THE CNLTX BUNDLE

Documentation for LaTeX 2ε Packages or Classes

v0.0 2013/09/08

LaTeX examples the CN way

Clemens Niederberger

https://github.com/cgnieder/cnltx

contact@mychemistry.eu

A bundle of packages and classes for consistent format of control sequences, package options, source code with examples, writing a package manual (including an index containing the explained control sequences, options, …).

Table of Contents

1 Background

The CNLTX bundle contains of different packages and classes. I developed them as a successor of a class that I used for writing the documentation of my packages with the intention of a cleaner interface and less unnecessary ballast. Hence the separation into package and class. The package provides a source code environment that also prints the output and defines quite a number of macros for formatting of control sequence names, package names, package options and so on. The best documentation for the bundle as always is the source code but I'm trying to provide a documentation as comprehensive as possible.

2 Bundled Packages and Classes

The CNLTX bundle currently bundles the following packages and classes:

- CNLTX-BASE – defines base macros for error-messaging, expansion control and tokenlist manipulation. It also provides color definitions and defines a few color schemes for the CNLTX-DOC class. All other packages and classes of the cnbundle load this package.
- CNLTX-EXAMPLE – defines macros and environments for describing control sequences and options and for including source code.
- CNLTX-CSNAMES – defines a list of highlighted control sequence names, loaded by CNLTX-EXAMPLE.
- CNLTX-DOC – a class for writing package manuals. Loads CNLTX-EXAMPLE.

3 License and Requirements

 The package has the status "maintained."

The CNLTX-BASE package loads the following packages: pgfopts,[1] etoolbox,[2] trimspaces[3] and xcolor.[4]

The CNLTX-EXAMPLE package loads the following packages: CNLTX-BASE, listings,[5] accsupp,[6] mdframed[7] and idxcmds.[8]

1. on CTAN as `pgfopts`: `http://mirrors.ctan.org/macros/latex/contrib/pgfopts/`
2. on CTAN as `etoolbox`: `http://mirrors.ctan.org/macros/latex/contrib/etoolbox/`
3. on CTAN as `trimspaces`: `http://mirrors.ctan.org/macros/latex/contrib/trimspaces/`
4. on CTAN as `xcolor`: `http://mirrors.ctan.org/macros/latex/contrib/xcolor/`
5. on CTAN as `listings`: `http://mirrors.ctan.org/macros/latex/contrib/listings/`
6. on CTAN as `accsupp`: `http://mirrors.ctan.org/macros/latex/contrib/oberdiek/accsupp/`
7. on CTAN as `mdframed`: `http://mirrors.ctan.org/macros/latex/contrib/mdframed/`
8. on CTAN as `idxcmds`: `http://mirrors.ctan.org/macros/latex/contrib/idxcmds/`

The CNLTX-DOC class loads the package with the same name and additionally the following packages: CNLTX-BASE, CNLTX-EXAMPLE, ulem,[9] multicol,[10] ragged2e,[11] marginnote[12] and hyperref.[13] It is a wrapper class for the KOMA-Script class scrartcl.[14]

Like all of my packages CNLTX implicitly relies on an up to date TEX distribution.

4 Options and Setup

The CNLTX bundle has a number of options. The CNLTX-DOC class only knows the `load-preamble` (described in section 8.3) as a *class* option. All other options regardless if they're defined by a package or a class can be set with a setup command:

`\setcnltx{`⟨*options*⟩`}`
setup command for CNLTX.

The source code environments defined by the CNLTX-EXAMPLE package also have optional arguments that can be used to set the options for the environment locally.

5 Available Commands

5.1 Description of Macros, Environments and Options

provided by the CNLTX-EXAMPLE package

The commands described in this section all are provided by the CNLTX package. They all are related to the typesetting of provided macros, options and the like.

`\code{`⟨*arg*⟩`}`
Formatting of source code. This is *no* verbatim command. Used internally in the following commands.

`\cs*{`⟨*name*⟩`}`
Format the control sequence ⟨*name*⟩, `\cs{name}`: `\name`. Adds a corresponding index entry. The starred form does not add an index entry.

`\csidx{`⟨*name*⟩`}`
Adds an index entry but does not typeset the control sequence ⟨*name*⟩.

`\env*{`⟨*name*⟩`}`
Format the environment ⟨*name*⟩, `\env{name}`: `name`. Adds a corresponding index entry with a hint that the entry refers to an environment. The starred form does not add an index entry.

`\envidx{`⟨*name*⟩`}`
Adds an index entry but does not typeset the environment ⟨*name*⟩.

9. on CTAN as `ulem`: `http://mirrors.ctan.org/macros/latex/contrib/ulem/`
10. on CTAN as `multicol`: `http://mirrors.ctan.org/macros/latex/required/tools/multicol/`
11. on CTAN as `ragged2e`: `http://mirrors.ctan.org/macros/latex/contrib/ms/ragged2e/`
12. on CTAN as `marginnote`: `http://mirrors.ctan.org/macros/latex/contrib/marginnote/`
13. on CTAN as `hyperref`: `http://mirrors.ctan.org/macros/latex/contrib/hyperref/`
14. on CTAN as `koma-script`: `http://mirrors.ctan.org/macros/latex/contrib/koma-script/`

`\meta{`⟨*meta*⟩`}`
Description of an argument, `\meta{meta}`: ⟨*meta*⟩.

`\marg{`⟨*arg*⟩`}`
A mandatory argument. ⟨*arg*⟩ is formatted with `\meta` if it is not blank, `\marg{arg}`: {⟨*arg*⟩}.

`\oarg{`⟨*arg*⟩`}`
An optional argument. ⟨*arg*⟩ is formatted with `\meta` if it is not blank, `\oarg{arg}`: [⟨*arg*⟩].

`\darg{`⟨*arg*⟩`}`
An argument with parentheses as delimiters. ⟨*arg*⟩ is formatted with `\meta` if it is not blank, `\darg{arg}`: (⟨*arg*⟩).

`\sarg`
An optional star argument, `\sarg`: ∗.

`\option*{`⟨*name*⟩`}`
An option ⟨*name*⟩, `\option{name}`: name. Adds a corresponding index entry. The starred form does not add an index entry.

`\optionidx{`⟨*name*⟩`}`
Adds an index entry but does not typeset the option ⟨*name*⟩.

`\key*{`⟨*name*⟩`}{`⟨*values*⟩`}`
A key ⟨*name*⟩ with values ⟨*values*⟩, `\key{key}{value}`: key = {⟨*value*⟩}; `\key*{key}{value}`: key = ⟨*value*⟩

`\choices{`⟨*clist of choices*⟩`}`
A list of choices, `\choices{one,two,three}`: one|two|three

`\choicekey{`⟨*name*⟩`}{`⟨*clist of choices*⟩`}`
A key ⟨*name*⟩ with a list of possible values, `\choicekey{key}{one,two,three}`: key = one|two|three

`\boolkey{`⟨*name*⟩`}`
A boolean key ⟨*name*⟩ with choices `true` and `false`, `\boolkey{key}`: key = true|false

`\default{`⟨*value*⟩`}`
Markup for a default choice, `\choices{one,\default{two},three}`: one|two|three

5.2 Versioning Commands, Licensing and Related Stuff

provided by the CNLTX-DOC class

The commands described in this section are provided by the CNLTX class except where indicated differently. These commands are related to information about the legal stuff of a package and where to find it on th world wide web.

`\sinceversion{`⟨*version*⟩`}`

Introduced in version 0.0

Gives a sidenote like the one on the left.

`\changedversion{`⟨*version*⟩`}`
Changed in version 0.0
Gives a sidenote like the one on the left.

`\lppl`
Typesets "LPPL" and adds a corresponding index entry.

`\LPPL`
Typesets "LaTeX Project Public License" and adds a the same index entry as `\lppl`.

`\license*`
Typesets 'Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License, version 1.3 or later (`http://www.latex-project.org/lppl.txt`). The package has the status "maintained."'. The un-starred variant adds a `\par`.

`\ctan`
Typesets "CTAN" and adds a corresponding index entry.

`\CTAN`
Typesets "Comprehensive TeX Archive Network" and adds the same index entry as `\ctan`.

`\pkg*{`⟨*package*⟩`}`
provided by the CNLTX-EXAMPLE package
Format the package name ⟨*package*⟩ and add an index entry. The starred variant adds nothing to the index.

`\pkgidx{`⟨*package*⟩`}`
provided by the CNLTX-EXAMPLE package
Add an index entry for the package ⟨*package*⟩.

`\cls*{`⟨*class*⟩`}`
provided by the CNLTX-EXAMPLE package
Format the class name ⟨*class*⟩ and add an index entry. The starred variant adds nothing to the index.

`\clsidx{`⟨*class*⟩`}`
provided by the CNLTX-EXAMPLE package
Add an index entry for the class ⟨*class*⟩.

`\CTANurl[`⟨*[*⟩`]`directory `{`⟨*name*⟩`}]`
Writes a CTAN link like the ones in section 3 in the footnotes. The predefined directory is `macros/latex/contrib`.

5.3 Formatting Commands

One of the goals I wanted to achieve with this package is a consistent look and an easy interface for customization. No font choice and no color choice is fixed. In this section ways to change the formatting are shown.

The formatting of the different commands provided by CNLTX can be changed in two ways: either by redefining the internal commands that are used for the formatting or by setting a corresponding option. Both variants are described in the next subsections.

How the colors should be changed is described in section 9.

5.3.1 Formatting by Redefining Hooks

You can change the formatting by redefining the following commands. They're all defined by the CNLTX package except where indicated differently.

`\codefont` Default: `\ttfamily`
This command is used for all formatting of source code.

`\sourceformat` Default: `\codefont\small`
Formatting of the listings.

`\exampleformat` (initially empty)
Special formatting of the output of a listing.

`\versionnoteformat` Default: `\footnotesize\sffamily\RaggedRight`
provided by the CNLTX-DOC class
Formatting of the notes introduced in section 5.2.

`\packageformat` Default: `\sffamily`
The formatting of package names.

`\classformat` Default: `\sffamily`
The formatting of class names.

`\argumentformat` Default: `\normalfont\itshape`
The formatting of `\meta{⟨meta⟩}`.

```
\renewcommand*\codefont{\sffamily\bfseries}
\code{foo} and \cs*{bar}, option \option{baz}
```

---

foo and **\bar**, option **baz**

5.3.2 Formatting by Setting Options

You can change the formatting by setting the following options. They're all defined by the CNLTX package except where indicated differently.

`code-font` = {⟨*definition*⟩} Default: `\ttfamily`
Used for all formatting of source code.

`source-format` = {⟨*definition*⟩} Default: `\codefont\small`
Formatting of the listings.

`expl-format` = {⟨*definition*⟩} (initially empty)
Special formatting of the output of a listing.

`version-note-format` = {⟨*definition*⟩} Default: `\footnotesize\sffamily\RaggedRight`

provided by the CNLTX-DOC class

Formatting of the notes introduced in section 5.2.

`pkg-format` = {⟨*definition*⟩} Default: `\sffamily`

The formatting of package names.

`cls-format` = {⟨*definition*⟩} Default: `\sffamily`

The formatting of class names.

`arg-format` = {⟨*definition*⟩} Default: `\normalfont\itshape`

The formatting of `\meta{`⟨*meta*⟩`}`.

```
\setcnltx{code-font=\sffamily\itshape}
\code{foo} and \cs*{bar}, option \option{baz}
```

---

foo and *\bar*, option *baz*

6 Available Environments

CNLTX defines a few environments most of them related to the way how descriptions of commands are typeset.

The `example` environment is defined by the CNLTX package, the others are defined by the class.

`\begin{example}[`⟨*options*⟩`]`

This environment is a formatted verbatim environment that also inputs the output of the inputted code. This environment is described in section 7.4.

`\begin{sourcecode}[`⟨*options*⟩`]`

This environment is a formatted verbatim environment. This environment is described in section 7.4.

`\begin{commands}`

A description-like environment for describing commands. While this environment is a list internally and thus recognizes `\item` own commands are used to describe macros. They are explained in section 7.1.

`\begin{options}`

A description-like environment for describing options. While this environment is a list internally and thus recognizes `\item` own commands are used to describe options. They are explained in section 7.2.

`\begin{environments}`
A description-like environment for describing environments. While this environment is a list internally and thus recognizes `\item` own commands are used to describe environments. They are explained in section 7.3.

Except for the `example` and the `sourcecode` environments the environments are lists all using the same internal `\list`. The setup uf this list can be changed via an option:

`list-setup` = {⟨*definitions*⟩}
Default: `\leftmargin=0pt \labelwidth=2em \labelsep=0pt \itemindent=-1em`
The setup of the `\list` used by the `commands`, `options` and `environments` environments.

7 Usage

7.1 Command Descriptions

Inside of the environment `commands` that was introduced in section 6 items are input via the following command:

`\command*{`⟨*name*⟩`}[`⟨*stuff after*⟩`]`
This macro formats a control sequence with `\cs` and puts a line break after it. The optional argument allows printing things directly after the command name and can thus be used for adding arguments.

`\Default{`⟨*code*⟩`}`
This command can be placed after `\command` in order to give a default definition. The definition will then be placed on the same line flush right.

```
\begin{commands}
  \command{cs}
    This is about foo bar baz.
  \command{cs}[\marg{arg}]
    This one has an argument.
  \command{cs}[\sarg\oarg{option}]
    This has a star variant and an optional argument.
  \command{cs}\Default{foo bar}
    This one has the default replacement text \code{foo bar}
\end{commands}
```

`\cs`
This is about foo bar baz.

`\cs{`⟨*arg*⟩`}`
This one has an argument.

`\cs*[`⟨*option*⟩`]`
This has a star variant and an optional argument.

`\cs` Default: `foo bar`
This one has the default replacement text `foo bar`

7.2 Option Descriptions

The `options` environment knows a few more commands to meet all the different kinds of options.

`\opt*`
An option. The star prevents an index entry.

`\keyval*{`⟨*key*⟩`}{`⟨*value*⟩`}`
A key/value option. The star prevents an index entry.

`\keychoice*{`⟨*key*⟩`}{`⟨*list of choices*⟩`}`
A key/value option where the value is one of a list of choices. The star prevents an index entry.

`\keybool*{`⟨*name*⟩`}`
A boolean key, that ist a choice key with choices `true` and `false`. The star prevents an index entry.

```
\begin{options}
  \opt{foo}
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
  \opt*{foo}\Default{bar}
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
  \keyval{foo}{bar}\Default
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
  \keyval*{foo}{bar}
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
  \keychoice{foo}{one,two,three}
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
  \keybool{foo}
    This makes stuff.  Let's add a few more words so that the line gets
    filled and we can see how the output actually looks.
\end{options}
```

---

foo
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

foo Default: bar
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

foo = {⟨*bar*⟩} (initially empty)
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

foo = ⟨*bar*⟩
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

foo = one|two|three
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

foo = true|false
This makes stuff. Let's add a few more words so that the line gets filled and we can see how the output actually looks.

7.3 Environment Descriptions

Environment descriptions are made – unsurprisingly – with the `environments` environment. It knows the command `\environment`:

`\environment*{`⟨*name*⟩`}[`⟨*stuff after*⟩`]`
This macro prints the environment name and puts a line break after it. The optional argument allows printing things directly after the environment name and can thus be used for adding arguments.

```
\begin{environments}
  \environment*{foobar}[\oarg{options}]
    This is environment \env*{foobar}.  The star prevents it from being
    added to the index.
\end{environments}
```

`\begin{foobar}[`⟨*options*⟩`]`
This is environment `foobar`. The star prevents it from being added to the index.

7.4 Example Code

Example code can be included through the `example` environment or the `sourcecode` environment.

```
\begin{example}
  a \LaTeX\ code example
\end{example}
```

This example would give:

```
a \LaTeX\ code example
```

---

a LaTeX code example

Both environments can be influenced by options:

`code-only` = `true`|`false` Default: `false`
Only typeset the code as code but don't include it afterwards. The code box above is an example for the usage of this option. This option has no effect on the `sourcecode` environment: this is already what that environment does.

`side-by-side` = `true`|`false` Default: `false`
Typeset source and output side by side. The code is input on the left and the output on the right. Side by side examples are typeset in `minipage` environments with all consequences that come with them (think of `\parindent` ...).

`code-sep` = {⟨*definition*⟩} Default: `\hrulefill`
Code that is inserted between a source code and the corresponding output when printed below each other.

The same example again, this time using `side-by-side`:

```
a \LaTeX\ code example
```

a LaTeX code example

The frame around the examples is done by the mdframed package. It is of course possible to customize it:

`add-frame-options` = {⟨*mdframed options*⟩} (initially empty)
Add options to the predefined ones.

`frame-options` = {⟨*mdframed options*⟩}
Default: `backgroundcolor=cnltxbg,linecolor=cnltx,roundcorner=5pt`
Overwrite the options with new ones.

The source code is formatted using the listings package. Similar options exist to adapt listings' options that are used for formatting the source code. The predifined style has many options that will not be mentioned here. If you're interested you can find them in `cnltx-csnames.sty`.

`gobble` = {⟨*integer*⟩} Default: 2
The number of initial characters that is gobbled from each line.

`add-cmds` = {⟨*list of csnames*⟩} (initially empty)
A list of control sequence names that should be recognized as a command sequence in the source code examples and should be formatted accordingly. The control sequence names in this list will also get an index entry when they're used in the source example. This is done internally via `\csidx`. The option should be used to add the new commands that are defined by the package for which you are writing the manual for.

`add-silent-cmds` = {⟨*list of csnames*⟩}
A list of control sequence names that should be recognized as a command sequence in the source code examples and should be formatted accordingly. The control sequence names in this list will *not* get an index entry when they're used in the source example. There already is quite a large but far from comprehensive list of silent commands but many are still missing. This option allows you to extend the list on a per document basis.

`add-listings-options` = {⟨*listings options*⟩} (initially empty)
Additional options for the listings environments.

`listings-options` = {⟨*listings options*⟩}
Overwrite existing options with new ones. This can be used to build an own style from scratch.

`add-envs` = {⟨*list of environment names*⟩} (initially empty)
Like `add-cmds` but for environment names.

8 Package or Class Information, Building of the Manuals Title Page

8.1 Package or Class Information

A manual for a package or a class needs some information like the package name, the version number, the date and so on. This information is given with the following options. They are used to build the title page of the manual.

`package` = {⟨*package*⟩}
The name of the package that is described. Either this option or `class` or `name` should always be given. This command also defines a command sequence from the package name that formats the package name with color and small caps like CNLTX.

`class` = {⟨*class*⟩}
The name of the class that is described. Either this option or `package` or `name` should always be given. This command also defines a command sequence from the class name that formats the class name with color and small caps like CNLTX.

`name` = {⟨*name*⟩}
The name of the class/package that is described. Either this option or `package` or `class` should always be given. This command also defines a command sequence from the class name that formats the class name with color and small caps like CNLTX.

`authors` = {⟨*author list*⟩}
Comma separated list of package/class authors.

`version` = {⟨*version number*⟩}
Version number of the package/class. CNLTX tries to extract the information from the given `package` or `class`. This option can be used to set it explicitly.

`date` = {⟨*date*⟩}
Date of the package/class. CNLTX tries to extract the information from the given `package` or `class`. This option can be used to set it explicitly.

`info` = {⟨*package/class info*⟩}
Information about the package/class. CNLTX tries to extract the information from the given `package` or `class`. This option can be used to set it explicitly.

`subtitle` = {⟨*subtitle*⟩}
A subtitle that is typeset *instead* of the package/class info.

`url` = {⟨*url*⟩}
The homepage of the package.

`email` = {⟨*email*⟩}
A contact email address.

`abstract` = {⟨*abstract*⟩}
An abstract of the package/class/manual. This is text typeset in a box of `.75\linewidth`. Actually it does not have to be text but could be an image or whatever you like.

8.2 Building of the Manuals Title Page

If either the `package` or `class` has been given an automatic title page is built using the gathered information. Figure 1 roughly sketches which informations is used and how the different elements are arranged on the title page. The page style of the title page is `plain`. Additionally a table of contents is automatically built that is set in two columns.

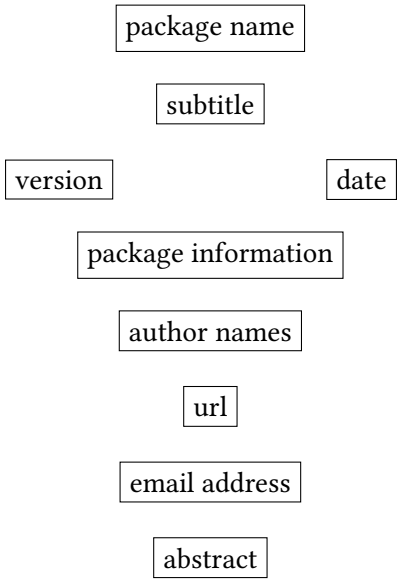

FIGURE 1: Schematic sketch of the title page.

8.3 Predefined Preamble

It is posssible to load a part of my standard preamble automatically by passing an option as class option.

`load-preamble`
Class option that preloads part of my custom preamble.

Using the option will include the following code:

```
\RequirePackage[oldstyle]{libertine}
\RequirePackage{libertinehologopatch}% not on CTAN, yet!
\RequirePackage[supstfm=libertinesups]{superiors}
\RequirePackage{microtype}
\RequirePackage[scaled=.83]{beramono}
\RequirePackage{fnpct}
\RequirePackage[english]{babel}
\renewcommand*\othersectionlevelsformat[3]{%
  \textcolor{cnltx}{#3\autodot}\enskip}
\renewcommand*\partformat{%
  \textcolor{cnltx}{\partname~\thepart\autodot}}
\deffootnote{2em}{1em}{\llap{\thefootnotemark. }}%
\pagestyle{headings}
\setcapindent{1.5em}
\setkomafont{caption}{\normalfont\footnotesize\sffamily}
\setkomafont{captionlabel}{\normalfont\footnotesize\sffamily\scshape}
```

9 Predefined Colors and Color-Schemes

9.1 Explicitly Defined Colors

The CNLTX-BASE package defines a number of colors:

`cnltxbrown` Per default used for the control sequences.

`cnltxblue` Unused per default.

`cnltxred` Per default used as base color in various places.

`cnltxgreen` Unused per default.

`cnltxgray` Per default used for formatting comments.

`cnltxyellow` Per default used for options.

`cnltxformalblue` Unused per default.

`cnltxformalred` Unused per default.

9.2 Actual Used Color Names and Color Schemes

The colors defined in section 9.1 are not directly used with those names. Instead colors are used whose names describe their function rather than the color. For this the color names are mapped to actual colors and saved as a coloring scheme. There are currently three predefined color schemes whose definitions are given below. Those definitions also show the actually used color names:

The 'default' color scheme is defined as follows:

```
\cnltx@define@colorscheme{default}{
  cs          => cnltxbrown , % command sequences
  option      => cnltxyellow ,% options
  comment     => cnltxgray ,  % comments
  beginend    => red ,        % \begin and \end
  env         => black ,      % environment names
  argument    => black ,      % argument delimiters
  meta        => black!80 ,   % arguments of \meta
  cnltx       => cnltxred ,   % base color
  cnltxbg     => white ,      % source code box background
  link        => black!90 ,   % hyperlinks
  versionnote => black!75     % versioning notes text
}
```

The 'blue' color scheme is defined this way:

```
\cnltx@define@colorscheme{blue}{
  cs          => cnltxbrown ,
  option      => cnltxgreen ,
  comment     => cnltxgray ,
  beginend    => red ,
  env         => black ,
  argument    => black ,
  meta        => black!80 ,
  cnltx       => cnltxblue ,
  cnltxbg     => yellow!10 ,
  link        => cnltx ,
  versionnote => black!75
}
```

Finally the 'formal' color scheme is defined like this:

```
\cnltx@define@colorscheme{formal}{
  cs          => black ,
  option      => cnltxformalblue ,
  comment     => cnltxgray ,
  beginend    => red ,
  env         => black ,
  argument    => black ,
  meta        => black!80 ,
  cnltx       => cnltxformalblue ,
  cnltxbg     => white ,
  link        => black!90 ,
  versionnote => black!75
}
```

10 Internal Helper Commands

The commands in this section are only described for the sake of completeness. They are not meant to be used in a document.

10.1 Defined by CNLTX-BASE

Especially CNLTX-BASE defines some useful helper macros that are also used by the other packages and classes.

`\cnltx@@date`
The creation date of the current version of the bundle.

`\cnltx@@version`
The version number of the bundle.

`\cnltx@@info`
The short description of the bundle.

`\cnltx@create@message{`⟨*module*⟩`}{Error|Warning|WarningNoLine|Info}`
Create suiting error and warning messaging commands for the module ⟨*module*⟩.

`\cnltx@base@error{`⟨*message*⟩`}`
Issue an error message using `\PackageError`.

`\cnltx@base@warning{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarning`.

`\cnltx@base@warningnoline{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarningNoLine`.

`\cnltx@base@info{`⟨*message*⟩`}`
Issue a message using `\PackageInfo`.

`\cnltx@fullexpand@onearg{`⟨*cs*⟩`}{`⟨*argument*⟩`}`
Exhaustive expansion of ⟨*argument*⟩ before it is passed as argument to ⟨*cs*⟩.

`\cnltx@fullexpand@twoargs{`⟨*cs*⟩`}{`⟨*argument1*⟩`}{`⟨*argument2*⟩`}`
Exhaustive expansion of ⟨*argument1*⟩ and ⟨*argument2*⟩ before they're passed as argumenst to ⟨*cs*⟩.

`\cnltx@stripbs`
A shortcut for `\expandafter\@gobble\string`.

`\cnltx@if@in{`⟨*tokenlist*⟩`}{`⟨*search*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
Places ⟨*true*⟩ in the input stream if ⟨*search*⟩ is found in ⟨*tokenlist*⟩ and ⟨*false*⟩ if it isn't.

`\cnltx@replace@once{`⟨*cs*⟩`}{`⟨*search*⟩`}{`⟨*replace*⟩`}`
Replaces the first occurence of ⟨*search*⟩ in the first expansion of ⟨*cs*⟩ with ⟨*replace*⟩.

`\cnltx@replace@all{`⟨*cs*⟩`}{`⟨*search*⟩`}{`⟨*replace*⟩`}`
Replaces all occurences of ⟨*search*⟩ in the first expansion of ⟨*cs*⟩ with ⟨*replace*⟩.

`\cnltx@define@colorscheme{`⟨*name*⟩`}{`⟨*scheme definition*⟩`}`
Command that can be used to define a color scheme.

10.2 Defined by CNLTX-EXAMPLE

`\cnltx@example@error{`⟨*message*⟩`}`
Issue an error message using `\PackageError`.

`\cnltx@example@warning{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarning`.

`\cnltx@example@warningnoline{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarningNoLine`.

`\cnltx@example@info{`⟨*message*⟩`}`
Issue a message using `\PackageInfo`.

`\at`
Robust command that typesets '@'. An @ in command names confuses the indexing of the command names. Either one uses another symbol for makeindex's "actual" recognition and also tells idxcmds about it or one uses `\at` in `\cs` and friends.

`\bang`
Robust command that typesets '!'.

`\equal`
Robust command that typesets '='.

`\cnltx@equal`
Used in definitions of the key/value option typesetting commands.

`\indexcs`
Version of `\csidx` that takes care of a `\textcompwordmark` inserted by listings. Also replaces all occurences of @ with category code 11 or 12 with `\at`. Used to index commands in the `sourcecode` and `example` environments that have been added with `add-cmds`.

`\newarg{`⟨*cs*⟩`}{`⟨*left delim*⟩`}{`⟨*right delim*⟩`}`
Command used to define the argument commands: `\newarg\marg{\{}{\}}`

`\MakePercentComment`
Sets the category code of % to 14.

`\cnltx@copyablespace`
Prints a space that is also copyable. Uses the accsupp.

`\cnltx@mdframed@options`
Predefined option list for the mdframed style `cnltx`.

`\cnltx@listings@style`
Predefined option list for the listings style `cnltx`.

10.3 Defined by CNLTX-DOC

`\cnltx@doc@error{`⟨*message*⟩`}`
Issue an error message using `\PackageError`.

`\cnltx@doc@warning{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarning`.

`\cnltx@doc@warningnoline{`⟨*message*⟩`}`
Issue a warning message using `\PackageWarningNoLine`.

`\cnltx@doc@info{`⟨*message*⟩`}`
Issue a message using `\PackageInfo`.

`\cnltx@getfileinfo{`⟨*file name*⟩`}{`⟨*file extension*⟩`}`
Extract the date, version and background information for a package or a class.

`\cnltx@newname{`⟨*cs*⟩`}{`⟨*first name*⟩`}`last name
Defines ⟨*cs*⟩ to write out the name and add an index entry. Also defines a starred variant that only writes the last name but still adds an index entry.

`\cnltx@version@note{`⟨*note*⟩`}`
Command that is used for the versioning notes interally. Sets `\reversemarginpar` and then writes the note ⟨*note*⟩ to the margin with corresponding formatting.

`\begin{cnltxlist}`
The list environment that is used by the environments `commands`, `options` and `environments`.

10.4 Defined by CNLTX-CSNAMES

`\cnltx@predefined@control@sequences`
A comma-separated list of predefind ‘silent’ control sequence names.

Index