

The l3doc class

The L^AT_EX3 Project*

Released 2019-11-07

Contents

1	Introduction	2
2	Features of other packages	2
2.1	The hypdoc package	2
2.2	The docmfp package	3
2.3	The xdoc2 package	3
2.4	The gmdoc package	3
3	Problems & Todo	3
4	Documentation	4
4.1	Configuration	4
4.2	Partitioning documentation and implementation	4
4.3	General text markup	4
4.4	Describing functions in the documentation	6
4.5	Describing functions in the implementation	7
4.6	Keeping things consistent	8
4.7	Documenting templates	8
5	l3doc implementation	8
5.1	Variables	9
5.2	Variants and helpers	13
5.3	Messages	20
5.4	Options and configuration	20
5.5	Class and package loading	21
5.6	Configuration and tweaks	22
5.7	Design	23
5.8	Text markup	24
5.9	Implementing text markup	28
5.9.1	Common between macro and function	30
5.9.2	The function environment	35
5.9.3	The macro environment	43
5.9.4	Misc	52
5.9.5	NB and NOTE	53

*<https://www.latex-project.org/latex3/>

5.10 Documenting templates	54
5.11 Inheriting doc	55
5.11.1 The <code>macrocode</code> environment	58
5.12 At end document	59
5.13 Indexing	62
5.13.1 Userspace commands	62
5.13.2 Internal index commands	63
5.13.3 Finding sort-key and module	66
5.14 Change history	68
5.15 Default configuration	68
5.16 Internal macros for \LaTeX 3 sources	69
5.17 Math extras	69
5.18 Makeindex configuration	70

Index	70
--------------	-----------

1 Introduction

This is an ad-hoc class for documenting the `expl3` bundle, a collection of modules or packages that make up \LaTeX 3’s programming environment. Eventually it will replace the `ltxdoc` class for \LaTeX 3, 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 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.

<code>\cmd</code>	<code>\cmd [<i>options</i>] <control sequence></code>
<code>\cs</code>	<code>\cs [<i>options</i>] {<csname>}</code>

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

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

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

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

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

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

<code>\tn</code>	<code>\tn [<i>options</i>] {<csname>}</code>
------------------	--

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>}`.

<code>\meta</code>	<code>\meta {<name>}</code>
--------------------	-----------------------------------

`\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}*”.

<code>\Arg</code>	<code>\Arg {<name>}</code>
-------------------	----------------------------------

<code>\marg</code>	Typesets the <i>name</i> as for <code>\meta</code> and wraps it in braces.
<code>\oarg</code>	
<code>\parg</code>	

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

<code>\file</code>	<code>\pkg {<name>}</code>
--------------------	----------------------------------

<code>\env</code>	These all take one argument and are intended to be used as semantic commands for representing files, environments, package names, and class names, respectively.
<code>\pkg</code>	
<code>\cls</code>	

<code>\NB</code>	<code>\NB {<tag>} {<comments>}</code>
<code>\NOTE</code>	<code>\begin{NOTE} {<tag>}</code> <code> <comments></code> <code>\end{NOTE}</code>

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

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

<code>\function_one:</code>	<code>\foo_bar: {<meta>} <test₁></code>
<code>\function_two:</code>	<code><description></code>

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}
```

<code>\cs_if_exist_p:N</code> ★	<code>\cs_if_exist_p:N <cs></code>
<code>\cs_if_exist:N</code> ★	<code><description></code>

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 L^AT_EX 2_ε 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} {\langle template type name \rangle}
  \TemplateArgument{none}{---}
OR ONE OR MORE OF THESE:
  \TemplateArgument {\langle arg no \rangle} {\langle meaning \rangle}
AND
\TemplateSemantics
  \langle text describing the template type semantics \rangle
\end{TemplateInterfaceDescription}

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

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

5 l3doc implementation

```
1 \<class>
2 \@@=codedoc>
```


5.1 Variables

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

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

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

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

```
4 \seq_new:N \g_doc_functions_seq
```

```
5 \seq_new:N \g_doc_macros_seq
```

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

`\l_codedoc_detect_internals_bool` 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.
`\l_codedoc_detect_internals_tl`

```
6 \bool_new:N \l_codedoc_detect_internals_bool
```

```
7 \bool_set_true:N \l_codedoc_detect_internals_bool
```

```
8 \tl_new:N \l_codedoc_detect_internals_tl
```

```
9 \tl_new:N \l_codedoc_detect_internals_cs_tl
```

(End definition for `\l_codedoc_detect_internals_bool` and `\l_codedoc_detect_internals_tl`.)

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

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

(End definition for `\l__codedoc_output_coffin`.)

`\l__codedoc_functions_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.
`\l__codedoc_descr_coffin`
`\l__codedoc_syntax_coffin`

```
11 \coffin_new:N \l__codedoc_functions_coffin
```

```
12 \coffin_new:N \l__codedoc_descr_coffin
```

```
13 \coffin_new:N \l__codedoc_syntax_coffin
```

(End 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`.

```
14 \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.

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

(End definition for `\l__codedoc_in_function_bool`.)

<code>\l__codedoc_long_name_bool</code> <code>\l__codedoc_trial_width_dim</code>	<p>The boolean <code>\l__codedoc_long_name_bool</code> is true if the width <code>\l__codedoc_trial_width_dim</code> of the coffin <code>\l__codedoc_functions_coffin</code> (containing the current function names) is bigger than the space available in the margin.</p> <pre> 16 \bool_new:N \l__codedoc_long_name_bool 17 \dim_new:N \l__codedoc_trial_width_dim </pre> <p>(End definition for <code>\l__codedoc_long_name_bool</code> and <code>\l__codedoc_trial_width_dim</code>.)</p>
<code>\l__codedoc_nested_macro_int</code>	<p>The nesting of macro environments (this is now 0 outside a <code>macro</code> environment).</p> <pre> 18 \int_new:N \l__codedoc_nested_macro_int </pre> <p>(End definition for <code>\l__codedoc_nested_macro_int</code>.)</p>
<code>\l__codedoc_macro_tested_bool</code> <code>\g__codedoc_missing_tests_prop</code> <code>\g__codedoc_not_tested_seq</code> <code>\g__codedoc_testfiles_seq</code>	<p>A boolean describing whether the current macro has tests, and some global structures which contain information about test files and which tests are missing.</p> <pre> 19 \bool_new:N \l__codedoc_macro_tested_bool 20 \prop_new:N \g__codedoc_missing_tests_prop 21 \seq_new:N \g__codedoc_not_tested_seq 22 \seq_new:N \g__codedoc_testfiles_seq </pre> <p>(End definition for <code>\l__codedoc_macro_tested_bool</code> and others.)</p>
<code>\l__codedoc_macro_internal_set_bool</code> <code>\l__codedoc_macro_internal_bool</code> <code>\l__codedoc_macro_TF_bool</code> <code>\l__codedoc_macro_pTF_bool</code> <code>\l__codedoc_macro_noTF_bool</code> <code>\l__codedoc_macro_EXP_bool</code> <code>\l__codedoc_macro_rEXP_bool</code> <code>\l__codedoc_macro_var_bool</code> <code>\l__codedoc_override_module_tl</code> <code>\l__codedoc_macro_documented_tl</code>	<p>Contain information about some options of function/macro environments. We initialize <code>\l__codedoc_override_module_tl</code> to avoid overriding module names by an empty name (meaning no module).</p> <pre> 23 \bool_new:N \l__codedoc_macro_internal_set_bool 24 \bool_new:N \l__codedoc_macro_internal_bool 25 \bool_new:N \l__codedoc_macro_TF_bool 26 \bool_new:N \l__codedoc_macro_pTF_bool 27 \bool_new:N \l__codedoc_macro_noTF_bool 28 \bool_new:N \l__codedoc_macro_EXP_bool 29 \bool_new:N \l__codedoc_macro_rEXP_bool 30 \bool_new:N \l__codedoc_macro_var_bool 31 \tl_new:N \l__codedoc_override_module_tl 32 \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value } 33 \tl_new:N \l__codedoc_macro_documented_tl </pre> <p>(End definition for <code>\l__codedoc_macro_internal_set_bool</code> and others.)</p>
<code>\g__codedoc_lmodern_bool</code> <code>\g__codedoc_checkfunc_bool</code> <code>\g__codedoc_checktest_bool</code> <code>\g__codedoc_cs_break_bool</code> <code>\g__codedoc_show_notes_bool</code> <code>\g__codedoc_kernel_bool</code>	<p>Information about package options.</p> <pre> 34 \bool_new:N \g__codedoc_lmodern_bool 35 \bool_new:N \g__codedoc_checkfunc_bool 36 \bool_new:N \g__codedoc_checktest_bool 37 \bool_new:N \g__codedoc_kernel_bool 38 \bool_new:N \g__codedoc_cs_break_bool 39 \bool_new:N \g__codedoc_show_notes_bool 40 \bool_gset_true:N \g__codedoc_cs_break_bool </pre> <p>(End definition for <code>\g__codedoc_lmodern_bool</code> and others.)</p>
<code>\l__codedoc_tmpa_tl</code> <code>\l__codedoc_tmpp_tl</code> <code>\l__codedoc_tmpa_int</code> <code>\l__codedoc_tmpa_seq</code>	<p>Some temporary variables.</p> <pre> 41 \tl_new:N \l__codedoc_tmpa_tl 42 \tl_new:N \l__codedoc_tmpp_tl 43 \int_new:N \l__codedoc_tmpa_int 44 \int_new:N \l__codedoc_tmpa_seq </pre>

(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).

45 `\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.

46 `\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.

47 `\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.

48 `\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.

49 `\seq_new:N \g__codedoc_nested_names_seq`

(End definition for `\g__codedoc_nested_names_seq`.)

`\l__codedoc_index_macro_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 specific **macro** environment.

50 `\tl_new:N \l__codedoc_index_macro_tl`

51 `\tl_new:N \l__codedoc_index_key_tl`

52 `\tl_new:N \l__codedoc_index_module_tl`

53 `\tl_new:N \l__codedoc_macro_do_not_index_tl`

54 `\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>`.

```

55 \tl_new:N \g__codedoc_module_name_tl

```

(End definition for `\g__codedoc_module_name_tl`.)

`\c__codedoc_iow_rule_tl` 40 equal signs.

```

56 \tl_const:Nn \c__codedoc_iow_rule_tl
57 { ===== }
58 \tl_const:Nn \c__codedoc_iow_mid_rule_tl
59 { ----- }

```

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

`\l__codedoc_macro_box` 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).

```

60 \box_new:N \l__codedoc_macro_box
61 \box_new:N \l__codedoc_macro_index_box
62 \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` Variables used to control the behaviour of `\cmd`, `\cs` and `\tn`.

```

63 \tl_new:N \l__codedoc_cmd_tl
64 \tl_new:N \l__codedoc_cmd_index_tl
65 \tl_new:N \l__codedoc_cmd_module_tl
66 \bool_new:N \l__codedoc_cmd_noindex_bool
67 \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.

```

68 \bool_new:N \l__codedoc_in_implementation_bool

```

(End definition for `\l__codedoc_in_implementation_bool`.)

`\g__codedoc_typeset_documentation_bool` These booleans control whether the documentation/implementation should be typeset.

`\g__codedoc_typeset_implementation_bool` By default both should be.

```

69 \bool_new:N \g__codedoc_typeset_documentation_bool
70 \bool_new:N \g__codedoc_typeset_implementation_bool
71 \bool_set_true:N \g__codedoc_typeset_documentation_bool
72 \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` 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.

```

73 \tl_new:N \g__codedoc_base_name_tl
74 \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` 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.

```
75 \clist_new:N \l__codedoc_function_label_clist
76 \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` Values of some options of the **function** environment.
`\l__codedoc_date_updated_tl`

```
77 \tl_new:N \l__codedoc_date_added_tl
78 \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.

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

(End definition for `\l__codedoc_macro_argument_tl`.)

```
80 % \int_new:N \c@CodelineNo
```

5.2 Variants and helpers

`__codedoc_tmpa:w` Auxiliary macros for temporary use.

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

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

`\seq_set_split:NoV` A few missing variants.

```
\str_case:fn
\str_count:f
83 \cs_generate_variant:Nn \seq_set_split:Nnn { NoV }
\str_grep:fn
84 \cs_generate_variant:Nn \seq_grep:Nnn { NoV }
\str_replace_all:Nxn
85 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_replace_all:Nno
86 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_if_head_eq_charcode:oNTF
87 \cs_generate_variant:Nn \str_if_head_eq_charcode:Nnn { NoV }
\str_if_head_eq_charcode:oNT
88 \cs_generate_variant:Nn \str_if_head_eq_charcode:Nnn { NoV }
\str_if_head_eq_charcode:oNF
89 \cs_generate_variant:Nn \str_if_head_eq_charcode:Nnn { NoV }
\str_if_head_eq_meaning:VNF
90 \cs_generate_variant:Nn \str_if_head_eq_meaning:Nnn { NoV }
\str_if_in:noTF
91 \cs_generate_variant:Nn \str_if_in:Nnn { NoV }
\str_if_in:ooTF
92 \cs_generate_variant:Nn \str_if_in:Nnn { NoV }
\str_if_in:NoTF
93 \cs_generate_variant:Nn \str_if_in:Nnn { NoV }
\str_if_in:NoT
94 \cs_generate_variant:Nn \str_if_in:Nnn { NoV }
\str_if_in:NoF
95 \cs_generate_variant:Nn \str_if_in:Nnn { NoV }
\str_remove_all:Nx
96 \cs_generate_variant:Nn \str_remove_all:Nnn { NoV }
\str_replace_all:Nxn
97 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_replace_all:Nnx
98 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_replace_all:Non
99 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_replace_all:Nno
100 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_replace_all:Noo
101 \cs_generate_variant:Nn \str_replace_all:Nnn { NoV }
\str_to_str:f
102 \cs_generate_variant:Nn \str_to_str:Nnn { NoV }
\str_to_str:o
103 \cs_generate_variant:Nn \str_to_str:Nnn { NoV }
\prop_get:NxNTF
104 \cs_generate_variant:Nn \prop_get:Nnn { NoV }
\prop_put:Nxn
\prop_gput:NVx
```

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

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

```

105 \prg_new_protected_conditional:Npnn \__codedoc_if_almost_str:n #1 { TF , T , F }
106 {
107   \int_compare:nNnTF
108     { \tl_count:n {#1} }
109     < { \tl_count:f { \tl_to_str:f {#1} } }
110     { \prg_return_false: }
111     { \prg_return_true: }
112 }
113 \cs_generate_variant:Nn \__codedoc_if_almost_str:nT { V }

```

(End definition for `__codedoc_if_almost_str:nTF`.)

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

```

114 \cs_new_protected:Npn \__codedoc_trim_right:Nn #1#2
115 {
116   \cs_set:Npn \__codedoc_tmp:w ##1 #2 ##2 \q_stop { \exp_not:n {##1} }
117   \tl_set:Nx #1 { \exp_after:wN \__codedoc_tmp:w #1 #2 \q_stop }
118 }
119 \cs_generate_variant:Nn \__codedoc_trim_right:Nn { No }

```

(End definition for `__codedoc_trim_right:Nn`.)

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

```

120 \prg_new_protected_conditional:Npnn \__codedoc_str_if_begin:nn #1#2 { TF , T , F }
121 {
122   \tl_if_in:ooTF
123     { \exp_after:wN \scan_stop: \tl_to_str:n {#1} }
124     { \exp_after:wN \scan_stop: \tl_to_str:n {#2} }
125     { \prg_return_true: }
126     { \prg_return_false: }
127 }
128 \prg_generate_conditional_variant:Nnn \__codedoc_str_if_begin:nn
129 { oo } { TF , T , F }

```

(End definition for `__codedoc_str_if_begin:nnTF`.)

`__codedoc_replace_at_at:N` The goal is to replace `@@` by the current module name. We take advantage of this function
`__codedoc_replace_at_at_aux:Nn` 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.

```

130 \cs_new_protected:Npn \__codedoc_replace_at_at:N #1
131 {
132   \tl_if_empty:NF \g__codedoc_module_name_tl
133   {

```

```

134         \exp_args:NNo \__codedoc_replace_at_at_aux:Nn
135         #1 \g__codedoc_module_name_tl
136     }
137 }
138 \cs_new_protected:Npx \__codedoc_replace_at_at_aux:Nn #1#2
139 {
140     \tl_replace_all:Nnn #1 { \token_to_str:N @ } { @ }
141     \tl_replace_all:Nnn #1 { \token_to_str:N _ } { _ }
142     \tl_replace_all:Nnn #1 { @ @ @ @ } { \token_to_str:N a a }
143     \tl_replace_all:Nnn #1 { _ _ @ @ } { _ _ #2 }
144     \tl_replace_all:Nnn #1 { _ _ @ @ } { _ _ #2 }
145     \tl_replace_all:Nnn #1 { @ @ } { _ _ #2 }
146     \tl_replace_all:Nnn #1 { \token_to_str:N a a } { @ @ }
147 }

```

(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.

```

148 \cs_new_protected:Npn \__codedoc_detect_internals:N #1
149 {
150     \bool_if:NT \l__codedoc_detect_internals_bool
151     { \__codedoc_detect_internals_aux:N #1 }
152 }
153 \group_begin:
154 \char_set_catcode_active:N ^^M
155 \cs_new_protected:Npn \__codedoc_detect_internals_aux:N #1
156 {
157     \tl_set_eq:NN \l__codedoc_detect_internals_tl #1
158     \tl_replace_all:Non \l__codedoc_detect_internals_tl { \token_to_str:N _ } { _ }
159     \seq_set_split:NnV \l__codedoc_tmpa_seq { _ _ } \l__codedoc_detect_internals_tl
160     \seq_pop_left:NN \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
161     \seq_map_variable:NNn \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
162     {
163         \__codedoc_trim_right:No \l__codedoc_detect_internals_tl
164         \c_catcode_active_space_tl
165         \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl ^^M
166         \__codedoc_if_detect_internals_ok:NF \l__codedoc_detect_internals_tl
167         {
168             \tl_set_eq:NN \l__codedoc_detect_internals_cs_tl \l__codedoc_detect_internals_
169             \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl _
170             \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl :
171             \__codedoc_trim_right:No \l__codedoc_detect_internals_tl { \token_to_str:N : }
172             \msg_warning:nxxx { l3doc } { foreign-internal }
173             { \tl_to_str:N \l__codedoc_detect_internals_cs_tl }
174             { \tl_to_str:N \l__codedoc_detect_internals_tl }
175             { \tl_to_str:N \g__codedoc_module_name_tl }
176         }
177     }

```

```

178     }
179 \group_end:
180 \prg_new_protected_conditional:Npnn \__codedoc_if_detect_internals_ok:N #1 { F }
181 {
182   \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl _ }
183   { \prg_return_true: }
184   {
185     \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl : }
186     { \prg_return_true: }
187     {
188       \bool_if:NTF \g__codedoc_kernel_bool
189       {
190         \__codedoc_str_if_begin:ooTF {#1} { kernel _ }
191         { \prg_return_true: }
192         { \prg_return_false: }
193       }
194       { \prg_return_false: }
195     }
196   }
197 }

```

(End definition for __codedoc_detect_internals:N, __codedoc_detect_internals_aux:N, and __codedoc_if_detect_internals_ok:NF.)

__codedoc_signature_base_form:n 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.

```

198 \cs_new:Npn \__codedoc_signature_base_form:n #1
199 { \__codedoc_signature_base_form_aux:n #1 \q_stop }
200 \cs_new:Npn \__codedoc_signature_base_form_aux:n #1
201 {
202   \str_case:nnTF {#1}
203   {
204     { N } { N }
205     { c } { N }
206     { n } { n }
207     { o } { n }
208     { f } { n }
209     { e } { n }
210     { x } { n }
211     { V } { n }
212     { v } { n }
213   }
214   { \__codedoc_signature_base_form_aux:n }
215   { \__codedoc_signature_base_form_aux:w #1 }
216 }
217 \cs_new:Npn \__codedoc_signature_base_form_aux:w #1 \q_stop
218 { \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.


```

219 \cs_new:Npn \__codedoc_predicate_from_base:n #1
220 {
221   \__codedoc_get_function_name:n {#1}
222   \tl_to_str:n { _p: }
223   \__codedoc_get_function_signature:n {#1}
224 }

```

(End definition for __codedoc_predicate_from_base:n.)

```

\__codedoc_split_function_do:nn Similar to internal functions defined in l3basics, but here we operate on strings directly
\__codedoc_split_function_do:on rather than control sequences.
\__codedoc_get_function_name:n
\__codedoc_get_function_signature:n
\__codedoc_split_function_auxi:w
\__codedoc_split_function_auxii:w
225 \cs_new:Npn \__codedoc_get_function_name:n #1
226 { \__codedoc_split_function_do:nn {#1} { \use_i:nnn } }
227 \cs_new:Npn \__codedoc_get_function_signature:n #1
228 { \__codedoc_split_function_do:nn {#1} { \use_ii:nnn } }
229 \cs_set_protected:Npn \__codedoc_tmpa:w #1
230 {
231   \cs_new:Npn \__codedoc_split_function_do:nn ##1
232   {
233     \exp_after:wN \__codedoc_split_function_auxi:w
234     \tl_to_str:n {##1} \q_mark \c_true_bool
235     #1 \q_mark \c_false_bool
236     \q_stop
237   }
238   \cs_new:Npn \__codedoc_split_function_auxi:w
239   ##1 #1 ##2 \q_mark ##3##4 \q_stop ##5
240   { \__codedoc_split_function_auxii:w {##5} ##1 \q_mark \q_stop {##2} ##3 }
241   \cs_new:Npn \__codedoc_split_function_auxii:w
242   ##1##2 \q_mark ##3 \q_stop
243   { ##1 {##2} }
244 }
245 \exp_args:No \__codedoc_tmpa:w { \token_to_str:N : }
246 \cs_generate_variant:Nn \__codedoc_split_function_do:nn { o }

```

(End definition for __codedoc_split_function_do:nn and others.)

__codedoc_key_get_base:n 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.

```

247 \cs_new_protected:Npn \__codedoc_key_get_base:nN #1#2
248 {
249   \__codedoc_if_almost_str:nTF {#1}
250   {
251     \__codedoc_key_get_base_TF:nN {#1} \l__codedoc_tmpa_tl
252     \tl_set:Nx #2
253     { \__codedoc_split_function_do:on \l__codedoc_tmpa_tl { \__codedoc_base_form_aux:n
254     }
255     { \tl_set:Nn #2 {#1} }
256   }
257   \cs_new:Npx \__codedoc_key_get_base_TF:nN #1#2
258   {
259     \tl_set:Nx #2 { \exp_not:N \tl_to_str:n {#1} }
260     \tl_if_in:Nof #2 { \tl_to_str:n {:} }

```

```

261     { \exp_not:N \prg_break: }
262 \tl_if_in:onT { #2 z } { \tl_to_str:n {TF} z }
263     { \exp_not:N \prg_break: }
264 \tl_if_in:onT { #2 z } { \tl_to_str:n {T} z }
265     {
266       \tl_put_right:Nn #2 { \tl_to_str:n {F} }
267       \exp_not:N \prg_break:
268     }
269 \tl_if_in:onT { #2 z } { \tl_to_str:n {F} z }
270     {
271       \tl_put_right:Nn #2 { z }
272       \tl_replace_once:Nnn #2 { \tl_to_str:n {F} z } { \tl_to_str:n {TF} }
273       \exp_not:N \prg_break:
274     }
275 \exp_not:N \prg_break_point:
276 }
277 \cs_new:Npn \__codedoc_base_form_aux:nnN #1#2#3
278 {
279   \exp_not:n {#1}
280   \bool_if:NT #3
281   {
282     \token_to_str:N :
283     \bool_lazy_or:nnTF
284       { \str_if_eq_p:nn { #1 ~ } { \exp_args } }
285       { \str_if_eq_p:nn { #1 ~ } { \exp_last_unbraced } }
286     { \exp_not:n {#2} }
287     { \__codedoc_signature_base_form:n {#2} }
288   }
289 }

```

(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.

```

290 \cs_new_protected:Npn \__codedoc_base_form_signature_do:nnn #1#2#3
291 {
292   \__codedoc_split_function_do:nn {#1}
293   { \__codedoc_base_form_aux:nnnnN {#1} {#2} {#3} }
294 }
295 \cs_new_protected:Npn \__codedoc_base_form_aux:nnnnN #1#2#3#4#5#6
296 {
297   \bool_if:NTF #6
298   {
299     \tl_if_head_eq_charcode:nNTF {#4} :
300     { #2 {#1} }
301     {
302       \use:x
303       {
304         \exp_not:n {#3}
305         { \__codedoc_base_form_aux:nnN {#4} {#5} #6 }
306       }
307       {#4} {#5}
308     }

```

```

309     }
310     { #2 {#1} }
311 }

```

(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.

```

312 \prg_new_conditional:Npnn \__codedoc_date_compare:nNn #1#2#3 { TF , T , F , p }
313 { \__codedoc_date_compare_aux:w #1--- \q_mark #2 #3--- \q_stop }
314 \cs_new:Npn \__codedoc_date_compare_aux:w
315   #1 - #2 - #3 - #4 \q_mark #5 #6 - #7 - #8 - #9 \q_stop
316 {
317   \__codedoc_date_compare_aux:nnnNnnn
318   { \tl_if_empty:nTF {#1} { 0 } {#1} }
319   { \tl_if_empty:nTF {#2} { 0 } {#2} }
320   { \tl_if_empty:nTF {#3} { 0 } {#3} }
321   #5
322   { \tl_if_empty:nTF {#6} { 0 } {#6} }
323   { \tl_if_empty:nTF {#7} { 0 } {#7} }
324   { \tl_if_empty:nTF {#8} { 0 } {#8} }
325 }
326 \cs_new:Npn \__codedoc_date_compare_aux:nnnNnnn #1#2#3#4#5#6#7
327 {
328   \int_compare:nNnTF {#1} = {#5}
329   {
330     \int_compare:nNnTF {#2} = {#6}
331     {
332       \int_compare:nNnTF {#3} #4 {#7}
333       { \prg_return_true: } { \prg_return_false: }
334     }
335     {
336       \int_compare:nNnTF {#2} #4 {#6}
337       { \prg_return_true: } { \prg_return_false: }
338     }
339   }
340   {
341     \int_compare:nNnTF {#1} #4 {#5}
342     { \prg_return_true: } { \prg_return_false: }
343   }
344   \use_none:n
345   \q_stop
346 }

```

(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.

```

347 \cs_new:Npn \__codedoc_gprop_name:n #1 { g__codedoc ~ \tl_to_str:n {#1} }
348 \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

```

349 \msg_new:nnnn { l3doc } { no-signature-TF }
350 { Function/macro~'#1'~cannot~be~turned~into~a~conditional. }
351 {
352   A~function~or~macro~environment~with~option~pTF,~TF~or~noTF~
353   received~the~argument~'#1'.~This~function's~name~has~no~
354   ':'~hence~it~is~not~clear~where~to~add~'_p'~or~'TF'.~
355   Please~follow~expl3~naming~conventions.
356 }
357 \msg_new:nnn { l3doc } { deprecated-function }
358 { The~deprecated~function(s)~'#1'~should~have~been~removed~on~'#2'. }
359 \msg_new:nnn { l3doc } { date-format }
360 { The~date~'#1'~should~be~given~in~YYYY-MM-DD~format. }
361 \msg_new:nnn { l3doc } { future-date }
362 { The~added/updated~date~'#2'~of~'#1'~is~in~the~future. }
363 \msg_new:nnn { l3doc } { syntax-nested-function }
364 {
365   The~'syntax'~environment~should~be~used~in~the~
366   innermost~'function'~environment.
367 }
368 \msg_new:nnn { l3doc } { multiple-syntax }
369 {
370   The~'syntax'~environment~should~only~be~used~once~in~
371   a~'function'~environment.
372 }
373 \msg_new:nnn { l3doc } { deprecated-option }
374 { The~option~'#1'~has~been~deprecated~for~'#2'. }
375 \msg_new:nnn { l3doc } { foreign-internal }
376 {
377   A~control~sequence~of~the~form~'..._#1'~was~used.~
378   It~should~only~be~used~in~the~module~'#2'
379   \tl_if_empty:nF {#3} { ,~not~in~'#3' } .
380 }

```

5.4 Options and configuration

```

381 \DeclareOption { a5paper } { \@latexerr { Option~not~supported } { } }
382 \DeclareOption { full }
383 {
384   \bool_gset_true:N \g__codedoc_typeset_documentation_bool
385   \bool_gset_true:N \g__codedoc_typeset_implementation_bool
386 }
387 \DeclareOption { onlydoc }
388 {
389   \bool_gset_true:N \g__codedoc_typeset_documentation_bool
390   \bool_gset_false:N \g__codedoc_typeset_implementation_bool
391 }
392 \DeclareOption { check }
393 { \bool_gset_true:N \g__codedoc_checkfunc_bool }
394 \DeclareOption { nocheck }
395 { \bool_gset_false:N \g__codedoc_checkfunc_bool }

```

```

396 \DeclareOption { checktest }
397   { \bool_gset_true:N \g__codedoc_checktest_bool }
398 \DeclareOption { nochecktest }
399   { \bool_gset_false:N \g__codedoc_checktest_bool }
400 \DeclareOption { kernel }
401   { \bool_gset_true:N \g__codedoc_kernel_bool }
402 \DeclareOption { stdmodule }
403   { \bool_gset_false:N \g__codedoc_kernel_bool }
404 \DeclareOption { cm-default }
405   { \bool_gset_false:N \g__codedoc_lmodern_bool }
406 \DeclareOption { lm-default }
407   { \bool_gset_true:N \g__codedoc_lmodern_bool }
408 \DeclareOption { cs-break-off }
409   { \bool_gset_false:N \g__codedoc_cs_break_bool }
410 \DeclareOption { cs-break-nohyphen }
411   { \PassOptionsToPackage{nohyphen}{underscore} }
412 \DeclareOption { show-notes }
413   { \bool_gset_true:N \g__codedoc_show_notes_bool }
414 \DeclareOption { hide-notes }
415   { \bool_gset_false:N \g__codedoc_show_notes_bool }
416 \DeclareOption* { \PassOptionsToClass { \CurrentOption } { article } }
417 \ExecuteOptions { full, kernel, nocheck, nochecktest, lm-default }
418 \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.

```

419 \msg_new:nnn { l3doc } { input-cfg }
420   { Local-config~file~l3doc.cfg~loaded. }
421 \file_if_exist:nT { l3doc.cfg }
422   {
423     \file_input:n { l3doc.cfg }
424     \cs_if_exist:cF { ExplMakeTitle }
425       { \msg_info:nn { l3doc } { input-cfg } }
426   }
427 \ProcessOptions

```

5.5 Class and package loading

```

428 \LoadClass{article}
429 \RequirePackage{doc}
430 \RequirePackage
431   {
432     array,
433     alphalph,
434     amsmath,
435     amssymb,
436     booktabs,
437     color,
438     colortbl,
439     hologo,

```

```

440     enumitem,
441     pifont,
442     textcomp,
443     trace,
444     csquotes,
445     fancyvrb,
446     underscore,
447     verbatim
448   }
449   \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)

```

450   \bool_if:NT \g__codedoc_lmodern_bool
451   {
452     \RequirePackage[T1]{fontenc}
453     \RequirePackage{lmodern}
454     \group_begin:
455       \ttfamily
456       \DeclareFontShape{T1}{lmtt}{m}{it}{<->ec-lmtto10}{ }
457     \group_end:
458   }

```

Must be last, as usual.

```

459   \RequirePackage{hypdoc}

```

5.6 Configuration and tweaks

`\MakePrivateLetters` A few more letters are “private” in a L^AT_EX3 programming environment.

```

460   \cs_gset:Npn \MakePrivateLetters
461   {
462     \char_set_catcode_letter:N \@
463     \char_set_catcode_letter:N \_
464     \char_set_catcode_letter:N \:
465   }

```

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

`CodelineNo` Some configurations which have to do with line numbering.

```

466   \setcounter{StandardModuleDepth}{1}
467   \@addtoreset{CodelineNo}{part}
468   \tl_replace_once:Nnn \theCodelineNo
469   { \HDorg@theCodelineNo }
470   { \textcolor[gray]{0.5} { \sffamily\tiny\arabic{CodelineNo} } }

```

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

`\verbatim` 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`:

```

471   \fvset{gobble=2}
472   \cs_gset_eq:NN \verbatim \Verbatim
473   \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.

```

474 \cs_set_protected:Npn \ifnot@excluded
475 {
476   \exp_args:Nxx \expanded@notin
477   { \c_backslash_str \tl_to_str:N \macro@namepart , }
478   { \exp_args:NV \tl_to_str:n \index@excludelist }
479 }

```

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

`\pdfstringnewline` 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.

`_codedoc_pdfstring_newline:w`

```

480 \cs_new:Npn \pdfstringnewline { : ~ }
481 \DeclareExpandableDocumentCommand
482 { \\_codedoc_pdfstring_newline:w } { s o m } { \pdfstringnewline #3 }
483 \pdfstringdefDisableCommands
484 { \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.

```

485 \setlength \textwidth { 385pt }
486 \addtolength \marginparwidth { 30pt }
487 \addtolength \oddsidemargin { 20pt }
488 \addtolength \evensidemargin { 20pt }

```

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

`\list` Customise lists.

`_codedoc_oldlist:nn`

```

489 \cs_new_eq:NN \\_codedoc_oldlist:nn \list
490 \cs_gset:Npn \list #1 #2
491 { \\_codedoc_oldlist:nn {#1} { #2 \dim_zero:N \listparindent } }
492 \setlength \parindent { 2em }
493 \setlength \itemindent { 0pt }
494 \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.

```

495 \tl_gset:Nn \partname {File}

```

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

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

```

496 \addtoreset{section}{part}
497 \cs_gset:Npn \l@section #1#2
498 {
499   \ifnum \c@tocdepth > \z@
500     \addpenalty\@secpenalty
501     \addvspace{1.0em \@plus\p@}
502     \setlength\@tempdima{2.5em} % was 1.5em
503     \begingroup
504       \parindent \z@ \rightskip \@pnumwidth
505       \parfillskip -\@pnumwidth
506       \leavevmode \bfseries
507       \advance\leftskip\@tempdima
508       \hskip -\leftskip
509       #1\nobreak\hfil \nobreak\hb@xt@\@pnumwidth{\hss #2}\par
510     \endgroup
511   \fi
512 }
513 \cs_gset:Npn \l@subsection
514 { \@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 `|l`).

```

515 \AtBeginDocument
516 {
517   \MakeShortVerb \"
518   \MakeShortVerb \l
519 }
520 \AtEndDocument
521 {
522   \DeleteShortVerb \"
523   \DeleteShortVerb \l
524 }

```

`\eTeX` Some commands for logos.

```

\IniTeX 525 \providecommand*\eTeX{\hologo{eTeX}}
\Lua    526 \providecommand*\IniTeX{\hologo{iniTeX}}
\LuaTeX 527 \providecommand*\Lua{Lua}
\pdfTeX 528 \providecommand*\LuaTeX{\hologo{LuaTeX}}
\XeTeX  529 \providecommand*\pdfTeX{\hologo{pdfTeX}}
\pTeX   530 \providecommand*\XeTeX{\hologo{XeTeX}}
\upTeX  531 \providecommand*\pTeX{\p{kern-.2em}\hologo{TeX}}
\epTeX  532 \providecommand*\upTeX{\up{kern-.2em}\hologo{TeX}}
\epTeX  533 \providecommand*\epTeX{\varepsilon\pTeX}
\epTeX  534 \providecommand*\eupTeX{\varepsilon\upTeX}
\epTeX  535 \providecommand*\ConTeXt{\hologo{ConTeXt}}

```


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

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

```

536 \DeclareDocumentCommand \cmd { 0{ } m }
537   { \__codedoc_cmd:no {#1} { \token_to_str:N #2 } }
538 \DeclareDocumentCommand \cs { 0{ } m }
539   { \__codedoc_cmd:no {#1} { \c_backslash_str #2 } }
540 \DeclareDocumentCommand \tn { 0{ } m }
541   {
542     \__codedoc_cmd:no
543       { module = TeX , replace = false , #1 }
544       { \c_backslash_str #2 }
545   }

```

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

\meta A document-level command.

```

546 \DeclareDocumentCommand \meta { m }
547   { \__codedoc_meta:n {#1} }

```

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

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

```

\__codedoc_pdfstring_cs:w
\__codedoc_pdfstring_meta:w
548 \DeclareExpandableDocumentCommand
549   { \__codedoc_pdfstring_cmd:w } { o m } { \token_to_str:N #2 }
550 \DeclareExpandableDocumentCommand
551   { \__codedoc_pdfstring_cs:w } { o m } { \textbackslash \tl_to_str:n {#2} }
552 \cs_new:Npn \__codedoc_pdfstring_meta:w #1
553   { < \tl_to_str:n {#1} > }
554 \pdfstringdefDisableCommands
555   {
556     \cs_set_eq:NN \cmd \__codedoc_pdfstring_cmd:w
557     \cs_set_eq:NN \cs \__codedoc_pdfstring_cs:w
558     \cs_set_eq:NN \tn \__codedoc_pdfstring_cs:w
559     \cs_set_eq:NN \meta \__codedoc_pdfstring_meta:w
560   }

```

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

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

```

561 \newcommand\Arg[1]
562   { \texttt{\char'\{ \meta{#1} \texttt{\char'\}} }
563 \providecommand\marg[1]{ \Arg{#1} }
564 \providecommand\oarg[1]{ \texttt[ \meta{#1} \texttt] }
565 \providecommand\parg[1]{ \texttt( \meta{#1} \texttt) }

```

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

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

```

\env 566 \DeclareRobustCommand \file {\nolinkurl}
\pkg 567 \DeclareRobustCommand \env {\texttt}
\cls 568 \DeclareRobustCommand \pkg {\textsf}
569 \DeclareRobustCommand \cls {\textsf}

```

(End definition for \file and others. These functions are documented on page 5.)

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

```

\EnableImplementation
\DisableDocumentation 570 \NewDocumentCommand \EnableDocumentation { }
\DisableImplementation 571 { \bool_gset_true:N \g__codedoc_typeset_documentation_bool }
572 \NewDocumentCommand \EnableImplementation { }
573 { \bool_gset_true:N \g__codedoc_typeset_implementation_bool }
574 \NewDocumentCommand \DisableDocumentation { }
575 { \bool_gset_false:N \g__codedoc_typeset_documentation_bool }
576 \NewDocumentCommand \DisableImplementation { }
577 { \bool_gset_false:N \g__codedoc_typeset_implementation_bool }

```

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

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

```

578 \NewDocumentEnvironment { documentation } { }
579 {
580   \bool_if:NTF \g__codedoc_typeset_documentation_bool
581     { \bool_set_false:N \l__codedoc_in_implementation_bool }
582     { \comment }
583 }
584 { \bool_if:NF \g__codedoc_typeset_documentation_bool { \endcomment } }
585 \NewDocumentEnvironment { implementation } { }
586 {
587   \bool_if:NTF \g__codedoc_typeset_implementation_bool
588     { \bool_set_true:N \l__codedoc_in_implementation_bool }
589     { \comment }
590 }
591 { \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.

```

592 \DeclareDocumentEnvironment { variable } { 0{} +v }
593 {
594   \bool_if:NTF \l__codedoc_in_implementation_bool
595     { \__codedoc_macro:nnw { var , #1 } {#2} }
596     { \__codedoc_function:nnw {#1} {#2} }
597 }
598 {
599   \bool_if:NTF \l__codedoc_in_implementation_bool
600     { \__codedoc_macro_end: }
601     { \__codedoc_function_end: }
602 }

```

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

```
603 \DeclareDocumentEnvironment { function } { 0{} +v }
604 { \__codedoc_function:nnw {#1} {#2} }
605 { \__codedoc_function_end: }
606 \DeclareDocumentEnvironment { macro } { 0{} +v }
607 { \__codedoc_macro:nnw {#1} {#2} }
608 { \__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.

```
609 \NewDocumentEnvironment { syntax } { }
610 { \__codedoc_syntax:w }
611 {
612   \__codedoc_syntax_end:
613   \ignorespacesafterend
614 }
```

texnote Used to describe information destined to T_EX experts only.

```
615 \NewDocumentEnvironment { texnote } { }
616 {
617   \endgraf
618   \vspace{3mm}
619   \small\textbf{\TeX-hackers~note:}
620 }
621 {
622   \vspace{3mm}
623 }
```

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

```
624 \NewDocumentEnvironment { arguments } { }
625 {
626   \enumerate [
627     nolistsep ,
628     label = \texttt{\#\arabic*} ~ : ,
629     labelsep = * ,
630   ]
631 }
632 {
633   \endenumerate
634 }
```

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

```
\CodedocExplainEXP 635 \NewDocumentCommand { \CodedocExplain } { }
\CodedocExplainREXP 636 { \CodedocExplainEXP \ \CodedocExplainREXP \ \CodedocExplainTF }
\CodedocExplainTF 637 \NewDocumentCommand { \CodedocExplainEXP } { }
638 {
639   \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{expstar}{}}%
640   \write \@auxout { \def \string \Codedoc@expstar { } }
641   \__codedoc_typeset_exp:\ indicates~fully~expandable~functions,~which~
642   can~be~used~within~an~\texttt{x}-type~argument~(in~plain~
```

```

643 \TeX{~terms,~inside~an~\cs{edef}},~as~well~as~within~an~
644 \texttt{f}-type~argument.
645 }
646 \NewDocumentCommand { \CodedocExplainREXP } { }
647 {
648 \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{rexpstar}{}}%
649 \write \@auxout { \def \string \Codedoc@rexpstar { } }
650 \__codedoc_typeset_rexp:\ indicates~
651 restricted~expandable~functions,~which~can~be~used~within~an~
652 \texttt{x}-type~argument~but~cannot~be~fully~expanded~within~an~
653 \texttt{f}-type~argument.
654 }
655 \NewDocumentCommand { \CodedocExplainTF } { }
656 {
657 \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{explTF}{}}%
658 \write \@auxout { \def \string \Codedoc@explTF { } }
659 \__codedoc_typeset_TF:\ indicates~conditional~(\texttt{if})~functions~
660 whose~variants~with~\texttt{T},~\texttt{F}~and~\texttt{TF}~
661 argument~specifiers~expect~different~
662 \enquote{true}/\enquote{false}~branches.
663 }

```

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

5.9 Implementing text markup

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

```

664 \keys_define:nn { l3doc/cmd }
665 {
666 index .tl_set:N = \l__codedoc_cmd_index_tl ,
667 module .tl_set:N = \l__codedoc_cmd_module_tl ,
668 no-index .bool_set:N = \l__codedoc_cmd_noindex_bool ,
669 replace .bool_set:N = \l__codedoc_cmd_replace_bool ,
670 }

```

`__codedoc_cmd:nn` 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.

```

671 \cs_new_protected:Npn \__codedoc_cmd:nn #1#2
672 {
673 \bool_set_false:N \l__codedoc_cmd_noindex_bool
674 \bool_set_true:N \l__codedoc_cmd_replace_bool
675 \tl_set:Nn \l__codedoc_cmd_index_tl { \q_no_value }
676 \tl_set:Nn \l__codedoc_cmd_module_tl { \q_no_value }
677 \keys_set:nn { l3doc/cmd } {#1}
678 \tl_set:Nn \l__codedoc_cmd_tl {#2}
679 \bool_if:NT \l__codedoc_cmd_replace_bool

```

```

680 {
681   \tl_set_rescan:Nnn \l__codedoc_tmpb_tl { } { _ }
682   \tl_replace_all:Non \l__codedoc_cmd_tl \l__codedoc_tmpb_tl { _ }
683   \__codedoc_replace_at_at:N \l__codedoc_cmd_tl
684   \tl_replace_all:Nno \l__codedoc_cmd_tl { _ } \l__codedoc_tmpb_tl
685 }

```

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.

```

686 \mode_if_math:T { \mbox }
687 {
688   \verbatim@font
689   \__codedoc_if_almost_str:VT \l__codedoc_cmd_tl
690   {
691     \tl_set:Nx \l__codedoc_cmd_tl { \tl_to_str:N \l__codedoc_cmd_tl }
692     \bool_if:NT \g__codedoc_cs_break_bool
693     {
694       \regex_replace_all:nnN
695       {([^\[])([^\_])}
696       {\1\c{BreakableUnderscore}\2}
697       \l__codedoc_cmd_tl
698     }
699   }
700   \tl_replace_all:Nnn \l__codedoc_cmd_tl { ~ } { \@xobeysp }
701   \l__codedoc_cmd_tl
702   \@
703 }

```

Indexing

```

704 \bool_if:NF \l__codedoc_cmd_noindex_bool
705 {
706   \quark_if_no_value:NF \l__codedoc_cmd_index_tl
707   {
708     \tl_set:Nx \l__codedoc_cmd_tl
709     { \c_backslash_str \exp_not:o { \l__codedoc_cmd_index_tl } }
710   }
711
712   \exp_args:No \__codedoc_key_get:n { \l__codedoc_cmd_tl }
713   \quark_if_no_value:NF \l__codedoc_cmd_module_tl
714   {
715     \tl_set:Nx \l__codedoc_index_module_tl
716     { \tl_to_str:N \l__codedoc_cmd_module_tl }
717   }
718   \__codedoc_special_index_module:oonN
719   { \l__codedoc_index_key_tl }
720   { \l__codedoc_index_macro_tl }
721   { \l__codedoc_index_module_tl }
722   { usage }
723   \l__codedoc_index_internal_bool
724 }
725 }

```

```
726 \cs_generate_variant:Nn \__codedoc_cmd:nn { no }
```

(End definition for `__codedoc_cmd:nn`.)

`__codedoc_meta: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`.

```
727 \cs_new_protected:Npn \__codedoc_meta:n #1
728 {
729   \tl_set:Nn \l__codedoc_tmpa_tl {#1}
730   \tl_map_inline:nn
731     { { 3 } { 4 } { 7 } { 8 } { 11 } { 12 } { 13 } }
732     {
733       \tl_set_rescan:Nnn \l__codedoc_tmpb_tl
734         { \char_set_catcode:nn { ' _ } {##1} } { _ }
735       \tl_replace_all:Non \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
736         { \__codedoc_ensuremath_sb:n }
737     }
738   \exp_args:NV \__codedoc_meta_original:n \l__codedoc_tmpa_tl
739 }
740 \cs_new_protected:Npn \__codedoc_ensuremath_sb:n #1
741 { \ensuremath { \sb {#1} } }
742 \cs_new_protected:Npn \__codedoc_meta_original:n #1
743 {
744   \ensuremath \langle
745   \mode_if_math:T { \nfss@text }
746   {
747     \meta@font@select
748     \edef \meta@hyphen@restore
749       { \hyphenchar \the \font \the \hyphenchar \font }
750     \hyphenchar \font \m@ne
751     \language \l@nohyphenation
752     #1 \/  

753     \meta@hyphen@restore
754   }
755   \ensuremath \rangle
756 }
```

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

```
\__codedoc_typeset_rexp:
\__codedoc_typeset_TF:
\__codedoc_typeset_aux:n
757 \cs_new_protected:Npn \__codedoc_typeset_exp:
758 {
759   \cs_if_exist:NTF \Codedoc@expstar
760     { \hyperlink { expstar } }
761     { \mbox {
762       { $\star$ }
763     } }
764 \cs_new_protected:Npn \__codedoc_typeset_rexp:
765 {
```

```

766 \cs_if_exist:NTF \Codedoc@rexpstar
767 { \hyperlink { rexpstar } }
768 { \mbox }
769 { \ding { 73 } } % hollow star
770 }
771 \cs_new_protected:Npn \__codedoc_typeset_TF:
772 {
773 \cs_if_exist:NTF \Codedoc@explTF
774 { \hyperlink { explTF } }
775 { \mbox }
776 {
777 \color{black}
778 \itshape TF
779 \makebox[Opt][r]
780 {
781 \cs_if_exist:NT \Codedoc@explTF { \color{red} }
782 \underline { \phantom{\itshape TF} \kern-0.1em }
783 }
784 }
785 }
786 \cs_new_protected:Npn \__codedoc_typeset_aux:n #1
787 {
788 { \color[gray]{0.5} #1 }
789 }

```

(End definition for __codedoc_typeset_exp: and others.)

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

```

790 \cs_new_protected:Npn \__codedoc_get_hyper_target:nN #1#2
791 {
792 \tl_set:Nx #2 { \tl_to_str:n {#1} }
793 \tl_replace_all:Nxn #2 { \c_underscore_str } { / }
794 \tl_remove_all:Nx #2 { \c_backslash_str }
795 \tl_put_left:Nn #2 { doc/function// }
796 }
797 \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.

```

798 \cs_new_protected:Npn \__codedoc_names_get_seq:nN #1#2
799 {
800 \tl_set:Nx \l__codedoc_tmpa_tl { \tl_to_str:n {#1} }
801 \bool_if:NTF \l__codedoc_names_verb_bool
802 {
803 \seq_clear:N #2

```

```

804     \seq_put_right:NV #2 \l__codedoc_tmpa_tl
805   }
806   {
807     \tl_remove_all:Nx \l__codedoc_tmpa_tl
808     { \iow_char:N \^^M \c_percent_str }
809     \tl_remove_all:Nx \l__codedoc_tmpa_tl { \tl_to_str:n { ^ ^ A } }
810     \tl_remove_all:Nx \l__codedoc_tmpa_tl { \iow_char:N \^^I }
811     \tl_remove_all:Nx \l__codedoc_tmpa_tl { \iow_char:N \^^M }
812     \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
813     \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
814     \exp_args:NNx \seq_set_from_clist:Nn #2
815     { \tl_to_str:N \l__codedoc_tmpa_tl }
816   }
817 }

```

(End definition for __codedoc_names_get_seq:nN.)

__codedoc_names_parse: 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>*, *<signature₁>*, *<signature₂>* 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.

```

818 \cs_new_protected:Npn \__codedoc_names_parse:
819 {
820   \tl_clear:N \l__codedoc_names_block_tl
821   \seq_map_function:NN
822     \l__codedoc_names_seq
823     \__codedoc_names_parse_one:n
824 }
825 \cs_new_protected:Npn \__codedoc_names_parse_one:n #1
826 {
827   \__codedoc_split_function_do:nn {#1}
828   { \__codedoc_names_parse_one_aux:nnNn }
829   {#1}
830 }
831 \cs_new_protected:Npn \__codedoc_names_parse_one_aux:nnNn #1#2#3#4
832 {
833   \bool_if:NTF #3
834   {
835     \tl_if_head_eq_charcode:nNTF {#2} :
836     { \__codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: } }
837     {
838       \exp_args:Nx \__codedoc_names_parse_aux:nnn
839       { \__codedoc_base_form_aux:nnN {#1} {#2} #3 }
840       {#1} {#2}
841     }
842   }
843   {
844     \bool_if:NT \l__codedoc_macro_TF_bool
845     { \msg_error:nnx { l3doc } { no-signature-TF } {#4} }

```



```

846     \_codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: }
847   }
848 }
849 \cs_new_protected:Npn \_codedoc_names_parse_aux:nnn #1
850 { \exp_args:Nc \_codedoc_names_parse_aux:Nnn { \_codedoc_lseq_name:n {#1} } }
851 \cs_new_protected:Npn \_codedoc_names_parse_aux:Nnn #1#2#3
852 {
853   \tl_if_in:NnF \l__codedoc_names_block_tl {#1}
854   {
855     \tl_put_right:Nn \l__codedoc_names_block_tl {#1}
856     \seq_clear_new:N #1
857     \seq_put_right:Nn #1 {#2}
858   }
859   \seq_put_right:Nn #1 {#3}
860 }

```

(End definition for `_codedoc_names_parse:` and `_codedoc_names_parse_one:n`.)

`_codedoc_names_typeset:` 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`.

```

861 \cs_new_protected:Npn \_codedoc_names_typeset:
862 {
863   \tl_map_function:NN \l__codedoc_names_block_tl
864   \_codedoc_names_typeset_auxi:n
865 }
866 \cs_new_protected:Npn \_codedoc_names_typeset_auxi:n #1
867 {
868   \seq_pop:NN #1 \l__codedoc_tmpa_tl
869   \tl_gset_eq:NN \g__codedoc_base_name_tl \l__codedoc_tmpa_tl
870   \tl_greplace_all:Nno \g__codedoc_base_name_tl
871   { ~ } { \c_catcode_other_space_tl }
872   \seq_get:NN #1 \l__codedoc_tmpa_tl
873   \str_if_eq:VnTF \l__codedoc_tmpa_tl { \scan_stop: }
874   {
875     \seq_gclear:N \g__codedoc_variants_seq
876     \_codedoc_names_typeset_auxii:x { \g__codedoc_base_name_tl }
877   }
878   {
879     \seq_gset_eq:NN \g__codedoc_variants_seq #1
880     \seq_gpop:NN \g__codedoc_variants_seq \l__codedoc_tmpb_tl
881     \_codedoc_names_typeset_auxii:x
882     { \g__codedoc_base_name_tl : \l__codedoc_tmpb_tl }
883   }
884 }

```

(End definition for `_codedoc_names_typeset:` and `_codedoc_names_typeset_auxi:n`.)

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

```

885 \cs_new_protected:Npn \__codedoc_names_typeset_auxii:n #1
886 {
887   \bool_if:NT \l__codedoc_macro_pTF_bool
888   {
889     \__codedoc_names_typeset_block:xN
890     { \__codedoc_predicate_from_base:n {#1} }
891     \c_false_bool
892   }
893   \bool_if:NT \l__codedoc_macro_noTF_bool
894   { \__codedoc_names_typeset_block:nN {#1} \c_false_bool }
895   \__codedoc_names_typeset_block:nN {#1} \l__codedoc_macro_TF_bool
896 }
897 \cs_generate_variant:Nn \__codedoc_names_typeset_auxii:n { x }

```

(End definition for `__codedoc_names_typeset_auxii:n`.)

`_codedoc_names_typeset_block:nN` Names in function and macro environments are typeset differently. To distinguish the
`_codedoc_names_typeset_block:xN` 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.

```

898 \cs_new_protected:Npn \__codedoc_names_typeset_block:nN
899 {
900   \int_compare:nNnTF \l__codedoc_nested_macro_int = 0
901   { \__codedoc_typeset_function_block:nN }
902   { \__codedoc_macro_typeset_block:nN }
903 }
904 \cs_generate_variant:Nn \__codedoc_names_typeset_block:nN { x }

```

(End definition for `__codedoc_names_typeset_block:nN`.)

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

```

905 \prg_new_conditional:Npnn \__codedoc_if_macro_internal:n #1 { p , T , F , TF }
906 {
907   \bool_if:NTF \l__codedoc_macro_internal_set_bool
908   {
909     \bool_if:NTF \l__codedoc_macro_internal_bool
910     { \prg_return_true: } { \prg_return_false: }
911   }
912   {
913     \tl_if_empty:FTF
914     {
915       \exp_after:wN \__codedoc_if_macro_internal_aux:w
916       \tl_to_str:n { #1 ~ __ }
917     }
918     { \prg_return_false: } { \prg_return_true: }
919   }
920 }
921 \exp_last_unbraced:NNNN
922 \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.

```

923 \cs_new_protected:Npn \_codedoc_names_block_base_map:N #1
924 {
925   \tl_map_inline:Nn \l__codedoc_names_block_tl
926   {
927     \group_begin:
928     \seq_set_eq:NN \l__codedoc_tmpa_seq ##1
929     \seq_pop:NN \l__codedoc_tmpa_seq \l__codedoc_tmpa_tl
930     \seq_get:NN \l__codedoc_tmpa_seq \l__codedoc_tmpb_tl
931     \exp_args:NNx
932     \group_end:
933     #1
934     {
935       \l__codedoc_tmpa_tl
936       \str_if_eq:VnF \l__codedoc_tmpb_tl { \scan_stop: }
937       { : \l__codedoc_tmpb_tl }
938       \bool_if:NT \l__codedoc_macro_TF_bool { TF }
939     }
940   }
941 }
```

(End definition for `_codedoc_names_block_base_map:N`.)

5.9.2 The function environment

```

942 \keys_define:nn { l3doc/function }
943 {
944   TF .value_forbidden:n = true ,
945   TF .code:n =
946   {
947     \bool_set_true:N \l__codedoc_macro_TF_bool
948   } ,
949   EXP .value_forbidden:n = true ,
950   EXP .code:n =
951   {
952     \bool_set_true:N \l__codedoc_macro_EXP_bool
953     \bool_set_false:N \l__codedoc_macro_rEXP_bool
954   } ,
955   rEXP .value_forbidden:n = true ,
956   rEXP .code:n =
957   {
958     \bool_set_false:N \l__codedoc_macro_EXP_bool
959     \bool_set_true:N \l__codedoc_macro_rEXP_bool
960   } ,
961   pTF .value_forbidden:n = true ,
962   pTF .code:n =
963   {
964     \bool_set_true:N \l__codedoc_macro_pTF_bool
965     \bool_set_true:N \l__codedoc_macro_TF_bool
```

```

966     \bool_set_true:N \l__codedoc_macro_EXP_bool
967     \bool_set_false:N \l__codedoc_macro_rEXP_bool
968   } ,
969   noTF .value_forbidden:n = true ,
970   noTF .code:n =
971   {
972     \bool_set_true:N \l__codedoc_macro_noTF_bool
973     \bool_set_true:N \l__codedoc_macro_TF_bool
974   } ,
975   added .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_added_tl {#1} } ,
976   updated .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_updated_tl {#1} } ,
977   deprecated .code:n = { \__codedoc_deprecated_on:n {#1} } ,
978   tested .code:n = { } ,
979   label .code:n =
980   {
981     \clist_set:Nn \l__codedoc_function_label_clist {#1}
982     \bool_set_true:N \l__codedoc_no_label_bool
983   } ,
984   verb .value_forbidden:n = true ,
985   verb .bool_set:N = \l__codedoc_names_verb_bool ,
986   module .tl_set:N = \l__codedoc_override_module_tl ,
987 }

```

`__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).

```

988 \cs_new_protected:Npn \__codedoc_date_set:Nn #1#2
989 {
990   \tl_set:Nn #1 {#2}
991   \regex_replace_once:nnNF
992   { \A(\d\d\d\d)[-/](\d\d?)[-/](\d\d?)\Z } { \1-\2-\3 } #1
993   {
994     \msg_error:nnn { l3doc } { date-format } {#2}
995     \tl_set:Nn #1 { 1970-01-01 }
996   }
997 }
998 \cs_new_protected:Npn \__codedoc_date_set_past:Nn #1#2
999 {
1000   \__codedoc_date_set:Nn #1 {#2}
1001   \exp_args:No \__codedoc_date_compare:nNnT
1002   {#1} > { \c_sys_year_int - \c_sys_month_int - \c_sys_day_int }
1003   {
1004     \msg_error:nnxx { l3doc } { future-date }
1005     { \tl_to_str:N \l__codedoc_macro_argument_tl }
1006     {#1}
1007   }
1008 }

```

(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 ...”.

```

1009 \cs_new_protected:Npn \__codedoc_deprecated_on:n #1

```

```

1010 {
1011   \__codedoc_date_set:Nn \l__codedoc_tmpa_tl {#1}
1012   \exp_args:No \__codedoc_date_compare:nNnT
1013     { \l__codedoc_tmpa_tl } < { \c_sys_year_int - \c_sys_month_int - \c_sys_day_int }
1014     {
1015       \msg_error:nnxx { l3doc } { deprecated-function }
1016       { \tl_to_str:N \l__codedoc_macro_argument_tl }
1017       { \l__codedoc_tmpa_tl }
1018     }
1019   \bool_set_true:N \l__codedoc_macro_internal_bool
1020   \bool_set_true:N \l__codedoc_macro_internal_set_bool
1021 }

```

(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.

```

1022 \cs_new_protected:Npn \__codedoc_function:nnw #1#2
1023 {
1024   \__codedoc_function_typeset_start:
1025   \__codedoc_function_init:
1026   \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
1027   \keys_set:nn { l3doc/function } {#1}
1028   \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
1029   \__codedoc_names_parse:
1030   \__codedoc_function_typeset:
1031   \__codedoc_function_reset:
1032   \__codedoc_function_descr_start:w
1033 }
1034 \cs_new_protected:Npn \__codedoc_function_end:
1035 {
1036   \__codedoc_function_descr_stop:
1037   \__codedoc_function_assemble:
1038   \__codedoc_function_typeset_stop:
1039 }

```

(End definition for __codedoc_function:nnw and __codedoc_function_end:.)

__codedoc_function_typeset_start: 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.

```

1040 \cs_new_protected:Npn \__codedoc_function_typeset_start:
1041 {
1042   \par \bigskip \noindent
1043 }

```

```

1044 \cs_new_protected:Npn \__codedoc_function_typeset_stop:
1045 {
1046   \par
1047   \dim_set:Nn \prevdepth { \box_dp:N \l__codedoc_descr_coffin }
1048   \allowbreak
1049 }

```

(End definition for __codedoc_function_typeset_start: and __codedoc_function_typeset_stop:.)

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

```

1050 \cs_new_protected:Npn \__codedoc_function_init:
1051 {
1052   \box_if_empty:NF \g__codedoc_syntax_box
1053   { \msg_error:nn { l3doc } { syntax-nested-function } }
1054   \coffin_clear:N \l__codedoc_descr_coffin
1055   \box_gclear:N \g__codedoc_syntax_box
1056   \coffin_clear:N \l__codedoc_syntax_coffin
1057   \coffin_clear:N \l__codedoc_functions_coffin
1058   \bool_set_false:N \l__codedoc_macro_TF_bool
1059   \bool_set_false:N \l__codedoc_macro_pTF_bool
1060   \bool_set_false:N \l__codedoc_macro_noTF_bool
1061   \bool_set_false:N \l__codedoc_macro_EXP_bool
1062   \bool_set_false:N \l__codedoc_macro_rEXP_bool
1063   \bool_set_false:N \l__codedoc_no_label_bool
1064   \bool_set_false:N \l__codedoc_names_verb_bool
1065   \bool_set_true:N \l__codedoc_in_function_bool
1066   \clist_clear:N \l__codedoc_function_label_clist
1067   \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
1068   \char_set_active_eq:NN \< \__codedoc_shorthand_meta:
1069   \char_set_catcode_active:N \<
1070 }

```

(End definition for __codedoc_function_init:.)

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

```

\__codedoc_shorthand_meta:w
1071 \cs_new_protected:Npn \__codedoc_shorthand_meta:
1072 { \mode_if_math:TF { < } { \__codedoc_shorthand_meta:w } }
1073 \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.

```

1074 \cs_new_protected:Npn \__codedoc_function_reset:
1075 {
1076   \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
1077 }

```

(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.

```

1078 \cs_new_protected:Npn \__codedoc_function_typeset:
1079 {

```

```

1080 \dim_zero:N \l__codedoc_trial_width_dim
1081 \hcoffin_set:Nn \l__codedoc_functions_coffin { \__codedoc_typeset_functions: }
1082 \dim_set:Nn \l__codedoc_trial_width_dim
1083 { \box_wd:N \l__codedoc_functions_coffin }
1084 \bool_set:Nn \l__codedoc_long_name_bool
1085 { \dim_compare_p:nNn \l__codedoc_trial_width_dim > \marginparwidth }
1086 }

```

(End definition for __codedoc_function_typeset:.)

__codedoc_function_descr_start:w 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).

```

1087 \cs_new_protected:Npn \__codedoc_function_descr_start:w
1088 {
1089   \vcoffin_set:Nnw \l__codedoc_descr_coffin { \textwidth }
1090   \noindent \ignorespaces
1091 }
1092 \cs_new_protected:Npn \__codedoc_function_descr_stop:
1093 { \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.

```

1094 \cs_new_protected:Npn \__codedoc_function_assemble:
1095 {
1096   \hcoffin_set:Nn \l__codedoc_syntax_coffin
1097   { \box_use_drop:N \g__codedoc_syntax_box }
1098   \bool_if:NTF \l__codedoc_long_name_bool
1099   {
1100     \coffin_join:NnnNnnnn
1101     \l__codedoc_output_coffin {hc} {vc}
1102     \l__codedoc_syntax_coffin {l} {T}
1103     {0pt} {0pt}
1104     \coffin_join:NnnNnnnn
1105     \l__codedoc_output_coffin {l} {t}
1106     \l__codedoc_functions_coffin {r} {t}
1107     {-\marginparsep} {0pt}
1108     \coffin_join:NnnNnnnn
1109     \l__codedoc_output_coffin {l} {b}
1110     \l__codedoc_descr_coffin {l} {t}
1111     {0.75\marginparwidth + \marginparsep} {-\medskipamount}
1112     \coffin_typeset:Nnnnn \l__codedoc_output_coffin
1113     {\l__codedoc_descr_coffin-l} {\l__codedoc_descr_coffin-t}
1114     {0pt} {0pt}
1115   }
1116   {
1117     \coffin_join:NnnNnnnn
1118     \l__codedoc_output_coffin {hc} {vc}
1119     \l__codedoc_syntax_coffin {l} {t}

```

```

1120         {Opt} {Opt}
1121     \coffin_join:NnnNnnnn
1122     \l__codedoc_output_coffin {l} {b}
1123     \l__codedoc_descr_coffin {l} {t}
1124     {Opt} {-\medskipamount}
1125     \coffin_join:NnnNnnnn
1126     \l__codedoc_output_coffin {l} {t}
1127     \l__codedoc_functions_coffin {r} {t}
1128     {-\marginparsep} {Opt}
1129     \coffin_typeset:Nnnnn \l__codedoc_output_coffin
1130     {\l__codedoc_syntax_coffin-l} {\l__codedoc_syntax_coffin-T}
1131     {Opt} {Opt}
1132 }
1133 }

```

(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.

```

1134 \cs_new_protected:Npn \__codedoc_typeset_functions:
1135 {
1136     \small\ttfamily
1137     \HD@savedestfalse
1138     \HD@target
1139     \Hy@MakeCurrentHref { HD. \int_use:N \c@HD@hypercount }
1140     \begin{tabular} [t] { @{} l @{} >{\hspace{\tabcolsep}} r @{} }
1141         \toprule
1142         \__codedoc_function_extra_labels:
1143         \__codedoc_names_typeset:
1144         \__codedoc_typeset_dates:
1145         \bottomrule
1146     \end{tabular}
1147     \normalfont\normalsize
1148 }

```

(End definition for __codedoc_typeset_functions:.)

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

__codedoc_typeset_function_block:xN 1149 \cs_new_protected:Npn __codedoc_typeset_function_block:nN #1#2

```

1150 {
1151     \__codedoc_function_index:x
1152     { #1 \bool_if:NT #2 { \tl_to_str:n {TF} } }
1153     \__codedoc_function_label:xN {#1} #2
1154     #1
1155     \bool_if:NT #2 { \__codedoc_typeset_TF: }
1156     \__codedoc_typeset_expandability:
1157     \seq_if_empty:NF \g__codedoc_variants_seq
1158     { \__codedoc_typeset_variant_list:nN {#1} #2 }
1159     \\
1160 }
1161 \cs_generate_variant:Nn \__codedoc_typeset_function_block:nN { x }
1162 \cs_new_protected:Npn \__codedoc_function_index:n #1

```



```

1163 {
1164   \seq_gput_right:Nn \g_doc_functions_seq {#1}
1165   \__codedoc_special_index:nn {#1} { usage }
1166 }
1167 \cs_generate_variant:Nn \__codedoc_function_index:n { x }
1168 \cs_new_protected:Npn \__codedoc_typeset_expandability:
1169 {
1170   &
1171   \bool_if:NT \l__codedoc_macro_EXP_bool { \__codedoc_typeset_exp: }
1172   \bool_if:NT \l__codedoc_macro_rEXP_bool { \__codedoc_typeset_rexp: }
1173 }

```

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

```

1174 \cs_new_protected:Npn \__codedoc_typeset_variant_list:nN #1#2
1175 {
1176   \\\
1177   \__codedoc_typeset_aux:n { \__codedoc_get_function_name:n {#1} }
1178   :
1179   \int_compare:nTF { \seq_count:N \g__codedoc_variants_seq == 1 }
1180   { \seq_use:Nn \g__codedoc_variants_seq { } }
1181   {
1182     \textrm(
1183       \seq_use:Nn \g__codedoc_variants_seq { \textrm| }
1184     \textrm)
1185   }
1186   \bool_if:NT #2 { \__codedoc_typeset_TF: }
1187   \__codedoc_typeset_expandability:
1188 }

```

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

```

1189 \cs_new_protected:Npn \__codedoc_function_extra_labels:
1190 {
1191   \bool_if:NT \l__codedoc_no_label_bool
1192   {
1193     \clist_map_inline:Nn \l__codedoc_function_label_clist
1194     {
1195       \__codedoc_get_hyper_target:oN { \token_to_str:N ##1 }
1196       \l__codedoc_tmpa_tl
1197       \exp_args:No \label { \l__codedoc_tmpa_tl }
1198     }
1199   }
1200 }
1201 \cs_new_protected:Npn \__codedoc_function_label:nN #1#2
1202 {
1203   \bool_if:NF \l__codedoc_no_label_bool
1204   {
1205     \__codedoc_get_hyper_target:xN
1206     {
1207       \exp_not:n {#1}
1208       \bool_if:NT #2 { \tl_to_str:n {TF} }
1209     }
1210     \l__codedoc_tmpa_tl
1211     \exp_args:No \label { \l__codedoc_tmpa_tl }
1212   }

```

```

1213 }
1214 \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.

```

1215 \cs_new:Npn \__codedoc_typeset_dates:
1216 {
1217   \bool_lazy_and:nnF
1218     { \tl_if_empty_p:N \l__codedoc_date_added_tl }
1219     { \tl_if_empty_p:N \l__codedoc_date_updated_tl }
1220     { \midrule }
1221   \tl_if_empty:NF \l__codedoc_date_added_tl
1222   {
1223     \multicolumn { 2 } { @{} r @{} }
1224     { \scriptsize New: \, \l__codedoc_date_added_tl } \\
1225   }
1226
1227   \tl_if_empty:NF \l__codedoc_date_updated_tl
1228   {
1229     \multicolumn { 2 } { @{} r @{} }
1230     { \scriptsize Updated: \, \l__codedoc_date_updated_tl } \\
1231   }
1232 }

```

(End definition for `__codedoc_typeset_dates:.`)

`__codedoc_syntax:w`
`__codedoc_syntax_end:`

Implement the syntax environment.

```

1233 \dim_new:N \l__codedoc_syntax_dim
1234 \cs_new_protected:Npn \__codedoc_syntax:w
1235 {
1236   \box_if_empty:NF \g__codedoc_syntax_box
1237   { \msg_error:nn { l3doc } { multiple-syntax } }
1238   \dim_set:Nn \l__codedoc_syntax_dim
1239   {
1240     \textwidth
1241     \bool_if:NT \l__codedoc_long_name_bool
1242     { + 0.75 \marginparwidth - \l__codedoc_trial_width_dim }
1243   }
1244   \hbox_gset:Nw \g__codedoc_syntax_box
1245   \small \ttfamily
1246   \arrayrulecolor{white}
1247   \begin{tabular} { @{} l @{} }
1248     \toprule
1249     \begin{minipage}[t]{\l__codedoc_syntax_dim}
1250       \raggedright
1251       \obeyspaces
1252       \obeylines
1253     }
1254   \cs_new_protected:Npn \__codedoc_syntax_end:
1255   {
1256     \end{minipage}
1257     \end{tabular}

```

```

1258     \arrayrulecolor{black}
1259 \hbox_gset_end:
1260 \bool_if:NF \l__codedoc_in_function_bool
1261 {
1262     \begin{quote}
1263         \mode_leave_vertical:
1264         \box_use_drop:N \g__codedoc_syntax_box
1265     \end{quote}
1266 }
1267 }

```

(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.

```

1268 \keys_define:nn { l3doc/macro }
1269 {
1270     aux .value_forbidden:n = true ,
1271     aux .code:n =
1272     {
1273         \msg_warning:nnnn { l3doc } { deprecated-option }
1274         { aux } { function/macro }
1275     } ,
1276     internal .value_forbidden:n = true ,
1277     internal .code:n =
1278     {
1279         \bool_set_true:N \l__codedoc_macro_internal_bool
1280         \bool_set_true:N \l__codedoc_macro_internal_set_bool
1281     } ,
1282     int .value_forbidden:n = true ,
1283     int .code:n =
1284     {
1285         \bool_set_true:N \l__codedoc_macro_internal_bool
1286         \bool_set_true:N \l__codedoc_macro_internal_set_bool
1287     } ,
1288     var .value_forbidden:n = true ,
1289     var .code:n =
1290     { \bool_set_true:N \l__codedoc_macro_var_bool } ,
1291     TF .value_forbidden:n = true ,
1292     TF .code:n =
1293     { \bool_set_true:N \l__codedoc_macro_TF_bool } ,
1294     pTF .value_forbidden:n = true ,
1295     pTF .code:n =
1296     {
1297         \bool_set_true:N \l__codedoc_macro_TF_bool
1298         \bool_set_true:N \l__codedoc_macro_pTF_bool
1299         \bool_set_true:N \l__codedoc_macro_EXP_bool
1300         \bool_set_false:N \l__codedoc_macro_rEXP_bool
1301     } ,
1302     noTF .value_forbidden:n = true ,
1303     noTF .code:n =
1304     {

```

```

1305     \bool_set_true:N \l__codedoc_macro_TF_bool
1306     \bool_set_true:N \l__codedoc_macro_noTF_bool
1307   } ,
1308   EXP .value_forbidden:n = true ,
1309   EXP .code:n =
1310   {
1311     \bool_set_true:N \l__codedoc_macro_EXP_bool
1312     \bool_set_false:N \l__codedoc_macro_rEXP_bool
1313   } ,
1314   rEXP .value_forbidden:n = true ,
1315   rEXP .code:n =
1316   {
1317     \bool_set_false:N \l__codedoc_macro_EXP_bool
1318     \bool_set_true:N \l__codedoc_macro_rEXP_bool
1319   } ,
1320   tested .code:n =
1321   {
1322     \bool_set_true:N \l__codedoc_macro_tested_bool
1323   } ,
1324   added .code:n = {} , % TODO
1325   updated .code:n = {} , % TODO
1326   deprecated .code:n = { \__codedoc_deprecated_on:n {#1} } ,
1327   verb .bool_set:N = \l__codedoc_names_verb_bool ,
1328   module .tl_set:N = \l__codedoc_override_module_tl ,
1329   documented-as .tl_set:N = \l__codedoc_macro_documented_tl ,
1330   do-not-index .value_required:n = true ,
1331   do-not-index .tl_set:N = \l__codedoc_macro_do_not_index_tl ,
1332   % do-not-index .default:n = \q_no_value ,
1333 }

```

`__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.

```

1334 \cs_new_protected:Npn \__codedoc_macro:nnw #1#2
1335 {
1336   \__codedoc_macro_init:
1337   \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
1338   \keys_set:nn { l3doc/macro } {#1}
1339   \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
1340   \__codedoc_names_parse:
1341   \__codedoc_macro_exclude_index:
1342   \__codedoc_macro_save_names:
1343   \__codedoc_names_typeset:
1344   \__codedoc_macro_dump:
1345   \__codedoc_macro_reset:
1346 }

```

(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).

```

1347 \cs_new_protected:Npn \__codedoc_macro_init:

```

```

1348 {
1349   \int_incr:N \l__codedoc_nested_macro_int
1350   \bool_set_false:N \l__codedoc_macro_internal_bool
1351   \bool_set_false:N \l__codedoc_macro_internal_set_bool
1352   \bool_set_false:N \l__codedoc_macro_TF_bool
1353   \bool_set_false:N \l__codedoc_macro_pTF_bool
1354   \bool_set_false:N \l__codedoc_macro_noTF_bool
1355   \bool_set_false:N \l__codedoc_macro_EXP_bool
1356   \bool_set_false:N \l__codedoc_macro_rEXP_bool
1357   \bool_set_false:N \l__codedoc_macro_var_bool
1358   \bool_set_false:N \l__codedoc_macro_tested_bool
1359   \bool_set_false:N \l__codedoc_names_verb_bool
1360   \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
1361   \tl_clear:N \l__codedoc_macro_documented_tl
1362   \cs_set_eq:NN \testfile \__codedoc_print_testfile:n
1363   \box_clear:N \l__codedoc_macro_index_box
1364   \vbox_set:Nn \l__codedoc_macro_box
1365     {
1366       \hbox:n
1367         {
1368           \strut
1369           \int_compare:nNnT \l__codedoc_macro_int = 0
1370             { \HD@target }
1371         }
1372       \vskip \int_eval:n { \l__codedoc_macro_int - 1 } \baselineskip
1373     }
1374 }

```

(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.

```

1375 \cs_new_protected:Npn \__codedoc_macro_reset:
1376 {
1377   \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
1378 }

```

(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.

```

1379 \cs_new_protected:Npn \__codedoc_macro_save_names:
1380 {
1381   \tl_if_empty:NTF \l__codedoc_macro_documented_tl
1382     { \__codedoc_names_block_base_map:N \__codedoc_macro_save_names_aux:n }
1383     {
1384       \seq_gput_right:Nf \g__codedoc_nested_names_seq
1385       { \exp_after:wN \token_to_str:N \l__codedoc_macro_documented_tl }
1386     }
1387 }
1388 \cs_new_protected:Npn \__codedoc_macro_save_names_aux:n #1
1389 { \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.

```

1390 \cs_new_protected:Npn \__codedoc_macro_exclude_index:
1391 {
1392   \tl_if_empty:NF \l__codedoc_macro_do_not_index_tl
1393   {
1394     \tl_set_rescan:NnV \l__codedoc_macro_do_not_index_tl
1395     { \MakePrivateLetters \catcode'\12 }
1396     \l__codedoc_macro_do_not_index_tl
1397     \exp_args:NNV \seq_set_from_clist:Nn
1398     \l__codedoc_tmpa_seq \l__codedoc_macro_do_not_index_tl
1399     \tl_set:Nx \l__codedoc_macro_do_not_index_tl
1400     { \seq_use:Nn \l__codedoc_tmpa_seq { , } }
1401     \exp_args:NV \DoNotIndex \l__codedoc_macro_do_not_index_tl
1402   }
1403 }

```

(End definition for `__codedoc_macro_exclude_index:`)

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

```

1404 \cs_new_protected:Npn \__codedoc_macro_dump:
1405 {
1406   \topsep\MacroTopsep
1407   \trivlist
1408   \cs_set:Npn \makelabel ##1
1409   {
1410     \llap
1411     {
1412       \hbox_unpack_drop:N \l__codedoc_macro_index_box
1413       \vtop to \baselineskip
1414       {
1415         \vbox_unpack_drop:N \l__codedoc_macro_box
1416         \vss
1417       }
1418     }
1419   }
1420   \item [ ]
1421 }

```

(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.

```

1422 \cs_new_protected:Npn \__codedoc_macro_typeset_block:nN #1#2
1423 {
1424   \__codedoc_macro_single:nNN {#1} \c_true_bool #2

```

```

1425 \seq_if_empty:NF \g__codedoc_variants_seq
1426 {
1427   \__codedoc_macro_typeset_variant_list:xN
1428   { \__codedoc_get_function_name:n {#1} } #2
1429 }
1430 }
1431 \cs_generate_variant:Nn \__codedoc_macro_typeset_block:nN { x }
1432 \cs_new_protected:Npn \__codedoc_macro_typeset_variant_list:nN #1#2
1433 {
1434   \seq_map_inline:Nn \g__codedoc_variants_seq
1435   { \__codedoc_macro_single:nNN { #1 : #1 } \c_false_bool #2 }
1436 }
1437 \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.

```

1438 \cs_new_protected:Npn \__codedoc_macro_single:nNN #1#2#3
1439 {
1440   \tl_set:Nn \saved@macroname {#1}
1441   \__codedoc_macro_typeset_one:nN {#1} #3
1442   \bool_if:NT #3 { \DoNotIndex {#1} }
1443   \exp_args:Nx \__codedoc_macro_index:nN
1444   { #1 \bool_if:NT #3 { \tl_to_str:n { TF } } }
1445   #2
1446 }
1447 \cs_new_protected:Npn \__codedoc_macro_index:nN #1#2
1448 {
1449   \DoNotIndex {#1}
1450   \bool_if:NT #2
1451   {
1452     \__codedoc_if_macro_internal:nF {#1}
1453     { \seq_gput_right:Nn \g_doc_macros_seq {#1} }
1454     \hbox_set:Nw \l__codedoc_macro_index_box
1455     \hbox_unpack_drop:N \l__codedoc_macro_index_box
1456     \int_gincr:N \c@CodelineNo
1457     \__codedoc_special_index:nn {#1} { main }
1458     \int_gdecr:N \c@CodelineNo
1459     \exp_args:NNNo \hbox_set_end:
1460     \tl_set:Nn \saved@indexname { \l__codedoc_index_key_tl }
1461   }
1462 }

```

(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.

```

1463 \cs_new_protected:Npn \__codedoc_macro_typeset_one:nN #1#2
1464 {
1465   \vbox_set:Nn \l__codedoc_macro_box
1466   {
1467     \vbox_unpack_drop:N \l__codedoc_macro_box
1468     \hbox { \llap { \__codedoc_print_macroname:nN {#1} #2 \ } }
1469   }
1470   \int_incr:N \l__codedoc_macro_int
1471 }

```

(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.

```

1472 \cs_new_protected:Npn \__codedoc_print_macroname:nN #1#2
1473 {
1474   \strut
1475   \__codedoc_get_hyper_target:xN
1476   {
1477     \exp_not:n {#1}
1478     \bool_if:NT #2 { \tl_to_str:n {TF} }
1479   }
1480   \l__codedoc_tmpa_tl
1481   \cs_if_exist:cTF { r@ \l__codedoc_tmpa_tl }
1482   { \exp_last_unbraced:NNo \hyperref [ \l__codedoc_tmpa_tl ] }
1483   { \use:n }
1484   {
1485     \int_compare:nTF { \str_count:n {#1} <= 28 }
1486     { \MacroFont } { \MacroLongFont }
1487     \tl_set:Nn \l__codedoc_tmpa_tl {#1}
1488     \tl_replace_all:Nno \l__codedoc_tmpa_tl
1489     { ~ } { \c_catcode_other_space_tl }
1490     \__codedoc_macroname_prefix:o \l__codedoc_tmpa_tl
1491     \__codedoc_macroname_suffix:N #2
1492   }
1493 }
1494 \cs_new_protected:Npn \__codedoc_macroname_prefix:n #1
1495 {
1496   \__codedoc_if_macro_internal:nTF {#1}
1497   { \__codedoc_typeset_aux:n {#1} } {#1}
1498 }
1499 \cs_generate_variant:Nn \__codedoc_macroname_prefix:n { o }
1500 \cs_new_protected:Npn \__codedoc_macroname_suffix:N #1
1501 { \bool_if:NTF #1 { \__codedoc_typeset_TF: } { } }

```

(End definition for `__codedoc_print_macroname:nN`.)

`\MacroLongFont`

```

1502 \providecommand \MacroLongFont
1503 {
1504   \fontfamily{lm}tt\fontseries{lc}\small
1505 }

```


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

```

\__codedoc_print_testfile:n Used to show that a macro has a test, somewhere.
\__codedoc_print_testfile_aux:n
1506 \cs_new_protected:Npn \__codedoc_print_testfile:n #1
1507 {
1508   \bool_set_true:N \l__codedoc_macro_tested_bool
1509   \tl_if_eq:nnF {#1} {*}
1510   {
1511     \seq_if_in:NnF \g__codedoc_testfiles_seq {#1}
1512     {
1513       \seq_gput_right:Nn \g__codedoc_testfiles_seq {#1}
1514       \par
1515       \__codedoc_print_testfile_aux:n {#1}
1516     }
1517   }
1518 }
1519 \cs_new_protected:Npn \__codedoc_print_testfile_aux:n #1
1520 {
1521   \footnotesize
1522   (
1523   \textit
1524   {
1525     The~ test~ suite~ for~ this~ command,~
1526     and~ others~ in~ this~ file,~ is~ \textsf{#1}
1527   }.
1528   )\par
1529 }

```

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

`\TestFiles`

```

1530 \DeclareDocumentCommand \TestFiles {m}
1531 {
1532   \par
1533   \textit
1534   {
1535     The~ following~ test~ files~ are~
1536     used~ for~ this~ code:~ \textsf{#1}.
1537   }
1538   \par \ignorespaces
1539 }

```

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

`\UnitTested`

```

1540 \DeclareDocumentCommand \UnitTested { } { \testfile* }

```

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

`\TestMissing`

```

1541 \DeclareDocumentCommand \TestMissing { m }
1542 { \__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.

```

1543 \cs_new_protected:Npn \__codedoc_test_missing:n #1
1544 {
1545   \__codedoc_test_missing_aux:Nxn
1546   \g__codedoc_missing_tests_prop
1547   { \seq_use:Nn \l__codedoc_names_seq { , } }
1548   { { \g_file_curr_name_str \c_space_tl (#1) } }
1549 }
1550 \cs_new_protected:Npn \__codedoc_test_missing_aux:Nnn #1#2#3
1551 {
1552   \prop_get:NnNTF #1 {#2} \l__codedoc_tmpa_tl
1553   { \tl_put_right:Nn \l__codedoc_tmpa_tl { , #3 } }
1554   { \tl_set:Nn \l__codedoc_tmpa_tl {#3} }
1555   \prop_put:Nno #1 {#2} \l__codedoc_tmpa_tl
1556 }
1557 \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.

```

1558 \cs_new_protected:Npn \__codedoc_macro_end:
1559 {
1560   \endtrivlist
1561   \__codedoc_macro_end_check_tested:
1562   \int_compare:nNnT \l__codedoc_nested_macro_int = 1
1563   { \__codedoc_macro_end_style:n { \__codedoc_print_end_definition: } }
1564 }

```

(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.

```

1565 \cs_new_protected:Npn \__codedoc_macro_end_check_tested:
1566 {
1567   \bool_lazy_all:nT
1568   {
1569     { \g__codedoc_checktest_bool }
1570     { ! \l__codedoc_macro_var_bool }
1571     { ! \l__codedoc_macro_tested_bool }
1572   }
1573   {
1574     \seq_set_filter:Nnn \l__codedoc_tmpa_seq \l__codedoc_names_seq
1575     { ! \__codedoc_if_macro_internal_p:n {##1} }
1576     \seq_gput_right:Nx \g__codedoc_not_tested_seq
1577     {
1578       \seq_use:Nn \l__codedoc_tmpa_seq { , }
1579       \bool_if:NNTF \l__codedoc_macro_pTF_bool {~(pTF)}
1580       { \bool_if:NT \l__codedoc_macro_TF_bool {~(TF)} }

```

```

1581     }
1582   }
1583 }

```

(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.

```

1584 \cs_new_protected:Npn \__codedoc_macro_end_style:n #1
1585 {
1586   \nobreak \noindent
1587   { \footnotesize ( \emph{#1} ) \par }
1588 }

```

(End definition for `__codedoc_macro_end_style:n`.)

`__codedoc_print_end_definition:` 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.

`__codedoc_macro_end_wrap_item:n`
`__codedoc_print_documented:`

```

1589 \cs_new_protected:Npn \__codedoc_macro_end_wrap_item:n #1
1590 {
1591   \tl_set:Nn \l__codedoc_tmpa_tl {#1}
1592   \tl_replace_all:Non \l__codedoc_tmpa_tl
1593   { \token_to_str:N _ } { \_ }
1594   \texttt { \l__codedoc_tmpa_tl }
1595 }
1596 \cs_new_protected:Npn \__codedoc_print_end_definition:
1597 {
1598   \seq_set_map:Nnn \l__codedoc_tmpa_seq
1599   \g__codedoc_nested_names_seq
1600   { \exp_not:n { \__codedoc_macro_end_wrap_item:n {##1} } }
1601   End~ definition~ for~
1602   \int_compare:nTF { \seq_count:N \l__codedoc_tmpa_seq <= 3 }
1603   {
1604     \seq_use:Nnnn \l__codedoc_tmpa_seq
1605     { \_,~and~ } { \_,,~ } { \_,,~and~ }
1606   }
1607   { \seq_item:Nn \l__codedoc_tmpa_seq {1}\_,~and~others }
1608   \@.
1609   \__codedoc_print_documented:
1610 }
1611 \cs_new_protected:Npn \__codedoc_print_documented:
1612 {
1613   \seq_gset_filter:Nnn \g__codedoc_nested_names_seq
1614   \g__codedoc_nested_names_seq
1615   { ! \__codedoc_if_macro_internal_p:n {##1} }
1616   \seq_if_empty:NF \g__codedoc_nested_names_seq
1617   {
1618     \int_set:Nn \l__codedoc_tmpa_int
1619     { \seq_count:N \g__codedoc_nested_names_seq }
1620     \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~This~} {~These~}
1621     \bool_if:NTF \l__codedoc_macro_var_bool {variable} {function}
1622     \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~is~} {s~are~}
1623     documented~on~page~
1624     \__codedoc_get_hyper_target:xN

```

```

1625         { \seq_item:Nn \g__codedoc_nested_names_seq { 1 } }
1626         \l__codedoc_tmpa_tl
1627         \exp_args:Nx \pageref { \l__codedoc_tmpa_tl } .
1628     }
1629     \seq_gclear:N \g__codedoc_nested_names_seq
1630 }

```

(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.

```

1631 \newcommand*{\DescribeOption}
1632 {
1633     \leavevmode
1634     \@bsphack
1635     \begingroup
1636     \MakePrivateLetters
1637     \Describe@Option
1638 }
1639 \newcommand*{\Describe@Option}[1]
1640 {
1641     \endgroup
1642     \marginpar{
1643         \raggedleft
1644         \PrintDescribeEnv{#1}
1645     }
1646     \SpecialOptionIndex{#1}
1647     \@esphack
1648     \ignorespaces
1649 }
1650 \newcommand*{\SpecialOptionIndex}[1]
1651 {
1652     \@bsphack
1653     \begingroup
1654     \HD@target
1655     \let\HDorg@encapchar\encapchar
1656     \edef\encapchar usage
1657     {
1658         \HDorg@encapchar hdclindex{\the\c@HD@hypercount}{usage}
1659     }
1660     \index
1661     {
1662         #1\actualchar{\protect\ttfamily#1}~(option)
1663         \encapchar usage
1664     }
1665     \index
1666     {
1667         options:\levelchar#1\actualchar{\protect\ttfamily#1}
1668         \encapchar usage
1669     }
1670     \endgroup

```

```

1671     \@esphack
1672   }

```

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

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

```

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

```



Provides a danger bend, as known from the T_EXbook.

The actual character from the font `manfnt`:

```

1673 \font \manual = manfnt \scan_stop:
1674 \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.

```

1675 \newenvironment {danger}
1676 {
1677   \begin{trivlist}\item[]\noindent
1678   \begin{group}\hangindent=2pc\hangafter=-2
1679   \cs_set:Npn \par{\endgraf\endgroup}
1680   \hbox to0pt{\hskip-\hangindent\dbend\hfill}\ignorespaces
1681 }
1682 {
1683   \par\end{trivlist}
1684 }

```



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.

```

1685 \newenvironment {ddanger}
1686 {
1687   \begin{trivlist}\item[]\noindent
1688   \begin{group}\hangindent=3.5pc\hangafter=-2
1689   \cs_set:Npn \par{\endgraf\endgroup}
1690   \hbox to0pt{\hskip-\hangindent\dbend\kern2pt\dbend\hfill}\ignorespaces
1691 }{
1692   \par\end{trivlist}
1693 }

```

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!}

```

```

1694 \bool_if:NTF \g__codedoc_show_notes_bool
1695 {
1696   \NewDocumentCommand\NB{mm}
1697   {
1698     (\emph{Note}\footnote{\ttfamily [#1]:~\detokenize{#2}})
1699   }
1700 }
1701 {
1702   \NewDocumentCommand\NB{mm}{ }
1703 }

```

(End definition for \NB. This function is documented on page 6.)

```

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

```

```

1704 \bool_if:NTF \g__codedoc_show_notes_bool
1705 {
1706   \NewDocumentEnvironment{NOTE}{m}
1707   {
1708     \par\noindent (\emph{Note}~[\texttt{#1}]:\par
1709     \verbatim
1710   }
1711   {
1712     \endverbatim
1713     \par\noindent \emph{Note~end}}\par
1714   }
1715 }
1716 {
1717   \NewDocumentEnvironment{NOTE}{m}{\comment}{\endcomment}
1718 }

```

5.10 Documenting templates

```

1719 \newenvironment{TemplateInterfaceDescription}[1]
1720 {
1721   \subsection{The~object~type~‘#1’}
1722   \begingroup
1723   \@beginparpenalty\@M
1724   \description
1725   \def\TemplateArgument##1##2{\item[Arg:~##1]##2\par}
1726   \def\TemplateSemantics
1727   {
1728     \enddescription\endgroup
1729     \subsubsection*{Semantics:}
1730   }
1731 }
1732 {
1733   \par\bigskip
1734 }
1735 \newenvironment{TemplateDescription}[2]
1736 {
1737   \subsection{The~template~‘#2’~(object~type~#1)}

```

```

1738 \subsubsection*{Attributes:}
1739 \beginngroup
1740 \@beginparpenalty\@M
1741 \description
1742 \def\TemplateKey##1##2##3##4
1743 {
1744   \item[##1~(##2)]##3%
1745   \ifx\TemplateKey##4\TemplateKey\else
1746 %     \hskip0ptplus3em\penalty-500\hskip Opt plus 1filll Default:~##4%
1747     \hfill\penalty500\hbox{}\hfill Default:~##4%
1748     \nobreak\hskip-\parfillskip\hskip0pt\relax
1749   \fi
1750   \par
1751 }
1752 \def\TemplateSemantics
1753 {
1754   \enddescription\endgroup
1755   \subsubsection*{Semantics~\&~Comments:}
1756 }
1757 }
1758 { \par \bigskip }

1759 \newenvironment{InstanceDescription}[4][xxxxxxxxxxxxxx]
1760 {
1761   \subsubsection{The~instance~‘#3’~(template~#2/#4)}
1762   \subsubsection*{Attribute~values:}
1763   \beginngroup
1764   \@beginparpenalty\@M
1765   \def\InstanceKey##1##2{\>\textbf{##1}\>##2\\}
1766   \def\InstanceSemantics{\endtabbing\endgroup
1767     \vskip-30pt\vskip0pt
1768     \subsubsection*{Layout~description~\&~Comments:}}
1769   \tabbing
1770   xxxx\=#1\=\kill
1771 }
1772 { \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.

<pre> \StopEventually \Finale \AlsoImplementation \OnlyDescription \g__codedoc_finale_tl </pre>	<pre> 1773 \DeclareDocumentCommand \OnlyDescription { } 1774 { \bool_gset_false:N \g__codedoc_typeset_implementation_bool } 1775 \DeclareDocumentCommand \AlsoImplementation { } 1776 { \bool_gset_true:N \g__codedoc_typeset_implementation_bool } 1777 \DeclareDocumentCommand \StopEventually { m } 1778 { 1779 \bool_if:NTF \g__codedoc_typeset_implementation_bool 1780 { 1781 \@bsphack 1782 \tl_gset:Nn \g__codedoc_finale_tl { #1 \check@checksum } 1783 \init@checksum </pre>
---	--

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

```

1784         \@esphack
1785     }
1786     { #1 \endinput }
1787 }
1788 \DeclareDocumentCommand \Finale { }
1789 { \tl_use:N \g__codedoc_finale_tl }
1790 \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.

```

1791 \cs_new_protected:Npn \__codedoc_input:n #1
1792 {
1793     \tl_gclear:N \g__codedoc_module_name_tl
1794     \MakePercentIgnore
1795     \input{#1}
1796     \MakePercentComment
1797 }

```

(End definition for `__codedoc_input:n`.)

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

```

1798 \DeclareDocumentCommand \DocInput { m }
1799 {
1800     \clist_map_inline:nn {#1}
1801     {
1802         \clist_put_right:Nn \g_docinput_clist {##1}
1803         \__codedoc_input:n {##1}
1804     }
1805 }

```

(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.

```

1806 \DeclareDocumentCommand \DocInputAgain { }
1807 { \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.

```

1808 \NewDocumentCommand \DocInclude { m }
1809 {
1810     \relax\clearpage
1811     \docincludeaux
1812     \IfFileExists{#1.fdd}
1813     { \cs_set:Npn \currentfile{#1.fdd} }
1814     { \cs_set:Npn \currentfile{#1.dtx} }
1815     \int_compare:nNnTF \@auxout = \@partaux
1816     { \@latexerr{\string\include\space cannot~be~nested}\@eha }
1817     { \docinclude #1 }
1818 }

```



```

1819 \cs_gset:Npn \@docinclude #1
1820 {
1821   \clearpage
1822   \immediate\write\@mainaux{\string\@input{#1.aux}}
1823   \@tempswtrue
1824   \if@partsw
1825     \@tempswafalse
1826     \cs_set:Npx \@tempb {#1}
1827     \clist_map_inline:Nn \@partlist
1828     {
1829       \if_meaning:w \@tempa \@tempb
1830       \@tempswtrue
1831       \fi:
1832     }
1833   \fi
1834   \if@tempswa
1835     \cs_set_eq:NN \@auxout \@partaux
1836     \immediate\openout\@partaux #1.aux
1837     \immediate\write\@partaux{\relax}
1838     \cs_set_eq:NN \@ltxdoc@PrintIndex \PrintIndex
1839     \cs_set_eq:NN \PrintIndex \relax
1840     \cs_set_eq:NN \@ltxdoc@PrintChanges \PrintChanges
1841     \cs_set_eq:NN \PrintChanges \relax
1842     \cs_set_eq:NN \@ltxdoc@theglossary \theglossary
1843     \cs_set_eq:NN \@ltxdoc@endtheglossary \endtheglossary
1844     \part{\currentfile}
1845     {
1846       \cs_set_eq:NN \ttfamily\relax
1847       \cs_gset:Npx \filekey
1848       { \filekey, \thepart = { \ttfamily \currentfile } }
1849     }
1850     \DocInput{\currentfile}
1851     \cs_set_eq:NN \PrintIndex \@ltxdoc@PrintIndex
1852     \cs_set_eq:NN \PrintChanges \@ltxdoc@PrintChanges
1853     \cs_set_eq:NN \theglossary \@ltxdoc@theglossary
1854     \cs_set_eq:NN \endtheglossary \@ltxdoc@endtheglossary
1855     \clearpage
1856     \@writeckpt{#1}
1857     \immediate \closeout \@partaux
1858   \else
1859     \@nameuse{cp@#1}
1860   \fi
1861   \cs_set_eq:NN \@auxout \@mainaux
1862 }
1863 \cs_gset:Npn \codeline@wrindex #1
1864 {
1865   \immediate\write\@indexfile
1866   {
1867     \string\indexentry{#1}
1868     { \filesep \int_use:N \c@CodelineNo }
1869   }
1870 }
1871 \tl_gclear:N \filesep

```

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

\docincludeaux

```

1872 \cs_gset:Npn \docincludeaux
1873 {
1874   \tl_set:Nn \thepart { \alphalph { part } }
1875   \tl_set:Nn \filesep { \thepart - }
1876   \cs_set_eq:NN \filekey \use_none:n
1877   \tl_gput_right:Nn \index@prologue
1878   {
1879     \cs_gset:Npn \@oddfoot
1880     {
1881       \parbox { \textwidth }
1882       {
1883         \strut \footnotesize
1884         \raggedright { \bfseries File~Key: } ~ \filekey
1885       }
1886     }
1887     \cs_set_eq:NN \@evenfoot \@oddfoot
1888   }
1889   \cs_gset_eq:NN \docincludeaux \relax
1890   \cs_gset:Npn \@oddfoot
1891   {
1892     \cs_if_exist:cTF { ver @ \currentfile }
1893     { File~\thepart :~{\ttfamily\currentfile}~ }
1894     {
1895       \GetFileInfo{\currentfile}
1896       File~\thepart :~{\ttfamily\filename}~
1897       Date:~\ExplFileDate\ % space
1898       Version~\ExplFileVersion
1899     }
1900     \hfill \thepage
1901   }
1902   \cs_set_eq:NN \@evenfoot \@oddfoot
1903 }

```

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

5.11.1 The macrocode environment

\xmacro@code 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 <@@=.

```

1904 \group_begin:
1905   \char_set_catcode_other:N \^^A
1906   \char_set_catcode_active:N \^^S
1907   \char_set_catcode_active:N \^^B

```

```

1908 \char_set_catcode_other:N \^^L
1909 \char_set_catcode_other:N \^^R
1910 \char_set_lccode:nn { '\^^A } { '%' }
1911 \char_set_lccode:nn { '\^^S } { '\ }
1912 \char_set_lccode:nn { '\^^B } { '\\ }
1913 \char_set_lccode:nn { '\^^L } { '\{ }
1914 \char_set_lccode:nn { '\^^R } { '\} }
1915 \tex_lowercase:D
1916 {
1917   \group_end:
1918   \cs_set_protected:Npn \xmacro@code
1919     #1 ^^A ^^S^^S^^S^^S ^^Bend ^^Lmacrocode^^R
1920     { \__codedoc_xmacro_code:n {#1} \end{macrocode} }
1921 }
1922 \group_begin:
1923 \char_set_catcode_active:N \<
1924 \char_set_catcode_active:N \>
1925 \cs_new_protected:Npn \__codedoc_xmacro_code:n #1
1926 {
1927   \tl_clear:N \l__codedoc_tmpa_tl
1928   \tl_if_in:nnTF {#1} { < @ @ = }
1929   { \__codedoc_xmacro_code:w #1 < @ @ = \q_recursion_tail > \q_recursion_stop }
1930   {
1931     \tl_set:Nn \l__codedoc_tmpa_tl {#1}
1932     \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
1933     \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
1934     \tl_use:N \l__codedoc_tmpa_tl
1935   }
1936 }
1937 \cs_new_protected:Npn \__codedoc_xmacro_code:w #1 < @ @ = #2 >
1938 {
1939   % Add code before <@@=...>
1940   \tl_set:Nn \l__codedoc_tmpb_tl {#1}
1941   \__codedoc_detect_internals:N \l__codedoc_tmpb_tl
1942   \__codedoc_replace_at_at:N \l__codedoc_tmpb_tl
1943   \tl_put_right:NV \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
1944   % Check for \q_recursion_tail
1945   \quark_if_recursion_tail_stop_do:nn {#2}
1946   { \tl_use:N \l__codedoc_tmpa_tl }
1947   % Change module name and add <@@=#2> to typeset output
1948   \tl_gset:Nn \g__codedoc_module_name_tl {#2}
1949   \tl_put_right:Nn \l__codedoc_tmpa_tl { < \text { \verbatim@font @ @ = #2 } > }
1950   % Loop
1951   \__codedoc_xmacro_code:w
1952 }
1953 \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.

```

1954 \iow_new:N \g__codedoc_func_iow

```

```

1955 \tl_new:N \l__codedoc_doc_def_tl
1956 \tl_new:N \l__codedoc_doc_undef_tl
1957 \tl_new:N \l__codedoc_undoc_def_tl
1958 \cs_new_protected:Npn \__codedoc_show_functions_defined:
1959 {
1960   \bool_lazy_and:nnT
1961     { \g__codedoc_typeset_implementation_bool } { \g__codedoc_checkfunc_bool }
1962     {
1963       \iow_term:x { \c__codedoc_iow_separator_tl \iow_newline: }
1964       \iow_open:Nn \g__codedoc_func_iow { \c_sys_jobname_str .cmds }
1965
1966       \tl_clear:N \l__codedoc_doc_def_tl
1967       \tl_clear:N \l__codedoc_doc_undef_tl
1968       \tl_clear:N \l__codedoc_undoc_def_tl
1969       \seq_map_inline:Nn \g_doc_functions_seq
1970       {
1971         \seq_if_in:NnTF \g_doc_macros_seq {##1}
1972         {
1973           \tl_put_right:Nx \l__codedoc_doc_def_tl
1974             { ##1 \iow_newline: }
1975           \iow_now:Nn \g__codedoc_func_iow { > ~ ##1 }
1976         }
1977         {
1978           \tl_put_right:Nx \l__codedoc_doc_undef_tl
1979             { ##1 \iow_newline: }
1980           \iow_now:Nn \g__codedoc_func_iow { ! ~ ##1 }
1981         }
1982       }
1983       \seq_map_inline:Nn \g_doc_macros_seq
1984       {
1985         \seq_if_in:NnF \g_doc_functions_seq {##1}
1986         {
1987           \tl_put_right:Nx \l__codedoc_undoc_def_tl
1988             { ##1 \iow_newline: }
1989           \iow_now:Nn \g__codedoc_func_iow { ? ~ ##1 }
1990         }
1991       }
1992       \__codedoc_functions_typeout:nN
1993       {
1994         Functions~both~documented~and~defined: \iow_newline:
1995         (In~order~of~being~documented)
1996       }
1997       \l__codedoc_doc_def_tl
1998       \__codedoc_functions_typeout:nN
1999       { Functions~documented~but~not~defined: }
2000       \l__codedoc_doc_undef_tl
2001       \__codedoc_functions_typeout:nN
2002       { Functions~defined~but~not~documented: }
2003       \l__codedoc_undoc_def_tl
2004
2005       \iow_close:N \g__codedoc_func_iow
2006       \iow_term:x { \c__codedoc_iow_separator_tl }
2007     }
2008 }

```

```

2009 \AtEndDocument { \__codedoc_show_functions_defined: }
      TODO: use \iow_term:x.
2010 \cs_new_protected:Npn \__codedoc_functions_typeout:nN #1#2
2011 {
2012   \tl_if_empty:NF #2
2013   {
2014     \typeout
2015     {
2016       \c__codedoc_iow_midrule_tl \iow_newline:
2017       #1 \iow_newline:
2018       \c__codedoc_iow_midrule_tl \iow_newline:
2019       #2
2020     }
2021     \tl_clear:N #2
2022   }
2023 }
2024 \cs_new_protected:Npn \__codedoc_show_not_tested:
2025 {
2026   \bool_if:NT \g__codedoc_checktest_bool
2027   {
2028     \tl_clear:N \l__codedoc_tmpa_tl
2029     \prop_if_empty:NF \g__codedoc_missing_tests_prop
2030     {
2031       \cs_set:Npn \__codedoc_tmpa:w ##1##2
2032       {
2033         \iow_newline:
2034         \space\space\space\space \exp_not:n {##1}
2035         \clist_map_function:nN {##2} \__codedoc_tmpb:w
2036       }
2037       \cs_set:Npn \__codedoc_tmpb:w ##1
2038       {
2039         \iow_newline:
2040         \space\space\space\space\space\space * ~ ##1
2041       }
2042       \tl_put_right:Nx \l__codedoc_tmpa_tl
2043       {
2044         \iow_newline: \iow_newline:
2045         The~ following~ macro(s)~ have~ incomplete~ tests:
2046         \iow_newline:
2047         \prop_map_function:NN
2048         \g__codedoc_missing_tests_prop \__codedoc_tmpa:w
2049       }
2050     }
2051     \seq_if_empty:NF \g__codedoc_not_tested_seq
2052     {
2053       \cs_set:Npn \__codedoc_tmpa:w ##1
2054       { \clist_map_function:nN {##1} \__codedoc_tmpb:w }
2055       \cs_set:Npn \__codedoc_tmpb:w ##1
2056       {
2057         \iow_newline:
2058         \space\space\space\space ##1
2059       }
2060       \tl_put_right:Nx \l__codedoc_tmpa_tl

```

```

2061         {
2062             \iow_newline:
2063             \iow_newline:
2064             The~ following~ macro(s)~ do~ not~ have~ any~ tests:
2065             \iow_newline:
2066             \seq_map_function:NN
2067             \g__codedoc_not_tested_seq \__codedoc_tmpa:w
2068         }
2069     }
2070     \tl_if_empty:NF \l__codedoc_tmpa_tl
2071     {
2072         \int_set:Nn \l__codedoc_tmpa_int { \tex_interactionmode:D }
2073         \errorstopmode
2074         \ClassError { l3doc } { \l__codedoc_tmpa_tl } { }
2075         \int_set:Nn \tex_interactionmode:D { \l__codedoc_tmpa_int }
2076     }
2077 }
2078 }
2079 \AtEndDocument { \__codedoc_show_not_tested: }

```

5.13 Indexing

5.13.1 Userspace commands

Fix index (for now):

```

2080 \g@addto@macro \theindex { \MakePrivateLetters }
2081 \cs_gset:Npn \verbatimchar {&}
2082 \setcounter { IndexColumns } { 2 }

```

Set up the Index to use \part

```

2083 \IndexPrologue
2084 {
2085     \part*{Index}
2086     \markboth{Index}{Index}
2087     \addcontentsline{toc}{part}{Index}
2088     The~italic-numbers-denote-the-pages-where-the-
2089     corresponding-entry-is-described,~
2090     numbers-underlined-point-to-the-definition,~
2091     all-others-indicate-the-places-where-it-is-used.
2092 }

```

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

```

2093 \cs_gset_protected:Npn \SpecialIndex #1
2094 {
2095     \@bsphack
2096     \__codedoc_special_index:nn {#1} { }
2097     \@esphack
2098 }

```

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

```

2099 \msg_new:nnn { l3doc } { print-index-howto }
2100 {

```

```

2101     Generate~the~index~by~executing\\
2102     \iow_indent:n
2103     { makeindex~~s~gind.ist~-o~\c_sys_jobname_str.ind~\c_sys_jobname_str.idx }
2104   }
2105 \tl_gput_right:Nn \PrintIndex
2106 { \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.

```

2107 \cs_gset_protected:Npn \it@is@a #1
2108 {
2109   \use:x
2110   {
2111     \__codedoc_special_index_module:nnnnN
2112     {#1}
2113     { \bslash #1 }
2114     { }
2115     { }
2116     \c_false_bool
2117   }
2118 }

```

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

`__codedoc_special_index:nn`

```

2119 \cs_new_protected:Npn \__codedoc_special_index:nn #1#2
2120 {
2121   \__codedoc_key_get:n {#1}
2122   \quark_if_no_value:NF \l__codedoc_override_module_tl
2123   { \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_override_module_tl }
2124   \__codedoc_special_index_module:ooonN
2125   { \l__codedoc_index_key_tl }
2126   { \l__codedoc_index_macro_tl }
2127   { \l__codedoc_index_module_tl }
2128   {#2}
2129   \l__codedoc_index_internal_bool
2130 }
2131 \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{...}`.

```

2132 \tl_new:N \l__codedoc_index_escaped_macro_tl
2133 \tl_new:N \l__codedoc_index_escaped_key_tl
2134 \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

2135 {
2136   \use:x
2137   {
2138     \exp_not:n { \__codedoc_special_index_aux:nnnnnn {#1} {#2} }
2139     \tl_if_empty:nTF {#3}
2140       { { } { } { } }
2141       {
2142         \str_if_eq:nnTF {#3} { TeX }
2143         {
2144           { TeX~and~LaTeX2e }
2145           { \string\TeX{ }~and~\string\LaTeXe{ } }
2146         }
2147         {
2148           {#3}
2149           { \string\pkg{#3} }
2150         }
2151         { \bool_if:NT #5 { ~internal } ~commands: }
2152       }
2153     }
2154     {#4}
2155   }

2156 \cs_generate_variant:Nn \__codedoc_special_index_module:nnnnN { ooo }
2157 \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.)

2158 {
2159   \tl_set:Nn \l__codedoc_index_escaped_key_tl {#1}
2160   \__codedoc_quote_special_char:N \l__codedoc_index_escaped_key_tl
2161   \__codedoc_special_index_set:Nn \l__codedoc_index_escaped_macro_tl {#2}
2162   \str_if_eq:onTF { \@currenvir } { macrocode }
2163     { \codeline@wrindex }
2164     {
2165       \str_case:nnF {#6}
2166       {
2167         { main } { \codeline@wrindex }
2168         { usage } { \index }
2169       }
2170       { \HD@target \index }
2171     }
2172   {
2173     \tl_if_empty:nF { #3 #4 #5 }
2174     { #3 #5 \actualchar #4 #5 \levelchar }
2175     \l__codedoc_index_escaped_key_tl
2176     \actualchar
2177     {

```



```

2178         \token_to_str:N \verbatim@font \c_space_tl
2179         \l__codedoc_index_escaped_macro_tl
2180     }
2181     \encapchar
2182     hdclindex{\the\c@HD@hypercount}{#6}
2183 }
2184 }
2185 \cs_new_protected:Npn \__codedoc_special_index_set:Nn #1#2
2186 {
2187     \tl_set:Nx #1 { \tl_to_str:n {#2} }
2188     \__codedoc_if_almost_str:nTF {#2}
2189     {
2190         \tl_replace_all:Non #1 { \tl_to_str:n { __ } }
2191         {
2192             \verbatimchar
2193             \token_to_str:N \_ \token_to_str:N \_
2194             \token_to_str:N \verb * \verbatimchar
2195         }
2196         \exp_args:Nx \tl_map_inline:nn
2197         { \tl_to_str:N \verbatimchar \token_to_str:N _ }
2198         {
2199             \tl_replace_all:Nnn #1 {##1}
2200             {
2201                 \verbatimchar \c_backslash_str ##1
2202                 \token_to_str:N \verb * \verbatimchar
2203             }
2204         }
2205         \tl_set:Nx #1
2206         {
2207             \token_to_str:N \verb * \verbatimchar
2208             #1 \verbatimchar
2209         }
2210     }
2211     {
2212         \tl_set:Nn #1 {#2}
2213         \tl_replace_all:Non #1
2214         { \c_backslash_str }
2215         { \token_to_str:N \bslash \c_space_tl }
2216     }
2217     \__codedoc_quote_special_char:N #1
2218 }

```

(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.

```

2219 \cs_new_protected:Npn \__codedoc_quote_special_char:N #1
2220 {
2221     \tl_map_inline:nn { \quotechar \actualchar \encapchar \levelchar \bslash }
2222     {
2223         \tl_replace_all:Nxn #1
2224         { \tl_to_str:N ##1 } { \quotechar \tl_to_str:N ##1 }
2225     }
2226 }

```

(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 $\langle key \rangle$ 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 $\text{\TeX/L\TeX} 2_{\epsilon}$ 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 $\text{\TeX/L\TeX} 2_{\epsilon}$ commands the module name is `TeX`, and others have an empty module name.

```

2227 \cs_new_protected:Npn \__codedoc_key_get:n #1
2228 {
2229   \__codedoc_key_get_base:nN {#1} \l__codedoc_index_macro_tl
2230   \tl_set:Nx \l__codedoc_index_key_tl
2231     { \tl_to_str:N \l__codedoc_index_macro_tl }
2232   \tl_clear:N \l__codedoc_index_module_tl
2233   \tl_if_in:NoTF \l__codedoc_index_key_tl { \tl_to_str:n { __ } }
2234     { \bool_set_true:N \l__codedoc_index_internal_bool }
2235     { \bool_set_false:N \l__codedoc_index_internal_bool }
2236   \exp_last_unbraced:NNo
2237   \tl_if_head_eq_charcode:oNT
2238     { \l__codedoc_index_key_tl } \c_backslash_str
2239     { \__codedoc_key_pop: }
2240   \tl_if_in:NoTF \l__codedoc_index_key_tl { \token_to_str:N : }
2241     { \__codedoc_key_func: }
2242     {
2243       \tl_if_in:NoTF \l__codedoc_index_key_tl { \token_to_str:N _ }
2244         { \__codedoc_key_var: }
2245         {
2246           \tl_if_in:NoT \l__codedoc_index_key_tl { \token_to_str:N @ }
2247             { \tl_set:Nn \l__codedoc_index_module_tl { TeX } }
2248         }
2249     }
2250 }
2251 \cs_new_protected:Npn \__codedoc_key_pop:
2252 {
2253   \tl_set:Nx \l__codedoc_index_key_tl
2254     { \tl_tail:N \l__codedoc_index_key_tl }
2255 }

```

(End definition for `__codedoc_key_get:n`.)

`__codedoc_key_trim_module:n` Helper that removes from `\l__codedoc_index_module_tl` everything after the first occurrence of `#1`. `__codedoc_key_drop_underscores:` Helper that removes any leading underscore from `\l__codedoc_index_key_tl`.

```

2256 \cs_new_protected:Npn \__codedoc_key_trim_module:n #1
2257 {
2258   \cs_set:Npn \__codedoc_tmpa:w ##1 #1 ##2 \q_stop
2259     { \exp_not:n {##1} }

```

```

2260     \tl_set:Nx \l__codedoc_index_module_tl
2261     { \exp_after:wN \__codedoc_tmpa:w \l__codedoc_index_module_tl #1 \q_stop }
2262   }
2263   \cs_new_protected:Npn \__codedoc_key_drop_underscores:
2264   {
2265     \tl_if_head_eq_charcode:oNT { \l__codedoc_index_key_tl } _
2266     { \__codedoc_key_pop: \__codedoc_key_drop_underscores: }
2267   }

```

(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.

```

2268   \cs_new_protected:Npn \__codedoc_key_func:
2269   {
2270     \tl_if_head_eq_charcode:oNT { \l__codedoc_index_key_tl } .
2271     { \__codedoc_key_pop: }
2272     \__codedoc_key_drop_underscores:
2273     \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
2274     \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N : }
2275     \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
2276   }

```

(End definition for __codedoc_key_func:.)

__codedoc_key_var: 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.

```

2277   \cs_new_protected:Npn \__codedoc_key_var:
2278   {
2279     \exp_args:Nx \tl_if_head_eq_charcode:nNTF
2280     { \exp_args:No \str_tail:n \l__codedoc_index_key_tl } _
2281     {
2282       \str_case:fn { \str_head:N \l__codedoc_index_key_tl }
2283       {
2284         { q } { \tl_set:Nn \l__codedoc_index_module_tl { quark } }
2285         { s } { \tl_set:Nn \l__codedoc_index_module_tl { quark } }
2286       }
2287       \__codedoc_key_pop:
2288       \__codedoc_key_pop:
2289       \__codedoc_key_drop_underscores:
2290       \tl_if_empty:NT \l__codedoc_index_module_tl
2291       {
2292         \seq_set_split:NoV \l__codedoc_tmpa_seq
2293         { \token_to_str:N _ } \l__codedoc_index_key_tl
2294         \tl_set:Nx \l__codedoc_index_module_tl

```

```

2295         {
2296         \int_case:nnF { \seq_count:N \l__codedoc_tmpa_seq }
2297         {
2298             { 0 } { }
2299             { 1 } { int }
2300             { 2 } { \seq_item:Nn \l__codedoc_tmpa_seq { 2 } }
2301         }
2302         { \seq_item:Nn \l__codedoc_tmpa_seq { 1 } }
2303     }
2304 }
2305 }
2306 {
2307     \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
2308     \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
2309 }
2310 }

```

(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...

```

2311 \GlossaryPrologue
2312 {
2313     \part*{Change~History}
2314     {\GlossaryParms\ttfamily\hyphenchar\font='{-}
2315     \markboth{Change~History}{Change~History}
2316     \addcontentsline{toc}{part}{Change~History}
2317 }
2318 \msg_new:nnn { l3doc } { print-changes-howto }
2319 {
2320     Generate~the~change~list~by~executing\
2321     \iow_indent:n
2322     { makeindex--s-gglo.ist--o~\c_sys_jobname_str.gls~\c_sys_jobname_str.glo }
2323 }
2324 \tl_gput_right:Nn \PrintChanges
2325 { \AtEndDocument { \msg_info:nn { l3doc } { print-changes-howto } } }

```

5.15 Default configuration

```

2326 \bool_if:NTF \g__codedoc_typeset_implementation_bool
2327 {
2328     \RecordChanges
2329     \CodelineIndex
2330     \EnableCrossrefs
2331     \AlsoImplementation
2332 }
2333 {
2334     \CodelineNumbered
2335     \DisableCrossrefs
2336     \OnlyDescription
2337 }
2338 </class>

```

5.16 Internal macros for L^AT_EX3 sources

These definitions are only used by the L^AT_EX3 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.*

```
2339 <*cfg>
      The Guilty Parties.
2340 \tl_const:Nn \Team
2341 {
2342   The~\LaTeX3-Project\thanks
2343   {\url{https://www.latex-project.org/latex3/}}
2344 }
2345 \NewDocumentCommand{\ExplMakeTitle}{mm}
2346 {
2347   \title
2348   {
2349     The~\pkg{#1}~package \\\ #2
2350   }
2351   \author
2352   {
2353     The~\LaTeX3-Project\thanks{E-mail:~
2354     \href{mailto:latex-l@listserv.uni-heidelberg.de}
2355     {latex-l@listserv.uni-heidelberg.de}}
2356   }
2357   \date{Released~\ExplFileDate}
2358   \maketitle
2359 }
```

5.17 Math extras

For l3fp.

```
2360 \AtBeginDocument
2361 {
2362   \clist_map_inline:nn
2363   {
2364     asin, acos, atan, acot,
2365     asinh, acosh, atanh, acoth, round, floor, ceil
2366   }
2367   { \exp_args:Nc \DeclareMathOperator{#1}{#1} }
2368 }

\nan

2369 \NewDocumentCommand { \nan } { } { \text { \texttt { nan } } }

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

2370 </cfg>
```

5.18 Makeindex configuration

2371 `<*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.

```

2372 actual '='
2373 quote '!'
2374 level '>'
2375 preamble
2376 "\n \\\begin{theindex} \n \\\makeatletter\\scan@allowedfalse\n"
2377 postamble
2378 "\n\n \\\end{theindex}\n"
2379 item_x1 "\\efill \n \\\subitem "
2380 item_x2 "\\efill \n \\\subsubitem "
2381 delim_0 "\\pfill "
2382 delim_1 "\\pfill "
2383 delim_2 "\\pfill "
2384 % The next lines will produce some warnings when
2385 % running Makeindex as they try to cover two different
2386 % versions of the program:
2387 lethead_prefix "{\\bfseries\\hfil "
2388 lethead_suffix "\\hfil}\\nopagebreak\n"
2389 lethead_flag 1
2390 heading_prefix "{\\bfseries\\hfil "
2391 heading_suffix "\\hfil}\\nopagebreak\n"
2392 headings_flag 1
2393
2394 % and just for source3:
2395 % Remove R so I is treated in sequence I J K not I II III
2396 page_precedence "rnaA"

```

(End definition for .)

2397 `</docist>`

Index

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

Symbols		
<code>\"</code> 517, 522	<code>\-</code> 2314
<code>\#</code> 628	<code>\/</code> 752
<code>\%</code> 1910	<code>\:</code> 464
<code>\&</code> 1755, 1768	<code>\<</code> 1068, 1069, 1923
<code>\,</code> 1224, 1230, 1605, 1607	<code>\=</code> 1770
		<code>\></code> 1765, 1924

<code>\</code>	484, 1159, 1176, 1224, 1230, 1395, 1765, 1912, 2101, 2320, 2349	<code>\bool_new:N</code>	6, 15, 16, 19, 23, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 47, 54, 66, 67, 68, 69, 70, 76
<code>\{</code>	562, 1913	<code>\bool_set:Nn</code>	1084
<code>\}</code>	562, 1914	<code>\bool_set_false:N</code>	581, 673, 953, 958, 967, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1300, 1312, 1317, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 2235
<code>^</code>	154, 808, 810, 811, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914	<code>\bool_set_true:N</code>	7, 71, 72, 588, 674, 947, 952, 959, 964, 965, 966, 972, 973, 982, 1019, 1020, 1065, 1279, 1280, 1285, 1286, 1290, 1293, 1297, 1298, 1299, 1305, 1306, 1311, 1318, 1322, 1508, 2234
<code>_</code>	463, 1593, 2193	<code>\c_false_bool</code>	235, 891, 894, 1435, 2116
<code> </code>	518, 523	<code>\c_true_bool</code>	234, 1424
<code>_</code>	636, 641, 650, 659, 1468, 1897, 1911	<code>\bottomrule</code>	1145
A			
<code>\actualchar ..</code>	1662, 1667, 2174, 2176, 2221	box commands:	
<code>\addcontentsline</code>	2087, 2316	<code>\box_clear:N</code>	1363
<code>\addpenalty</code>	500	<code>\box_dp:N</code>	1047
<code>\addtolength</code>	486, 487, 488	<code>\box_gclear:N</code>	1055
<code>\advvspace</code>	501	<code>\box_if_empty:NTF</code>	1052, 1236
<code>\advance</code>	507	<code>\box_new:N</code>	14, 60, 61
<code>\allowbreak</code>	1048	<code>\box_use_drop:N</code>	1097, 1264
<code>\alphalph</code>	1874	<code>\box_wd:N</code>	1083
<code>\AlsoImplementation</code>	4, 1773, 2331	<code>\bslash</code>	2113, 2215, 2221
<code>\arabic</code>	470, 628	C	
<code>\Arg</code>	5, 25, 561	<code>\catcode</code>	1395
arguments (environment)	7, 624	catcode commands:	
<code>\arrayrulecolor</code>	1246, 1258	<code>\c_catcode_active_space_tl</code>	164
<code>\AtBeginDocument</code>	515, 2360	<code>\c_catcode_other_space_tl</code>	871, 1489
<code>\AtEndDocument</code>	520, 2009, 2079, 2106, 2325	<code>\changes</code>	47
<code>\author</code>	2351	<code>\char</code>	562, 1674
B			
<code>\baselineskip ..</code>	639, 648, 657, 1372, 1413	char commands:	
<code>\begin ..</code>	1140, 1247, 1249, 1262, 1677, 1687	<code>\char_set_active_eq:NN</code>	1068
<code>\begingroup</code>	503, 1635, 1653, 1678, 1688, 1722, 1739, 1763	<code>\char_set_catcode:nn</code>	734
<code>\bfseries</code>	506, 1884	<code>\char_set_catcode_active:N</code>	154, 1069, 1906, 1907, 1923, 1924
<code>\bigskip</code>	1042, 1733, 1758, 1772	<code>\char_set_catcode_letter:N</code>	462, 463, 464
bool commands:		<code>\char_set_catcode_other:N</code>	1905, 1908, 1909
<code>\bool_gset_false:N</code>	390, 395, 399, 403, 405, 409, 415, 575, 577, 1774	<code>\char_set_lccode:nn</code>	1910, 1911, 1912, 1913, 1914
<code>\bool_gset_true:N</code>	40, 384, 385, 389, 393, 397, 401, 407, 413, 571, 573, 1776	<code>\ClassError</code>	2074
<code>\bool_if:NTF</code>	150, 188, 280, 297, 450, 580, 584, 587, 591, 594, 599, 679, 692, 704, 801, 833, 844, 887, 893, 907, 909, 938, 1098, 1152, 1155, 1171, 1172, 1186, 1191, 1203, 1208, 1241, 1260, 1442, 1444, 1450, 1478, 1501, 1579, 1580, 1621, 1694, 1704, 1779, 2026, 2151, 2326	<code>\clearpage</code>	1810, 1821, 1855
<code>\bool_lazy_all:nnTF</code>	1567	clist commands:	
<code>\bool_lazy_and:nnTF</code>	1217, 1960	<code>\clist_clear:N</code>	1066
<code>\bool_lazy_or:nnTF</code>	283	<code>\clist_count:N</code>	11
		<code>\clist_count:n</code>	11
		<code>\clist_map_function:NN</code>	1807
		<code>\clist_map_function:nN</code>	2035, 2054

\clist_map_inline:Nn	1193, 1827	__codedoc_detect_internals_-		
\clist_map_inline:nn	1800, 2362	aux:N	148
\clist_new:N	3, 75	\l__codedoc_detect_internals_-		
\clist_put_right:Nn	1802	bool	6, 150
\clist_set:Nn	981	\l__codedoc_detect_internals_cs_-		
\g_docinput_clist	..	3, 56, 1802, 1807	tl	9, 168, 173
\closeout	1857	\l__codedoc_detect_internals_tl	.	
\cls	5, 566	6, 157, 158, 159, 160, 161,	
\cmd	5, 12, 28, 536, 556		163, 165, 166, 168, 169, 170, 171, 174	
codedoc internal commands:					
__codedoc_base_form_aux:nnN	...		\l__codedoc_doc_def_tl	
.....		253, 277, 305, 839	1955, 1966, 1973, 1997	
__codedoc_base_form_aux:nnnnN	.		\l__codedoc_doc_undef_tl	
.....		293, 295	1956, 1967, 1978, 2000	
__codedoc_base_form_signature_-			__codedoc_ensuremath_sb:n	..	30, 727
do:nnn	290	\g__codedoc_finale_tl	1773
\g__codedoc_base_name_tl		\g__codedoc_func_iow	
.....		73, 869, 870, 876, 882	..	1954, 1964, 1975, 1980, 1989, 2005	
\g__codedoc_checkfunc_bool		__codedoc_function:nnw	
.....		34, 393, 395, 1961	39, 596, 604, 1022	
\g__codedoc_checktest_bool		__codedoc_function_assemble:	...	
.....		34, 397, 399, 1569, 2026	1037, 1094	
__codedoc_cmd:nn	25, 537, 539, 542, 671		__codedoc_function_descr_-		
\l__codedoc_cmd_index_tl		start:w	1032, 1087
.....		63, 666, 675, 706, 709	__codedoc_function_descr_stop:	.	
\l__codedoc_cmd_module_tl	1036, 1087	
.....		63, 667, 676, 713, 716	__codedoc_function_end:	
\l__codedoc_cmd_noindex_bool	39, 601, 605, 1022	
.....		28, 63, 668, 673, 704	__codedoc_function_extra_-		
\l__codedoc_cmd_replace_bool	...		labels:	1142, 1189
.....		63, 669, 674, 679	__codedoc_function_index:n	..	1149
\l__codedoc_cmd_tl	63, 678, 682, 683,		__codedoc_function_init:	1025, 1050	
684, 689, 691, 697, 700, 701, 708, 712			__codedoc_function_label:nN	...	
\g__codedoc_cs_break_bool	34, 409, 692		1153, 1201, 1214	
\l__codedoc_date_added_tl		\l__codedoc_function_label_clist	75, 981, 1066, 1193
.....		77, 975, 1218, 1221, 1224	__codedoc_function_reset:	1031, 1074	
__codedoc_date_compare:nNnTF	...		__codedoc_function_typeset:	...	
.....		312, 1001, 1012	1030, 1078	
__codedoc_date_compare_aux:nnnNnnn	312	__codedoc_function_typeset_-		
.....		312	start:	1024, 1040
__codedoc_date_compare_aux:w	..	312	__codedoc_function_typeset_-		
__codedoc_date_compare_p:nNn	..	312	stop:	1038, 1040
__codedoc_date_set:Nn	988, 1011	\l__codedoc_functions_coffin	...	
__codedoc_date_set_past:Nn	10, 11,	
.....		36, 975, 976, 988	37, 38, 40, 1057, 1081, 1083, 1106, 1127		
\l__codedoc_date_updated_tl		__codedoc_functions_typeout:nN	.	
.....		77, 976, 1219, 1227, 1230	1992, 1998, 2001, 2010	
__codedoc_deprecated_on:n		__codedoc_get_function_name:n	..	
.....		977, 1009, 1326	221, 225, 1177, 1428	
\l__codedoc_descr_coffin		__codedoc_get_function_signature:n	223, 225
11, 1047, 1054, 1089, 1110, 1113, 1123			__codedoc_get_hyper_target:nN	..	
__codedoc_detect_internals:N	790, 1195, 1205, 1475, 1624	
.....		148, 812, 1932, 1941	__codedoc_gprop_name:n	347


```

\__codedoc_if_almost_str:nTF ...
..... 105, 113, 249, 689, 2188
\__codedoc_if_detect_internals_
ok:N ..... 180
\__codedoc_if_detect_internals_
ok:NTF ..... 148
\__codedoc_if_macro_internal:nTF
..... 905, 1452, 1496
\__codedoc_if_macro_internal_
aux:w ..... 905
\__codedoc_if_macro_internal_p:n
..... 905, 1575, 1615
\l_codedoc_in_function_bool ...
..... 15, 1065, 1260
\l_codedoc_in_implementation_
bool ..... 26, 68, 581, 588, 594, 599
\l_codedoc_index_escaped_key_tl
..... 2133, 2159, 2160, 2175
\l_codedoc_index_escaped_macro_
tl ..... 2132, 2161, 2179
\l_codedoc_index_internal_bool .
..... 11, 50, 723, 2129, 2234, 2235
\l_codedoc_index_key_tl .....
..... 11, 50, 66, 66,
719, 1460, 2125, 2230, 2233, 2238,
2240, 2243, 2246, 2253, 2254, 2265,
2270, 2273, 2280, 2282, 2293, 2307
\l_codedoc_index_macro_tl .....
..... 11, 50, 66, 720, 2126, 2229, 2231
\l_codedoc_index_module_tl .....
..... 11, 50, 66, 66, 715, 721,
2123, 2127, 2232, 2247, 2260, 2261,
2273, 2284, 2285, 2290, 2294, 2307
\__codedoc_input:n .. 1791, 1803, 1807
\c_codedoc_iow_mid_rule_tl .... 58
\c_codedoc_iow_midrule_tl .....
..... 56, 2016, 2018
\c_codedoc_iow_rule_tl ..... 56
\c_codedoc_iow_separator_tl ...
..... 1963, 2006
\g_codedoc_kernel_bool .....
..... 34, 188, 401, 403
\__codedoc_key_drop_underscores:
..... 2256, 2272, 2289
\__codedoc_key_func: 66, 67, 2241, 2268
\__codedoc_key_get:n . 712, 2121, 2227
\__codedoc_key_get_base:nN .....
..... 66, 247, 2229
\__codedoc_key_get_base_TF:nN ...
..... 251, 257
\__codedoc_key_get_module: ... 2277
\__codedoc_key_pop: .....
.. 2239, 2251, 2266, 2271, 2287, 2288

\__codedoc_key_trim_module:n ...
..... 14, 2256, 2274, 2275, 2308
\__codedoc_key_var: 66, 67, 2244, 2277
\g_codedoc_lmodern_bool .....
..... 34, 405, 407, 450
\l_codedoc_long_name_bool .....
..... 10, 16, 38, 1084, 1098, 1241
\__codedoc_lseq_name:n 11, 32, 347, 850
\__codedoc_macro:nnw . 595, 607, 1334
\l_codedoc_macro_argument_tl ...
..... 79, 1005, 1016, 1026, 1337
\l_codedoc_macro_box .....
..... 48, 60, 1364, 1415, 1465, 1467
\l_codedoc_macro_do_not_index_
tl ..... 11, 50, 1331,
1392, 1394, 1396, 1398, 1399, 1401
\l_codedoc_macro_documented_tl .
..... 23, 1329, 1361, 1381, 1385
\__codedoc_macro_dump: ... 1344, 1404
\__codedoc_macro_end: 600, 608, 1558
\__codedoc_macro_end_check_
tested: ..... 1561, 1565
\__codedoc_macro_end_style:n ...
..... 1563, 1584
\__codedoc_macro_end_wrap_item:n
..... 1589
\__codedoc_macro_exclude_index: .
..... 1341, 1390
\l_codedoc_macro_EXP_bool .....
..... 23, 952, 958,
966, 1061, 1171, 1299, 1311, 1317, 1355
\__codedoc_macro_index:nN 1443, 1447
\l_codedoc_macro_index_box ....
..... 60, 1363, 1412, 1454, 1455
\__codedoc_macro_init: ... 1336, 1347
\l_codedoc_macro_int .....
..... 60, 1369, 1372, 1470
\l_codedoc_macro_internal_bool .
.. 23, 34, 909, 1019, 1279, 1285, 1350
\l_codedoc_macro_internal_set_
bool . 23, 907, 1020, 1280, 1286, 1351
\l_codedoc_macro_noTF_bool .....
..... 23, 893, 972, 1060, 1306, 1354
\l_codedoc_macro_pTF_bool .....
. 23, 887, 964, 1059, 1298, 1353, 1579
\__codedoc_macro_reset: .. 1345, 1375
\l_codedoc_macro_rEXP_bool .....
..... 23, 953, 959,
967, 1062, 1172, 1300, 1312, 1318, 1356
\__codedoc_macro_save_names: ...
..... 1342, 1379
\__codedoc_macro_save_names_
aux:n ..... 1382, 1388

```

__codedoc_macro_single:nN	\g__codedoc_nested_names_seq . . .
. 30 , 44 , 1424 , 1435 , 1438 49 , 1384 , 1389 , 1599 ,
\l__codedoc_macro_tested_bool . . .	1613 , 1614 , 1616 , 1619 , 1625 , 1629
. 19 , 1322 , 1358 , 1508 , 1571	\l__codedoc_no_label_bool
\l__codedoc_macro_TF_bool 75 , 982 , 1063 , 1191 , 1203
. 23 , 844 , 895 , 938 , 947 , 965 ,	\g__codedoc_not_tested_seq
973 , 1058 , 1293 , 1297 , 1305 , 1352 , 1580 19 , 1576 , 2051 , 2067
__codedoc_macro_typeset_-	__codedoc_oldlist:nn 489
block:nN 902 , 1422	\l__codedoc_output_coffin
__codedoc_macro_typeset_one:nN 10 , 1101 , 1105 ,
. 1441 , 1463	1109 , 1112 , 1118 , 1122 , 1126 , 1129
__codedoc_macro_typeset_-	\l__codedoc_override_module_tl . .
variant_list:nN . 1427 , 1432 , 1437 10 , 23 , 986 , 1067 ,
\l__codedoc_macro_var_bool	1076 , 1328 , 1360 , 1377 , 2122 , 2123
. 23 , 1290 , 1357 , 1570 , 1621	__codedoc_pdfstring_cmd:w 548
__codedoc_macroname_prefix:n . . .	__codedoc_pdfstring_cs:w 548
. 1490 , 1494 , 1499	__codedoc_pdfstring_meta:w 548
__codedoc_macroname_suffix:N . . .	__codedoc_pdfstring_newline:w . 480
. 1491 , 1500	__codedoc_predicate_from_base:n
__codedoc_meta:n 547 , 727 219 , 890
__codedoc_meta_original:n 727	__codedoc_print_documented: . 1589
\g__codedoc_missing_tests_prop . .	__codedoc_print_end_definition:
. 19 , 50 , 1546 , 2029 , 2048 1563 , 1589
\g__codedoc_module_name_tl . . 55 ,	__codedoc_print_macroname:nN . . .
132 , 135 , 175 , 182 , 185 , 1793 , 1948 1468 , 1472
__codedoc_names_block_base_-	__codedoc_print_testfile:n
map:N 923 , 1382 1362 , 1506
\l__codedoc_names_block_tl 32 , 33 ,	__codedoc_print_testfile_aux:n 1506
35 , 38 , 45 , 45 , 820 , 853 , 855 , 863 , 925	__codedoc_quote_special_char:N .
__codedoc_names_get_seq:nN 2160 , 2217 , 2219
. 798 , 1028 , 1339	__codedoc_replace_at_at:N
__codedoc_names_parse: 28 , 130 , 683 , 813 , 1933 , 1942
. 818 , 1029 , 1340	__codedoc_replace_at_at_aux:Nn 130
__codedoc_names_parse_aux:Nnn . .	__codedoc_shorthand_meta: 1068 , 1071
. 850 , 851	__codedoc_shorthand_meta:w . . 1071
__codedoc_names_parse_aux:nmn . .	__codedoc_show_functions_-
. 836 , 838 , 846 , 849	defined: 1958 , 2009
__codedoc_names_parse_one:n . . 818	__codedoc_show_not_tested:
__codedoc_names_parse_one_- 2024 , 2079
aux:nnNn 828 , 831	\g__codedoc_show_notes_bool
\l__codedoc_names_seq 34 , 413 , 415 , 1694 , 1704
. 48 , 822 , 1028 , 1339 , 1547 , 1574	__codedoc_signature_base_form:n
__codedoc_names_typeset: 198 , 287
. 861 , 1143 , 1343	__codedoc_signature_base_form_-
__codedoc_names_typeset_auxi:n .	aux:n 198
. 33 , 861	__codedoc_signature_base_form_-
__codedoc_names_typeset_auxii:n	aux:w 198
. 33 , 876 , 881 , 885	__codedoc_special_index:nn
__codedoc_names_typeset_- 1165 , 1457 , 2096 , 2119
block:nN 889 , 894 , 895 , 898	__codedoc_special_index_-
\l__codedoc_names_verb_bool	aux:nnnnnn 2132
. 47 , 801 , 985 , 1064 , 1327 , 1359	__codedoc_special_index_-
\l__codedoc_nested_macro_int . . .	module:nnnnN . 718 , 2111 , 2124 , 2132
. 18 , 34 , 44 , 900 , 1349 , 1562	__codedoc_special_index_set:Nn 2132

__codedoc_split_function_auxi:w	225	__codedoc_typeset_expandability:	1156 , 1168 , 1187
__codedoc_split_function_-		__codedoc_typeset_function_-	
auxii:w	225	block:nN	901 , 1149
__codedoc_split_function_do:nn	225 , 253 , 292 , 827	__codedoc_typeset_functions:	38 , 1081 , 1134
__codedoc_str_if_begin:nnTF	120 , 182 , 185 , 190	\g__codedoc_typeset_implementation_-	
__codedoc_syntax:w	610 , 1233	bool	69 , 385 , 390 , 573 , 577 , 587 , 591 , 1774 , 1776 , 1779 , 1961 , 2326
\g__codedoc_syntax_box	14 , 39 , 1052 , 1055 , 1097 , 1236 , 1244 , 1264	__codedoc_typeset_rexp:	650 , 757 , 1172
\l__codedoc_syntax_coffin	9 , 11 , 1056 , 1096 , 1102 , 1119 , 1130	__codedoc_typeset_TF:	659 , 757 , 1155 , 1186 , 1501
\l__codedoc_syntax_dim	1233 , 1238 , 1249	__codedoc_typeset_variant_-	
__codedoc_syntax_end:	612 , 1233	list:nN	1158 , 1174
__codedoc_test_missing:n	1542 , 1543	\l__codedoc_undoc_def_tl	1957 , 1968 , 1987 , 2003
__codedoc_test_missing_aux:Nnn	1545 , 1550 , 1557	\l__codedoc_variants_prop	73
\g__codedoc_testfiles_seq	19 , 1511 , 1513	\g__codedoc_variants_seq	33 , 46 , 875 , 879 , 880 , 1157 , 1179 , 1180 , 1183 , 1425 , 1434
__codedoc_tmp:w	116 , 117	__codedoc_xmacro_code:n	58 , 1904
__codedoc_tmpa:w	81 , 229 , 245 , 2031 , 2048 , 2053 , 2067 , 2258 , 2261	__codedoc_xmacro_code:w	1904
\l__codedoc_tmpa_int	41 , 1618 , 1620 , 1622 , 2072 , 2075	\CodedocExplain	635
\l__codedoc_tmpa_seq	41 , 159 , 160 , 161 , 928 , 929 , 930 , 1398 , 1400 , 1574 , 1578 , 1598 , 1602 , 1604 , 1607 , 2292 , 2296 , 2300 , 2302	\CodedocExplainEXP	635
\l__codedoc_tmpa_tl	30 , 31 , 35 , 41 , 251 , 253 , 729 , 735 , 738 , 800 , 804 , 807 , 809 , 810 , 811 , 812 , 813 , 815 , 868 , 869 , 872 , 873 , 929 , 935 , 1011 , 1013 , 1017 , 1196 , 1197 , 1210 , 1211 , 1480 , 1481 , 1482 , 1487 , 1488 , 1490 , 1552 , 1553 , 1554 , 1555 , 1591 , 1592 , 1594 , 1626 , 1627 , 1927 , 1931 , 1932 , 1933 , 1934 , 1943 , 1946 , 1949 , 2028 , 2042 , 2060 , 2070 , 2074	\CodedocExplainREXP	635
__codedoc_tmpb:w	81 , 2035 , 2037 , 2054 , 2055	\CodedocExplainTF	635
\l__codedoc_tmpb_tl	35 , 41 , 681 , 682 , 684 , 733 , 735 , 880 , 882 , 930 , 936 , 937 , 1940 , 1941 , 1942 , 1943	\CodelineIndex	2329
\l__codedoc_trial_width_dim	10 , 16 , 1080 , 1082 , 1085 , 1242	CodelineNo	466
__codedoc_trim_right:Nn	114 , 163 , 165 , 169 , 170 , 171	\CodelineNumbered	2334
__codedoc_typeset_aux:n	757 , 1177 , 1497	coffin commands:	
__codedoc_typeset_dates:	1144 , 1215	\coffin_clear:N	1054 , 1056 , 1057
\g__codedoc_typeset_documentation_-	69 , 384 , 389 , 571 , 575 , 580 , 584	\coffin_join:NnnNnnnn	1100 , 1104 , 1108 , 1117 , 1121 , 1125
__codedoc_typeset_exp:	641 , 757 , 1171	\coffin_new:N	10 , 11 , 12 , 13
		\coffin_typeset:Nnnnn	1112 , 1129
		\color	777 , 781 , 788
		\comment	26 , 582 , 589 , 1717
		\ConTeXt	535
		\cs	5 , 5 , 9 , 12 , 28 , 45 , 536 , 557 , 643
		cs commands:	
		\cs_generate_variant:Nn	83 , 84 , 85 , 86 , 87 , 88 , 89 , 90 , 91 , 92 , 93 , 94 , 95 , 96 , 97 , 98 , 99 , 100 , 101 , 102 , 103 , 104 , 113 , 119 , 246 , 726 , 797 , 897 , 904 , 1161 , 1167 , 1214 , 1431 , 1437 , 1499 , 1557 , 2131 , 2156
		\cs_gset:Npn	460 , 490 , 497 , 513 , 1674 , 1819 , 1863 , 1872 , 1879 , 1890 , 2081
		\cs_gset:Npx	1847
		\cs_gset_eq:NN	472 , 473 , 1889
		\cs_gset_protected:Npn	2093 , 2107
		\cs_if_exist:NTF	6 , 424 , 759 , 766 , 773 , 781 , 1481 , 1892

<code>\cs_if_exist_p:N</code>	6	<code>\def</code>	640, 649, 658, 1725, 1726, 1742, 1752, 1765, 1766
<code>\cs_new:Npn</code>	198, 200, 217, 219, 225, 227, 231, 238, 241, 277, 314, 326, 347, 348, 480, 552, 922, 1215	<code>\DeleteShortVerb</code>	522, 523
<code>\cs_new:Npx</code>	257	<code>\Describe</code>	3
<code>\cs_new_eq:NN</code>	81, 82, 489	<code>\DescribeOption</code>	1631
<code>\cs_new_protected:Npn</code>	114, 130, 148, 155, 247, 290, 295, 671, 727, 740, 742, 757, 764, 771, 786, 790, 798, 818, 825, 831, 849, 851, 861, 866, 885, 898, 923, 988, 998, 1009, 1022, 1034, 1040, 1044, 1050, 1071, 1074, 1078, 1087, 1092, 1094, 1134, 1149, 1162, 1168, 1174, 1189, 1201, 1234, 1254, 1334, 1347, 1375, 1379, 1388, 1390, 1404, 1422, 1432, 1438, 1447, 1463, 1472, 1494, 1500, 1506, 1519, 1543, 1550, 1558, 1565, 1584, 1589, 1596, 1611, 1791, 1925, 1937, 1958, 2010, 2024, 2119, 2134, 2157, 2185, 2219, 2227, 2251, 2256, 2263, 2268, 2277	<code>\DescribeRoutine</code>	3
<code>\cs_new_protected:Npx</code>	138	<code>\DescribeVariable</code>	3
<code>\cs_new_protected_nopar:Npn</code> ..	1073	<code>\description</code>	1724, 1741
<code>\cs_set:Npn</code>	116, 1408, 1679, 1689, 1813, 1814, 2031, 2037, 2053, 2055, 2258	<code>\detokenize</code>	1698
<code>\cs_set:Npx</code>	1826	dim commands:	
<code>\cs_set_eq:NN</code> ..	484, 556, 557, 558, 559, 1362, 1835, 1838, 1839, 1840, 1841, 1842, 1843, 1846, 1851, 1852, 1853, 1854, 1861, 1876, 1887, 1902	<code>\dim_compare_p:nNn</code>	1085
<code>\cs_set_protected:Npn</code>	229, 474, 1918	<code>\dim_new:N</code>	17, 1233
<code>\currentfile</code>	1813, 1814, 1844, 1848, 1850, 1892, 1893, 1895	<code>\dim_set:Nn</code>	1047, 1082, 1238
<code>\CurrentOption</code>	416	<code>\dim_zero:N</code>	491, 1080
D		<code>\ding</code>	769
<code>danger</code> (environment)	1673	<code>\DisableCrossrefs</code>	2335
<code>\date</code>	2357	<code>\DisableDocumentation</code>	4, 570
<code>\dbend</code>	1674, 1680, 1690	<code>\DisableImplementation</code>	4, 570
<code>ddanger</code> (environment)	1673	doc commands:	
<code>\DeclareDocumentCommand</code>	536, 538, 540, 546, 1530, 1540, 1541, 1773, 1775, 1777, 1788, 1798, 1806	<code>\g_doc_functions_seq</code>	4, 1164, 1969, 1985
<code>\DeclareDocumentEnvironment</code>	592, 603, 606	<code>\g_doc_macros_seq</code> ..	4, 1453, 1971, 1983
<code>\DeclareExpandableDocumentCommand</code> ..	481, 548, 550	<code>\DocInclude</code>	1808
<code>\DeclareFontShape</code>	456	<code>\docincludeaux</code>	1811, 1872
<code>\DeclareMathOperator</code>	2367	<code>\DocInput</code>	4, 9, 1798, 1850
<code>\DeclareOption</code>	381, 382, 387, 392, 394, 396, 398, 400, 402, 404, 406, 408, 410, 412, 414, 416	<code>\DocInputAgain</code>	4, 1806
<code>\DeclareRobustCommand</code> ..	566, 567, 568, 569	documentation (environment)	578
E		<code>\DoNotIndex</code>	46, 1401, 1442, 1449
<code>\edef</code>	748, 1656		
<code>\else</code>	1745, 1858		
<code>\emph</code>	1587, 1698, 1708, 1713		
<code>\EnableCrossrefs</code>	2330		
<code>\EnableDocumentation</code>	4, 570		
<code>\EnableImplementation</code>	4, 570		
<code>\encapchar</code>	1655, 1656, 1663, 1668, 2181, 2221		
<code>\end</code>	1146, 1256, 1257, 1265, 1683, 1692, 1920		
<code>\endcomment</code>	26, 584, 591, 1717		
<code>\enddescription</code>	1728, 1754		
<code>\endenumerate</code>	633		
<code>\endgraf</code>	617, 1679, 1689		
<code>\endgroup</code>	510, 1641, 1670, 1679, 1689, 1728, 1754, 1766		
<code>\endinput</code>	1786		
<code>\endtabbing</code>	1766		
<code>\endtheglossary</code>	1843, 1854		
<code>\endtrivlist</code>	1560		
<code>\endVerbatim</code>	473		
<code>\endverbatim</code>	471, 1712		
<code>\enquote</code>	662		
<code>\ensuremath</code>	741, 744, 755		
<code>\enumerate</code>	626		
<code>\env</code>	5, 566		

<code>\immediate</code> ..	1822, 1836, 1837, 1857, 1865	<code>\let</code>	1655
<code>implementation (environment)</code>	578	<code>\levelchar</code>	1667, 2174, 2221
<code>\include</code>	1816	<code>\list</code>	489
<code>\index</code>	1660, 1665, 2168, 2170	<code>\listparindent</code>	491
<code>\indexentry</code>	1867	<code>\llap</code>	1410, 1468
<code>\IndexPrologue</code>	2083	<code>\LoadClass</code>	428
<code>\IniTeX</code>	525	<code>\Lua</code>	525
<code>\input</code>	1795	<code>\LuaTeX</code>	525
<code>\InstanceKey</code>	1765		
<code>\InstanceSemantics</code>	1766		
int commands:		M	
<code>\int_case:nnTF</code>	2296	<code>macro (environment)</code>	7, 603
<code>\int_compare:nNnTF</code>		<code>\MacroFont</code>	1486
.....	107, 328, 330, 332, 336,	<code>\MacroLongFont</code>	1486, 1502
341, 900, 1369, 1562, 1620, 1622, 1815		<code>\MacroTopsep</code>	1406
<code>\int_compare:nTF</code>	1179, 1485, 1602	<code>\makebox</code>	779
<code>\int_eval:n</code>	1372	<code>\makelabel</code>	1408
<code>\int_gdecr:N</code>	1458	<code>\MakePercentComment</code>	1796
<code>\int_gincr:N</code>	1456	<code>\MakePercentIgnore</code>	1794
<code>\int_incr:N</code>	1349, 1470	<code>\MakePrivateLetters</code> 460, 1395, 1636, 2080	
<code>\int_new:N</code>	18, 43, 44, 62, 80	<code>\MakeShortVerb</code>	517, 518
<code>\int_set:Nn</code>	1618, 2072, 2075	<code>\maketitle</code>	2358
<code>\int_use:N</code>	1139, 1868	<code>\manual</code>	1673, 1674
<code>\c_zero</code>	67	<code>\marg</code>	5, 25, 561
iow commands:		<code>\marginpar</code>	1642
<code>\iow_char:N</code>	808, 810, 811	<code>\marginparsep</code>	1107, 1111, 1128
<code>\iow_close:N</code>	2005	<code>\marginparwidth</code> ...	486, 1085, 1111, 1242
<code>\iow_indent:n</code>	2102, 2321	<code>\markboth</code>	2086, 2315
<code>\iow_new:N</code>	1954	<code>\mbox</code>	686, 761, 768, 775
<code>\iow_newline:</code> 1963, 1974, 1979, 1988,		<code>\medskipamount</code>	1111, 1124
1994, 2016, 2017, 2018, 2033, 2039,		<code>\meta</code> ...	5, 5, 546, 559, 562, 564, 565, 1073
2044, 2046, 2057, 2062, 2063, 2065		<code>\midrule</code>	1220
<code>\iow_now:Nn</code>	1975, 1980, 1989	mode commands:	
<code>\iow_open:Nn</code>	1964	<code>\mode_if_math:TF</code>	686, 745, 1072
<code>\iow_term:n</code>	61, 1963, 2006	<code>\mode_leave_vertical:</code>	1263
<code>\item</code>	1420, 1677, 1687, 1725, 1744	msg commands:	
<code>\itemindent</code>	493	<code>\msg_error:nn</code>	1053, 1237
<code>\itshape</code>	778, 782	<code>\msg_error:nnn</code>	845, 994
		<code>\msg_error:nnnn</code>	1004, 1015
K		<code>\msg_info:nn</code>	425, 2106, 2325
<code>\kern</code>	531, 532, 782, 1690	<code>\msg_new:nnn</code>	357, 359,
keys commands:		361, 363, 368, 373, 375, 419, 2099, 2318	
<code>\keys_define:nn</code>	664, 942, 1268	<code>\msg_new:nnnn</code>	349
<code>\keys_set:nn</code>	677, 1027, 1338	<code>\msg_warning:nnnn</code>	1273
<code>\kill</code>	1770	<code>\msg_warning:nnnnn</code>	172
		<code>\multicolumn</code>	1223, 1229
L			
<code>\label</code>	1197, 1211	N	
<code>\langle</code>	744	<code>\nan</code>	2369
<code>\language</code>	751	<code>\NB</code>	6, 1694
<code>\LaTeX</code>	2342, 2353	<code>\newcommand</code>	561, 1631, 1639, 1650
<code>\LaTeXe</code>	2145	<code>\NewDescribeEnvironment</code>	3
<code>\leavevmode</code>	506, 1633	<code>\NewDocumentCommand</code>	
<code>\leftskip</code>	507, 508	570, 572, 574, 576, 635, 637,
		646, 655, 1696, 1702, 1808, 2345, 2369	

<code>\NewDocumentEnvironment</code> .. 578, 585, 609, 615, 624, 1706, 1717	<code>\prg_return_true:</code> .. 111, 125, 183, 186, 191, 333, 337, 342, 910, 918
<code>\newenvironment</code> 1675, 1685, 1719, 1735, 1759	<code>\PrintChanges</code> 1840, 1841, 1852, 2324
<code>\NewMacroEnvironment</code> 3	<code>\PrintDescribeEnv</code> 1644
<code>\nobreak</code> 509, 1586, 1748	<code>\PrintIndex</code> 1838, 1839, 1851, 2105
<code>\noindent</code> 1042, 1090, 1586, 1677, 1687, 1708, 1713	<code>\ProcessOptions</code> 427
<code>\nolinkurl</code> 566	prop commands:
<code>\normalfont</code> 1147	<code>\prop_get:NnNTF</code> 83, 102, 1552
<code>\normalsize</code> 1147	<code>\prop_gput:Nnn</code> 83, 104
<code>NOTE (environment)</code> 1704	<code>\prop_if_empty:NTF</code> 2029
<code>\NOTE</code> 6	<code>\prop_map_function:NN</code> 2047
	<code>\prop_new:N</code> 20, 74
	<code>\prop_put:Nnn</code> 83, 103, 1555
	<code>\protect</code> 1662, 1667
	<code>\providecommand</code> 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 563, 564, 565, 1502
	<code>\pTeX</code> 525
	Q
	quark commands:
	<code>\q_mark</code> 234, 235, 239, 240, 242, 313, 315
	<code>\q_no_value</code> 32, 675, 676, 1067, 1076, 1332, 1360, 1377
	<code>\quark_if_no_value:NTF</code> 706, 713, 2122
	<code>\quark_if_recursion_tail_stop-</code>
	<code>do:nn</code> 1945
	<code>\q_recursion_stop</code> 1929
	<code>\q_recursion_tail</code> 1929, 1944
	<code>\q_stop</code> ... 116, 117, 199, 217, 236, 239, 240, 242, 313, 315, 345, 2258, 2261
	<code>\quotearch</code> 2221, 2224
	R
	<code>\raggedbottom</code> 449
	<code>\raggedleft</code> 1643
	<code>\raggedright</code> 1250, 1884
	<code>\raisebox</code> 639, 648, 657
	<code>\rangle</code> 755
	<code>\RecordChanges</code> 2328
	regex commands:
	<code>\regex_replace_all:nnN</code> 694
	<code>\regex_replace_once:nnNTF</code> 991
	<code>\relax</code> 1748, 1810, 1837, 1839, 1841, 1846, 1889
	<code>\RequirePackage</code> .. 429, 430, 452, 453, 459
	<code>\rightskip</code> 504
	S
	<code>\sb</code> 741
	scan commands:
	<code>\scan_stop:</code> 11, 32, 123, 124, 836, 846, 873, 936, 1673
	<code>\scriptsize</code> 1224, 1230
<code>\NewDocumentEnvironment</code> .. 578, 585, 609, 615, 624, 1706, 1717	
<code>\newenvironment</code> 1675, 1685, 1719, 1735, 1759	
<code>\NewMacroEnvironment</code> 3	
<code>\nobreak</code> 509, 1586, 1748	
<code>\noindent</code> 1042, 1090, 1586, 1677, 1687, 1708, 1713	
<code>\nolinkurl</code> 566	
<code>\normalfont</code> 1147	
<code>\normalsize</code> 1147	
<code>NOTE (environment)</code> 1704	
<code>\NOTE</code> 6	
O	
<code>\oarg</code> 5, 561	
<code>\obeylines</code> 1252	
<code>\obeyspaces</code> 1251	
<code>\oddsidemargin</code> 487	
<code>\OnlyDescription</code> 1773, 2336	
<code>\OnlyDocumentation</code> 4	
<code>\openout</code> 1836	
P	
<code>\pageref</code> 1627	
<code>\par</code> 509, 1042, 1046, 1514, 1528, 1532, 1538, 1587, 1679, 1683, 1689, 1692, 1708, 1713, 1725, 1733, 1750, 1758, 1772	
<code>\parbox</code> 1881	
<code>\parfillskip</code> 505, 1748	
<code>\parg</code> 5, 561	
<code>\parindent</code> 492, 504	
<code>\parskip</code> 494	
<code>\part</code> 1844, 2085, 2313	
<code>\partname</code> 495	
<code>\PassOptionsToClass</code> 416, 418	
<code>\PassOptionsToPackage</code> 411	
<code>\pdfstringdefDisableCommands</code> .. 483, 554	
<code>\pdfstringnewline</code> 480	
<code>\pdfTeX</code> 525	
<code>\penalty</code> 1746, 1747	
<code>\phantom</code> 782	
<code>\pkg</code> 5, 566, 2149, 2349	
<code>\prevdepth</code> 1047	
prg commands:	
<code>\prg_break:</code> 261, 263, 267, 273	
<code>\prg_break_point:</code> 275	
<code>\prg_generate_conditional_-variant:Nnn</code> 128	
<code>\prg_new_conditional:Npnn</code> .. 312, 905	
<code>\prg_new_protected_conditional:Npnn</code> 105, 120, 180	
<code>\prg_return_false:</code> 110, 126, 192, 194, 333, 337, 342, 910, 918	

seq commands:	
\seq_clear:N	803
\seq_clear_new:N	856
\seq_count:N	1179, 1602, 1619, 2296
\seq_gclear:N	875, 1629
\seq_get:NN	872, 930
\seq_gpop:NN	880
\seq_gput_right:Nn	84, 1164, 1384, 1389, 1453, 1513, 1576
\seq_gset_eq:NN	879
\seq_gset_filter:NNn	1613
\seq_if_empty:NTF	1157, 1425, 1616, 2051
\seq_if_in:NnTF	1511, 1971, 1985
\seq_item:Nn	1607, 1625, 2300, 2302
\seq_map_function:NN	44, 821, 2066
\seq_map_inline:Nn	1434, 1969, 1983
\seq_map_variable:NNn	161
\seq_new:N	4, 5, 21, 22, 46, 48, 49
\seq_pop:NN	868, 929
\seq_pop_left:NN	160
\seq_put_right:Nn	804, 857, 859
\seq_set_eq:NN	928
\seq_set_filter:NNn	1574
\seq_set_from_clist:Nn	46, 814, 1397
\seq_set_map:NNn	1598
\seq_set_split:Nnn	83, 83, 159, 2292
\seq_use:Nn	46, 1180, 1183, 1400, 1547, 1578
\seq_use:Nnnn	51, 1604
\setcounter	466, 2082
\setlength	485, 492, 493, 494, 502
\sffamily	470
\small	619, 1136, 1245, 1504
\space	1816, 2034, 2040, 2058
\SpecialIndex	2093
\SpecialOptionIndex	1646, 1650
\star	762
\StopEventually	3, 4, 1773
str commands:	
\c_backslash_str	14, 477, 539, 544, 709, 794, 2201, 2214, 2238
\c_percent_str	808
\str_case:nn	83, 85, 2282
\str_case:nnTF	202, 2165
\str_count:n	1485
\str_head:N	2282
\str_if_eq:nnTF	873, 936, 2142, 2162
\str_if_eq_p:nn	284, 285
\str_tail:n	2280
\c_underscore_str	793
\string	640, 649, 658, 1816, 1822, 1867, 2145, 2149
\strut	1368, 1474, 1883
\subsection	1721, 1737
\subsubsection	1729, 1738, 1755, 1761, 1762, 1768
syntax (environment)	6, 609
sys commands:	
\c_sys_day_int	1002, 1013
\c_sys_jobname_str	1964, 2103, 2322
\c_sys_month_int	1002, 1013
\c_sys_year_int	1002, 1013
T	
\tabbing	1769
\tabcolsep	1140
\Team	2340
\TemplateArgument	1725
\TemplateKey	1742, 1745
\TemplateSemantics	1726, 1752
\testfile	1362, 1540
\TestFiles	7
\TestFiles	7, 1530
\TestMissing	7
\TestMissing	7, 1541
\TeX	619, 643, 2145
TeX and L ^A T _E X 2 _ε commands:	
\@	28, 462, 702, 1608
\@M	1723, 1740, 1764
\@addtoreset	467, 496
\@auxout	640, 649, 658, 1815, 1835, 1861
\@beginparpenalty	1723, 1740, 1764
\@bsphack	1634, 1652, 1781, 2095
\@currenvir	2162
\@docinclude	1817, 1819
\@dottedtocline	514
\@eha	1816
\@esphack	1647, 1671, 1784, 2097
\@evenfoot	1887, 1902
\@indexfile	1865
\@input	1822
\@latexerr	381, 1816
\@ltxdoc@PrintChanges	1840, 1852
\@ltxdoc@PrintIndex	1838, 1851
\@ltxdoc@endtheglossary	1843, 1854
\@ltxdoc@theglossary	1842, 1853
\@mainaux	1822, 1861
\@nameuse	1859
\@oddfoot	1879, 1887, 1890, 1902
\@partaux	1815, 1835, 1836, 1837, 1857
\@partlist	1827
\@plus	501
\@pnumwidth	504, 505, 509
\@secpenalty	500
\@tempa	1829
\@tempb	1826, 1829
\@tempdima	502, 507

\@tempswafalse	1825	\verbatim@font	28, 63, 688, 1949, 2178
\@tempswattrue	1823, 1830	\xmacro@code	58, 1904
\@writeckpt	1856	\z@	499, 504
\@xobeysp	28, 700	tex commands:	
_	51	\tex_interactionmode:D	2072, 2075
\bottomrule	33, 40	\tex_lowercase:D	1915
\c@CodelineNo	80, 1456, 1458, 1868	texnote (environment)	6, 615
\c@HD@hypercount	1139, 1658, 2182	\text	1949, 2369
\c@tocdepth	499	\textbackslash	551
\check@checksum	1782	\textbf	619, 1765
\Codedoc@explTF	658, 773, 781	\textcolor	470
\Codedoc@expstar	640, 759	\textit	1523, 1533
\Codedoc@rexpstar	649, 766	\textrm	1182, 1183, 1184
\codeline@wrindex	1863, 2163, 2167	\textsf	568, 569, 1526, 1536
\Describe@Option	1637, 1639	\texttt	562, 564, 565, 567, 628, 642, 644, 652, 653, 659, 660, 1594, 1708, 2369
\DocInput	56, 56	\textwidth	485, 1089, 1240, 1881
\DoNotIndex	23	\thanks	2342, 2353
\endtrivlist	47	\the	749, 1658, 2182
\expanded@notin	476	\theCodelineNo	468
\g@addto@macro	2080	\theglossary	1842, 1853
\hb@xt@	509	\theindex	2080
\HD@savestfalse	1137	\thepage	1900
\HD@target	1138, 1370, 1654, 2170	\thepart	1848, 1874, 1875, 1893, 1896
\HDorg@encapchar	1655, 1658	\tiny	470
\HDorg@theCodelineNo	469	\title	2347
\Hy@MakeCurrentHref	1139	tl commands:	
\if@partsw	1824	\c_empty_tl	67
\if@tempswa	1834	\c_space_tl	1548, 2178, 2215
\ifnot@excluded	474	\tl_clear:N	820, 1361, 1927, 1966, 1967, 1968, 2021, 2028, 2232
\include	56	\tl_const:Nn	56, 58, 2340
\index@excludelist	23, 478	\tl_count:n	83, 86, 108, 109
\index@prologue	1877	\tl_gclear:N	1793, 1871
\init@checksum	1783	\tl_gput_right:Nn	1877, 2105, 2324
\input	56	\tl_greplace_all:Nnn	83, 87, 870
\it@is@a	63, 2107	\tl_gset:Nn	495, 1782, 1948
\l@nohyphenation	751	\tl_gset_eq:NN	869
\l@section	496	\tl_if_empty:NTF	132, 1221, 1227, 1381, 1392, 2012, 2070, 2290
\l@subsection	496	\tl_if_empty:nTF	88, 318, 319, 320, 322, 323, 324, 379, 913, 2139, 2173
\m@ne	750	\tl_if_empty_p:N	1218, 1219
\macro@namepart	23, 477	\tl_if_eq:nnTF	1509
\meta	30	\tl_if_head_eq_charcode:nNTF	83, 89, 90, 91, 299, 835, 2237, 2265, 2270, 2279
\meta@font@select	747	\tl_if_head_eq_meaning:nNTF	83, 92
\meta@hyphen@restore	748, 753	\tl_if_in:NnTF	83, 94, 95, 96, 260, 853, 2233, 2240, 2243, 2246
\midrule	40	\tl_if_in:nnTF	83, 93, 122, 262, 264, 269, 1928
\nfss@text	745	\tl_map_function:NN	863
\p@	501	\tl_map_inline:Nn	33, 925
\part	3, 62, 68		
\partname	3		
\saved@indexname	47, 1460		
\saved@macroname	47, 1440		
\texttt	50, 51		
\toprule	40		
\trivlist	47		
\verb	63		

