq {##1}
              {
                \tl_put_right:Nx \l__codedoc_doc_def_tl
                  { ##1 \iow_newline: }
                \iow_now:Nn \g__codedoc_func_iow { > ~ ##1 }
              }
              {
                \tl_put_right:Nx \l__codedoc_doc_undef_tl
                  { ##1 \iow_newline: }
                \iow_now:Nn \g__codedoc_func_iow { ! ~ ##1 }
              }
          }
        \seq_map_inline:Nn \g_doc_macros_seq
          {
            \seq_if_in:NnF \g_doc_functions_seq {##1}
              {
                \tl_put_right:Nx \l__codedoc_undoc_def_tl
                  { ##1 \iow_newline: }
                \iow_now:Nn \g__codedoc_func_iow { ? ~ ##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:x { \c__codedoc_iow_separator_tl }
      }
  }
```

```
\AtEndDocument { \__codedoc_show_functions_defined: }
```

TODO: use \iow_term:x.

```
\cs_new_protected:Npn \__codedoc_functions_typeout:nN #1#2
  {
    \tl_if_empty:NF #2
      {
        \typeout
          {
            \c__codedoc_iow_midrule_tl \iow_newline:
            #1 \iow_newline:
            \c__codedoc_iow_midrule_tl \iow_newline:
            #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:Nx \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:Nx \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: }
```

5.13 Indexing

5.13.1 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 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 } } }
```

5.13.2 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:x
      {
        \__codedoc_special_index_module:nnnnN
          {#1}
          { \bslash #1 }
          { }
          { }
          \c_false_bool
      }
  }
```

(*End 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 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:x
      {
        \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 }
      { \codeline@wrindex }
      {
        \str_case:nnF {#6}
          {
            { main }  { \codeline@wrindex }
            { usage } { \index }
          }
          { \HD@target \index }
      }
      {
        \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
        }
        \encapchar
        hdclindex{\the\c@HD@hypercount}{#6}
      }
  }
\cs_new_protected:Npn \__codedoc_special_index_set:Nn #1#2
  {
    \tl_set:Nx #1 { \tl_to_str:n {#2} }
    \__codedoc_if_almost_str:nTF {#2}
      {
        \tl_replace_all:Non #1 { \tl_to_str:n { __ } }
          {
            \verbatimchar
            \token_to_str:N \_ \token_to_str:N \_
            \token_to_str:N \verb * \verbatimchar
          }
        \exp_args:Nx \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
              }
          }
        \tl_set:Nx #1
          {
            \token_to_str:N \verb * \verbatimchar
            #1 \verbatimchar
          }
      }
      {
        \tl_set:Nn #1 {#2}
        \tl_replace_all:Non #1
          { \c_backslash_str }
          { \token_to_str:N \bslash \c_space_tl }
      }
    \__codedoc_quote_special_char:N #1
  }
```

(*End definition for __codedoc_special_index_module:nnnnN, __codedoc_special_index_aux:nnnnnn, and __codedoc_special_index_set:Nn.*)

__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:Nxn #1
          { \tl_to_str:N ##1 } { \quotechar \tl_to_str:N ##1 }
      }
  }
```

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

5.13.3 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 2$_\varepsilon$ 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 2$_\varepsilon$ commands the module name is `TeX`, and others have an empty module name.

```
\cs_new_protected:Npn \__codedoc_key_get:n #1
  {
    \__codedoc_key_get_base:nN {#1} \l__codedoc_index_macro_tl
    \tl_set:Nx \l__codedoc_index_key_tl
      { \tl_to_str:N \l__codedoc_index_macro_tl }
    \tl_clear:N \l__codedoc_index_module_tl
    \tl_if_in:NoTF \l__codedoc_index_key_tl { \tl_to_str:n { __ } }
      { \bool_set_true:N \l__codedoc_index_internal_bool }
      { \bool_set_false:N \l__codedoc_index_internal_bool }
    \exp_last_unbraced:NNo
    \tl_if_head_eq_charcode:oNT
      { \l__codedoc_index_key_tl } \c_backslash_str
      { \__codedoc_key_pop: }
    \tl_if_in:NoTF \l__codedoc_index_key_tl { \token_to_str:N : }
      { \__codedoc_key_func: }
      {
        \tl_if_in:NoTF \l__codedoc_index_key_tl { \token_to_str:N _ }
          { \__codedoc_key_var: }
          {
            \tl_if_in:NoT \l__codedoc_index_key_tl { \token_to_str:N @ }
              { \tl_set:Nn \l__codedoc_index_module_tl { TeX } }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_key_pop:
  {
    \tl_set:Nx \l__codedoc_index_key_tl
      { \tl_tail:N \l__codedoc_index_key_tl }
  }
```

(*End 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} }
    \tl_set:Nx \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:oNT { \l__codedoc_index_key_tl } _
      { \__codedoc_key_pop: \__codedoc_key_drop_underscores: }
  }
```

(*End 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:oNT { \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 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. If there is no underscore left we had something like \c_zero which we assume is an integer constant. If there is one underscore we assume it is a variable like \c_empty_tl whose module name is the last part. Otherwise the module name is the part before any underscore.

```
\cs_new_protected:Npn \__codedoc_key_var:
  {
    \exp_args:Nx \tl_if_head_eq_charcode:nNTF
      { \exp_args:No \str_tail:n \l__codedoc_index_key_tl } _
      {
        \str_case:fn { \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 { quark } }
          }
        \__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
            \tl_set:Nx \l__codedoc_index_module_tl
```

```
              {
                \int_case:nnF { \seq_count:N \l__codedoc_tmpa_seq }
                  {
                    { 0 } { }
                    { 1 } { int }
                    { 2 } { \seq_item:Nn \l__codedoc_tmpa_seq { 2 } }
                  }
                  { \seq_item:Nn \l__codedoc_tmpa_seq { 1 } }
              }
          }
      }
      {
        \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 definition for* __codedoc_key_var: *and* __codedoc_key_get_module:.)

5.14 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 } } }
```

5.15 Default configuration

```
\bool_if:NTF \g__codedoc_typeset_implementation_bool
  {
    \RecordChanges
    \CodelineIndex
    \EnableCrossrefs
    \AlsoImplementation
  }
  {
    \CodelineNumbered
    \DisableCrossrefs
    \OnlyDescription
  }
⟨/class⟩
```

5.16 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
  }
```

5.17 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 definition for* `\nan`. *This function is documented on page* ??.)

```
⟨/cfg⟩
```

5.18 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 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.