

# The cocotex.dtx Package

**A modular package suite for  
automatic, flexible typesetting**

Version 0.5.0

(2024/07/16)

Lupino

[lupino@le-tex.de](mailto:lupino@le-tex.de)

# Table of contents

---

<b>Introduction</b>	<b>vii</b>
1 Basic concepts	vii
1.1 Types, Inheritance and Abstract Containers	vii
1.2 Complex Components	vii
1.3 Relation to L <sup>A</sup> T <sub>E</sub> X Templates	viii
2 How to Read This Documentation	viii
2.1 Keyword Markings	viii
2.2 Data Types of Properties	ix
2.3 Types of Components	ix
 <b>Module 1 cocotex.dtx</b>	 <b>3</b>
1 Hard-coded requirements	3
2 Class Options	3
2.1 Options passed down to mandatory standard L <sup>A</sup> T <sub>E</sub> X packages	3
2.2 The Publication Type	4
2.3 User-Level Macro Names and Debugging Options	4
2.4 Accessibility Features	5
2.5 Options for Other CoCoT <sub>E</sub> X Modules	5
3 Class Hook	6
4 Internal Requirement	6
5 Loading and Adjusting Underlying DocumentClass	7
5.1 General Typography	7
6 Loading other CoCoT <sub>E</sub> X Modules	8
6.1 coco-accessibility	8
6.2 coco-script	8
6.3 coco-headings	8
6.4 coco-floats	8
6.5 coco-title	9
6.6 coco-notes	9
7 Further Hard Dependencies	9
7.1 Index	9
7.2 Hyperref	9
8 End of Document Class Hook	10
 <b>Module 2 coco-kernel.dtx</b>	 <b>11</b>
1 Preamble	11
1.1 Hard dependencies	11
1.2 Package Options	11
2 Exception handlers	12
3 Global Switches	13
4 Containers	14
5 Components	17
5.1 Simple Components	17
5.2 Counted Components	20
6 Properties	26
6.1 Setting Properties	26
6.2 Using Properties	27

6.3	Processing Instructions .....	29
6.4	Property Conditionals.....	29
7	Helper macros .....	29
7.1	Handling of Optional Arguments .....	29
7.2	Iterators .....	29
7.3	Attributes .....	30
7.4	Style Classes .....	33
7.5	The CoCoTeX Logo .....	34
<b>Module 3 coco-common.dtx</b>		<b>35</b>
1	Package options .....	35
1.1	Accessibility Features.....	35
2	Commonly Used Low-Level Macros and Registers .....	36
2.1	Hard Dependencies .....	36
2.2	Common Variables.....	36
2.3	Helper macros .....	37
2.4	Masks .....	37
2.5	Arithmetics .....	38
2.6	Determine actual page number .....	39
3	Re-Thinking L <sup>A</sup> T <sub>E</sub> X Core Functions .....	40
3.1	Keeping .aux-Files Up-to-Date .....	40
3.2	Content lists .....	41
3.3	Indentation and Left Margins of Potentially Numbered Items .....	43
3.4	Labelling and Cross referencing .....	47
3.5	Linguistic Name generation and selection .....	48
3.6	Link Generation .....	49
<b>Module 4 coco-accessibility.dtx</b>		<b>51</b>
1	LaTeX code .....	51
1.1	General Processing.....	51
1.2	Activating and Deactivating Accessibility Features .....	52
1.3	Accessibility-specific additions .....	53
1.4	Generic Macro to Declare Accessibility Features .....	54
1.5	Lua injection .....	58
1.6	Hyperlink handling .....	58
1.7	Tagging Page Styles as Artifacts .....	60
1.8	generic artifacts .....	61
1.9	Tagging for Floats.....	61
1.10	Transformation of Typographic Unicode characters .....	62
1.11	Automatic PDF Tagging .....	63
1.12	Math Tagging .....	63
1.13	Default Role Mapping .....	64
2	Lua code .....	64
2.1	Local Variables and Tables .....	64
2.2	Meta Data Extraction .....	64
2.3	Public Methods.....	66
<b>Module 5 coco-meta.dtx</b>		<b>67</b>
1	Counted Container Handlers.....	67
1.1	Generic Blocks .....	67
1.2	Contributor Roles .....	68
1.3	Declaring Contributor Role Blocks .....	68
2	Labeled Components .....	72
3	Meta Data Rolemaps for Tagged PDFs.....	73
4	Common Meta Data.....	73
4.1	Affiliations.....	75



3.7	Support for Greek script .....	119
3.8	Support for Ethiopian/Amharic script .....	119
3.9	Support for Syrian script .....	120

## **Module 9 coco-title.dtx 123**

1	Top-Level Interface.....	123
2	Processing of PDF Meta Data .....	125
2.1	Processing of the Document's Title .....	126
2.2	Processing of the Document's Author .....	127
2.3	Processing of the PDF's Creator, Producer, and Keywords Meta Data.....	127
2.4	Including the XMP Meta Data .....	128
3	Intermediate Level Interfaces .....	128
3.1	Funds, Grants, and Supporters .....	129
3.2	Simple Component Declarations .....	130
4	Default Settings .....	137
5	Accessibility Features .....	142
5.1	Output Intent and ICC Profiles .....	142
5.2	Encoding of the PDF-A Conformance .....	143
5.3	Titlepage Specific Role Maps .....	144

## **Module 10 coco-floats.dtx 145**

1	Package Setup .....	145
1.1	Hard requirements .....	145
1.2	Adjustments at the Beginning of the Document .....	146
1.3	Document Class-Option Overrides .....	146
1.4	Internal Registers .....	147
2	Internal macros .....	148
2.1	Generic resetter .....	148
2.2	Wrapper for L <sup>A</sup> T <sub>E</sub> X's Native float Environments .....	149
3	The Generic float Container .....	149
3.1	Common Float Components.....	150
3.2	Common Float Properties .....	151
3.3	The Generic float Environment .....	156
3.4	The SubFloat Environment .....	158
3.5	Attribute Handlers .....	159
3.6	Handling of List-of Entries .....	160
3.7	Label and Referencing mechanisms.....	162
3.8	Processing the Float.....	163
3.9	Caption mechanism .....	165
4	Generic User-Level Float Containers.....	168
5	Image Containers .....	169
5.1	Abstract Graphics Container .....	169
5.2	Floating Figure Container .....	169
5.3	Figure Output Routines .....	170
5.4	Inline Figures.....	171
6	Table Containers .....	172
6.1	The Abstract Tabular Container .....	172
6.2	The User-Level Table Container .....	173
6.3	The Table Output Handler .....	174
7	Other Float-Related Macros .....	174

## **Module 11 coco-frame.dtx 177**

1	Top-Level Interface.....	177
2	Cropmark printer .....	177

## **Module 12 coco-lists.dtx 181**

1	Preamble .....	181
1.1	Package Options .....	181
2	The List Container .....	182
2.1	List Properties .....	182
2.2	List Components .....	185
3	Declaring List Types .....	186
4	Declare Lists .....	186
4.1	The List Environment .....	189
4.2	Unpacking the List Type-Specific Handlers .....	191
5	Default List Types .....	192
5.1	Unnumbered Lists .....	192
5.2	Numbered Lists .....	193
5.3	Description Lists .....	194
5.4	Replacing L <sup>A</sup> T <sub>E</sub> X's Default Lists .....	196

## **Index** **197**

Macro and Environment Index .....	197
Container Index .....	202
Component Index .....	203
Property Index .....	205
Hook Index .....	208
Tag Index .....	209
Attribute Index .....	210

# Introduction

---

## 1 Basic concepts

The core concept of the CoCoTeX Framework to view typographical objects, such as *floats*, *headings*, *title pages*, etc., as closed units that contain a fixed set of elements that determine the exact nature of each occurrence. For a *heading*, such elements may be the heading’s *title*, an optional *subtitle*, a *counter* or a list of *authors* responsible for the part of a publication introduced by the *heading*.

In CoCoTeX those typographical units are referred to as *Containers*. The occurrence of a Container in a specific TeX document is an *Instance* of that Container. The elements inside each Container instance are called *Components*.

The final realization of a *Container* in the rendered output is done in local style files with so-called *Properties*; short snippets of L<sup>A</sup>T<sub>E</sub>X code, which tell the L<sup>A</sup>T<sub>E</sub>X interpreter how the Components in the Instances of Containers are to be read, processed and eventually rendered.

Typically, Containers are L<sup>A</sup>T<sub>E</sub>X environments that contain the Components in the form of L<sup>A</sup>T<sub>E</sub>X macros or other, embedded, environments. In the simpler cases, Component macros take the value for the Component in that specific *Instance* of the Container as their mandatory argument. Most Containers follow an *read first – process later* approach, i.e., the L<sup>A</sup>T<sub>E</sub>X interpreter reads the whole content of the Container and the processing is done at the `\end` macro of the corresponding environment.

### 1.1 Types, Inheritance and Abstract Containers

Components and Properties are both seen as (*Data*-)*Types* specific to each Container. A Container can be abstract, meaning that the Container is by itself not directly used in an end-user’s tex file, but serves as “blueprint” for other, more “user-level”, Containers. As such, Containers can *inherit* the Types of another Container. Containers that inherit Types from other Containers are called *Sub Containers* or *Child Containers*, while the inherited Container is called a *Parent Container*.

Containers are therefore somewhat comparable to *classes* in object-oriented programming languages, an Instance of a Container can be seen as an *object* (i. e., an *instance of a class*). Components are *object variables*, while Properties take the place of *class variables* and/or *methods*, depending on how exactly a certain Property is implemented. Sometimes, a Property holds only a simple value (which makes it a *class variable*), while another Property may contain a complex set of instructions and calls to other Properties and Component values (which would make it a *method*).

### 1.2 Complex Components

Components can also be more complex than simple data storage devices. Usually, a Component occurs only once in a Container, for instance, there can be only one (main) “Title” in each “Heading”.

Other Components may occur more than once in the same Container Instance, for example, a “chapter” (which itself may be a Sub Container of a more abstract Parent Container “Heading”) may have more than one “Author”. Such Components are called *Group Components*.

They are usually realized as L<sup>A</sup>T<sub>E</sub>X environments within a Container’s environment and contain themselves other Components. Those “second-level” Components are called *Counted Components*, as they are “enumerated” across all Group Component instances within the same Container Instance. For each Group Component, there is a *Collection Component*, in which all instances of a Group Component are collected during processing. How this collection is put together is controlled by a special *Collection Property*.

### 1.3 Relation to L<sup>A</sup>T<sub>E</sub>X Templates

Newer version of L<sup>A</sup>T<sub>E</sub>X adopt a quite similar design principle with the introduction of *Templates* into the L<sup>A</sup>T<sub>E</sub>X Kernel in mid 2024<sup>1</sup>

The template system in L<sup>A</sup>T<sub>E</sub>X provides three levels of abstraction: *Object Types*, *Templates* and *Template Instances*. An *Object Type* represents the general idea of a typographical element, like “heading”, “float”, or “list”. The *Object Type* also determines the exact number of “Arguments” each *Template Instance* must or may have. *Templates* define how the instances may be manipulated by the end-user by adding a list of pre-defined key-value pairs. Finally, *Template Instances* are what the end-user is supposed to be using in their documents, often masked behind more user-friendly interface macros.

As an example, an *Object Type* may be “sectioning” that provides the Arguments title, short title and number. A *Template* heading derived from that *Object Type* may introduce the Interface key-value pairs pre-skip, font-size and after-skip. A *Template Instance* then might be *subsection*, which defines the pre-skip to `2\baselineskip`, the after-skip to `1\baselineskip`

and the font-size to `{10}{12}`. The *Template Instance* is then being called in the definition of a user level macro like `\subsection[<label>][<shorttitle>]{<title>}`.

In CoCoT<sub>E</sub>X terms, an *Object Type* would be an abstract Container that defines the exact number and nature of Components each Instance of that Container could have, but does not declare any Properties. L<sup>A</sup>T<sub>E</sub>X *Templates* are equivalent to CoCoT<sub>E</sub>X’s second-level Child Containers whose parent is the abstract *Object-Type* Container, but it only declares the Properties that can be used to manipulate the output of each Container Instance. *Template Instances* are equivalent to third-level Sub Containers of the *Template* Child Container, that define fixed values for some or all of the *Properties*.

In summary, CoCoT<sub>E</sub>X has no formal distinction between *Object Types*, *Templates* and *Template Instances*. However, those “layers of abstraction” can be realized by the *Inheritance* mechanism, but there is no hard restrictions about *if* and *when* *Properties* and/or *Components* are introduced.

## 2 How to Read This Documentation

The documented source code is printed in red code boxes with line numbers referring to lines in the corresponding unpacked .sty files:

29 This is the documented source code

Code and usage examples are printed in blue boxes:

This is a {L<sup>A</sup>T<sub>E</sub>X} example.

### 2.1 Keyword Markings

Certain Parts of this documentation are icon- and color-coded:

- 📦 **Containers** are orange and marked with a box symbol 📦,
- 📄 **Hooks** are green and marked with an insertion icon 📄,
- ➡ **Components** are blue and marked with an arrow to box symbol ➡,
- ⚙ **Properties** are purple and marked with a gear symbol ⚙,
- 🏷 **PDF-Tags** are cyan and marked with a tag symbol 🏷,
- 🔗 **Attributes** are dark green and marked with a chain symbol 🔗, and
- 📄 **L<sup>A</sup>T<sub>E</sub>X-Macros** are red and have no symbol.

<sup>1</sup>It is however noteworthy that the principal functionality has been available for much longer in form of the `xtemplate` package.



## 2.2 Data Types of Properties

Whenever a Property is declared, the documentation contains a list of expected values for that property. The following list gives an overview over the various expected data types:

<code>&lt;dimen&gt;</code>	means that the Property is expected to return a dimensional value (or “length”) or a dimension register.
<code>&lt;skip&gt;</code>	means that the Property is expected to return a skip, i. e., a $\text{\LaTeX}$ dimension with or without glue, or a skip register.
<code>&lt;num&gt;</code>	means that the Property is expected to return a number or counter register.
<code>&lt;CS token&gt;</code>	means that one previously defined control sequence token (i. e., a $\text{\LaTeX}$ macro) is expected.
<code>[word1 word2]</code>	indicates that either exact word1 or word2 is expected. This notation may also contain other fixed data types, and more than one option could be given.
<code>&lt;name&gt;</code>	means that the name of a specific Component, Property or Container is expected. Details are usually in the description.
<code>&lt;any&gt;</code>	means that the Property can take any value.

## 2.3 Types of Components

LC	means that the Component is a Labeled Component
CC	means that the Component is a Counted Component
GC	means that the Component is a Group Component
CL	means that the Component is a Collection Component
OR	means that the Component is an Override



One more driver function

22 `<*driver>`

If we want to run the splitted development dtx locally, this macro prevents undefined control sequence errors and actually includes the dtx chunks.

23 `\def\includeDTX#1{\input src/#1.dtx}`

End driver function

24 `</driver>`



# Module 1

## cocotex.dtx

---

```
<*class>
```

This is the main class file for the CoCoT<sub>E</sub>X Framework.

File Preamble

```
23 %%
24 %% Common document class for \textit{xerif} projects.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesClass{cocotex}
32 [2024/07/16 0.5.0 cocotex]
```

## 1 Hard-coded requirements

```
33 \RequirePackage{xkeyval}
```

First, we set the default hook label for all CoCoT<sub>E</sub>X modules. Since some modules are “stand-alone”, we do this in all kernel files, i.e., here, in `coco-kernel` and in `coco-common`.

```
34 \SetDefaultHookLabel{cc}
```

## 2 Class Options

### 2.1 Options passed down to mandatory standard L<sup>A</sup>T<sub>E</sub>X packages

The main option controls the document’s main language passed down to the `babel` package.

```
35 \DeclareOptionX{main}{\PassOptionsToPackage{\CurrentOption}{babel}}
```

The next two options are used for Spanish documents:

```
36 \DeclareOption{es-noindentfirst}{\PassOptionsToPackage{es-noindentfirst}{babel}}
37 \DeclareOption{es-noshorthands}{\PassOptionsToPackage{es-noshorthands}{babel}}
```

By default, we disable babel shorthands as its character encoding may interfere with some CoCoT<sub>E</sub>X functionality.

```
38 \PassOptionsToPackage{shorthands=off}{babel}
```

The `no-hyperindex` switch prevents `hyperref` from auto-linking index terms.

```
39 \DeclareOption{no-hyperindex}{\global\let\cc@no@hyperindex\relax}
```

## 2.2 The Publication Type

The option `pubtype` (short for “publication type”) has four possible values: `mono`, `collection`, `journal`, and `article`. `mono` (also the default when no `pubtype` is given) and `collection` are used to switch between single and multiple contributor documents; `collection` and `journal` to switch between one-time text collections and periodicals, respectively. All three types implicitly load the L<sup>A</sup>T<sub>E</sub>X standard class `book`.

`collection` is used when the document’s components (i. e., chapters) are contributed by different authors like collections or proceedings. `journal` is used for collections where each contribution is accompanied by a myriad of meta data. `mono` stands for monographs, i. e., whole books that are written by the same author(s).

The publication type `article` is intended for single articles of a journal. It loads the L<sup>A</sup>T<sub>E</sub>X standard class `article`.

```
40 \newif\ifcollection \collectionfalse
41 \newif\ifarticle \articlefalse
42 \newif\ifmonograph \monographfalse
43 \newif\ifjournal \journalfalse
44 \define@choicekey{cocotex.cls}{pubtype}[\cc@pub@type\nr]{collection,article,journal,mono}{%
45   \ifcase\nr\relax% collection
46     \global\collectiontrue
47   \or% article
48     \global\articletrue
49   \or% journal
50     \global\journaltrue
51   \else% monograph
52     \global\monographtrue
53   \fi
54 }
```

## 2.3 User-Level Macro Names and Debugging Options

Next, we capture all options that needs passing down to the various CoCoT<sub>E</sub>X modules.

The prefix option is used to define prefix for user level macros. If CoCoT<sub>E</sub>X is used in conjunction with *xerif*, this will be `tp`.

```
55 \DeclareOptionX{prefix}{\PassOptionsToPackage{\CurrentOption}{coco-kernel}}
```

The debugging options: trigger `debug` will toggle on debug mode, valued `debug-domain` will determine the kinds of messages printed to the console.

```
56 \DeclareOptionX{debug}{\PassOptionsToPackage{\CurrentOption}{coco-kernel}}
57 \DeclareOptionX{debug-domain}{\PassOptionsToPackage{\CurrentOption}{coco-kernel}}
```

The `nofigs` options disables the `\includegraphics` command and prints a placeholder instead. This is intended to ensure successful L<sup>A</sup>T<sub>E</sub>X runs even thou image files may be missing.

```
58 \DeclareOptionX{nofigs}{\PassOptionsToPackage{\CurrentOption}{coco-floats}}
```

## 2.4 Accessibility Features

The next two options enable accessibility features and control the PDF standard used: `a11y` generates PDF version 1.7, `a11y20` generates PDF version 2.0.

```

59 \ExplSyntaxOn
60   \DeclareOptionX{a11y}{%
61     \sys_ensure_backend:
62     \pdf_version_gset:n{1.7}
63     \PassOptionsToPackage{init}{coco-accessibility}}
64   \DeclareOptionX{a11y20}{%
65     \sys_ensure_backend:
66     \pdf_version_gset:n{2.0}
67     \PassOptionsToPackage{init}{coco-accessibility}}
68 \ExplSyntaxOff

```

`lang-id` is the ISO-639/2 identifier of the document's main language. This is necessary for the PDF meta data and *different* from the main language name given via the `main` key.

```

69 \DeclareOptionX{lang-id}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}

```

If set, `nodetree` triggers extensive debugging output from the `ltpdfa` package.

```

70 \DeclareOptionX{nodetree}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}

```

`showspaces` enables whitespaces processed by `ltpdfa` to be visible in the PDF.

```

71 \DeclareOptionX{showspaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}

```

`no-spaces` disables whitespace processing by `ltpdfa`.

```

72 \DeclareOptionX{no-spaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}

```

`no-paras` disables paragraph tagging via `ltpdfa`.

```

73 \DeclareOptionX{no-paras}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}

```

`no-compress` disables PDF compression; useful for debugging the PDF source code.

```

74 \DeclareOptionX{no-compress}{\let\cc@no@pdf@compression\relax}

```

`color-enc` serves two purposes: First, it controls the colour space for colours invoked via the `xcolor` package. Second, it controls which default ICC colour profile is embedded into the PDF file when no explicit ICC profile is provided by the user.

```

75 \DeclareOptionX{color-enc}{\PassOptionsToPackage{\CurrentOption}{coco-common}}

```

## 2.5 Options for Other CoCoTeX Modules

### Options for the Script Module

The option `usescript` takes a comma-separated list of language names. Languages that use non-latin scripts may require fallback fonts when the script's glyphs are not included in the main font. This option tells the `coco-scripts` module which fonts to pre-load. The values in is also passed down to `\babelprovide`.

```

76 \DeclareOptionX{usescript}{\PassOptionsToPackage{\CurrentOption}{coco-script}}

```

### Options for the Notes Module

The switch `endnotes` triggers all footnotes to be collected and printed in a specific endnote section at the end of the document.

```
77 \DeclareOptionX{endnotes}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
```

The switch `ennotoc` triggers headings in the Endnotes area to *not* appear in the table of contents (read: *end-notes-no-toc*).

```
78 \DeclareOptionX{ennotoc}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
```

The switch `endnoteswithchapters` triggers chapter headings to be repeated as subsections within the endnote section.

```
79 \DeclareOptionX{endnoteswithchapters}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
```

The switch `resetnotesperchapter` causes foot- and endnote counters to be reset at the beginning of each new chapter.

```
80 \DeclareOptionX{resetnotesperchapter}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
```

### Remaining Options

All other unprocessed options are passed down to the base document class:

```
81 \DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{article}}
82 \DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{book}}
```

Process the options.

```
83 \ProcessOptionsX
```

## 3 Class Hook

`\ccAfterClassHook` Almost all user level macros have been renamed when CoCoT<sub>E</sub>X became independent from `xerif`. In order to ensure backwards-compatibility, we define a hook that holds aliases from the old names to the new ones. Those are defined in the `coco-xerif` module (which is *not* part of CoCoT<sub>E</sub>X itself, but included in `xerif`'s common files<sup>1</sup>). The hook is expanded at the very end of the `cocotex.cls` file. The `coco-xerif` module itself is loaded early in `coco-common.sty`.

```
84 \def\ccAfterClassHook{}
```

`\ccToggleCountedConditionalsHook` is a hook to ensure backwards-compatibility within the processing of Counted Components

```
85 \def\ccToggleCountedConditionalsHook{}
```

## 4 Internal Requirement

```
86 \RequirePackage{coco-common}
```

<sup>1</sup>see <https://github.com/transpect/xerif-latex>



## 5 Loading and Adjusting Underlying DocumentClass

All publication types supported by CoCoT<sub>E</sub>X are based on one of L<sub>A</sub>T<sub>E</sub>X's default classes `article` (when `pubtype=article`) or `book` (all other pubtypes):

```
87 \ifarticle
88   \LoadClass[10pt,a4paper]{article}
89 \else
90   \LoadClass[10pt,a4paper]{book}
91 \fi
```

### 5.1 General Typography

Offsets are the removed to make all values relative to the upper left corner of the page to ease maintainance.

```
92 \voffset-1in\relax
93 \hoffset-1in\relax
```

Automatted typesetting needs some room to play

```
94 \emergencystretch=2em
```

and strong restrictions:

```
95 \frenchspacing
96 \clubpenalty10000
97 \widowpenalty10000
```

#### Empty Pagestyle

Page style without any headers or footers

```
98 \def\ps@empty{%
99   \let\@oddhead\@empty
100   \let\@evenhead\@empty
101   \let\@oddfoot\@empty
102   \let\@evenfoot\@empty
103 }
```

#### Vacancy Pages

Vacancy pages in general need to have page style `empty`:

```
104 \def\cleardoublepage{\clearpage\if@twoside \ifodd\c@page\else
105   \hbox{}\thispagestyle{empty}\newpage\if@twocolumn\hbox{}\newpage\fi\fi}
```

#### Book Parts

re-defined to make front- and backmatter components distinguish-able

```
106 \ifarticle\else
107   \newif\if@frontmatter \@frontmatterfalse
108   \renewcommand\frontmatter{%
109     \cleardoublepage
110     \@mainmatterfalse
111     \@frontmattertrue
```

```

112 \pagenumbering{arabic}}
113 \renewcommand\mainmatter{%
114 \cleardoublepage
115 \@frontmatterfalse
116 \@mainmattertrue}
117 \renewcommand\backmatter{%
118 \cleardoublepage
119 \@mainmatterfalse
120 \@frontmatterfalse}
121 \fi

```

**WARNING!**  
The following section is deprecated and will be changed or deleted in future releases.

```

122 \usepackage{soul}

```

## 6 Loading other CoCoT<sub>E</sub>X Modules

### 6.1 coco-accessibility

We load the accessibility module always, even if we don't end up actually using it.

```

123 \RequirePackage{coco-accessibility}

```

### 6.2 coco-script

Inclusion of the script module which also loads the babel package

```

124 \ifLuaTeX
125 \RequirePackage{coco-script}
126 \else
127 \RequirePackage{babel}
128 \fi

```

### 6.3 coco-headings

```

129 \RequirePackage{coco-headings}

```

### 6.4 coco-floats

Inclusion of the float module

```

130 \RequirePackage{coco-floats}

```

## 6.5 coco-title

Inclusion of the title page module

```
131 \RequirePackage{coco-title}
```

## 6.6 coco-notes

Inclusion of the end-/footnotes module

```
132 \RequirePackage{coco-notes}
```

Fallback, in case, `coco-headings.sty` is not loaded for some reason.

# 7 Further Hard Dependencies

## 7.1 Index

Some more hard dependencies:

```
133 \RequirePackage{index}
134 \makeindex
```

## 7.2 Hyperref

```
135 \RequirePackage{hyperref}
```

Finally, some `hyperref` settings

**TODO**  
check, which of those  
are better placed inside  
the local publisher's  
styles

```
136 \hypersetup{%
```

first, we want links to be breakable

```
137 breaklinks%
```

and the table of contents not to be automatically linked, as this causes problems with the `ltpdfa` package and we add the links via the `coco-common` module, anyways.

```
138 ,linktoc=none%
```

pdf borders are controlled via the `coco-frame` module, if necessary

```
139 ,pdfborder={0 0 0}%
```

The next option causes `hyperref` to calculate the encoding of `DocumentInfo` and other direct-to-PDF data (bookmarks, etc.) automatically

```

140 ,pdfencoding=unicode
141 ,unicode=true

```

Bookmarks are numbered by default.

```

142 ,bookmarksnumbered=true%
143 ,bookmarksopen=false%
144 ,hyperindex=\ifx\cc@no@hyperindex\relax false\else true\fi
145 }

```

Disables PDF compression when the `no-compress` document option is set.

```

146 \ifx\cc@no@pdf@compression\relax
147 \ifx\pdfobjcompresslevel\@undefined
148 \edef\pdfobjcompresslevel{\pdfvariable objcompresslevel}%
149 \fi
150 \pdfcompresslevel=0
151 \pdfobjcompresslevel=0
152 \fi

```

## 8 End of Document Class Hook

Expanding backwards-compatibility aliases from the `coco-xerif` module:

```

153 \ccAfterClassHook

```

```

</class>

```

## Module 2

# coco-kernel.dtx

---

```
<*kernel>
```

This file provides the object-oriented interfaces for all other CoCoTeX modules.

## 1 Preamble

```
23 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
24 \ProvidesPackage{coco-kernel}
25 [2024/07/16 0.5.0 cocotex kernel]
```

Before we do anything, we check if the user uses a (more or less) current  $\text{\LaTeX}$  kernel version. If not, we issue a hard error message. The pivot that separates “new” from “old” is June 1, 2020.

```
26 \ifx\IfFormatAtLeastTF\undefined
27 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}%
28 \fi
29 \IfFormatAtLeastTF{2020/06/01}{}%
30 {\PackageError{CoCoTeX Kernel}
31 {LaTeX kernel too old!}
32 {CoCoTeX v0.5.0 and newer needs at least LaTeX kernel version 2020/06/01!}}
```

### 1.1 Hard dependencies

```
33 \RequirePackage{kvoptions-patch}
34 \RequirePackage{xkeyval}
35 \RequirePackage{etoolbox}
```

Default hook label for CoCoTeX modules:

```
36 \SetDefaultHookLabel{cc}
```

### 1.2 Package Options

The `debug` option triggers the output of additional information messages to the shell.

```
37 \newif\ifcc@debug \cc@debugfalse
38 \DeclareOptionX{debug}{\global\cc@debugtrue}
```

The `debug-domain` option serves as a filter for log messages. It takes a comma separated list of log message categories as argument. Only messages whose domain match any item of that list are printed. If the option is omitted or its argument empty, all messages are printed.

Implies `debug`.

```

39 \global\let\debug@domain@list\relax
40 \DeclareOptionX{debug-domain}{%
41   \global\@cc@debugtrue
42   \if!#1!\else
43     \def\do##1{\listadd\debug@domain@list{##1}}%
44     \docsvlist{#1}%
45   \fi
46 }%
```

The `prefix` option will be explained below in Sect. 3.

```

47 \let\cc@prefix\@empty
48 \DeclareOptionX{prefix}[]{\gdef\cc@prefix{#1}}%
49 \ProcessOptionsX
```

## 2 Exception handlers

The CoCoTeX kernel provides some macros to unify exception handling. There are four levels of output: `error`, `warning`, `info`, and `debug`.

`\ccPackageError` creates an error message specific to the Framework.

`{#1}` is the module  
`{#2}` is the type of error  
`{#3}` is the immediate error message  
`{#4}` is the help string

```

50 \def\ccPackageError#1#2#3#4{%
51   \GenericError{%
52     (#1)\@spaces\@spaces\@spaces\@spaces
53   }{%
54     [CoCoTeX #1 #2 Error] #3%
55   }-#4}%
56 }
```

`\ccPackageWarning` is a macro to create warnings specific to the Framework.

`{#1}` is the module  
`{#2}` is the type of error  
`{#3}` is the immediate warning message

```

57 \def\ccPackageWarning#1#2#3{%
58   \GenericWarning{%
59     (#1)\@spaces\@spaces\@spaces\@spaces
60   }{%
61     [CoCoTeX #1 \if!#2!\else#2 \fi Warning] #3%
62   }%
63 }
```

`\ccPackageInfo` is a macro to create shell output specific to the Framework.

`{#1}` is the module  
`{#2}` is the type of message  
`{#3}` is the immediate info string

```

64 \def\ccPackageInfo#1#2#3{%
65   \GenericInfo{%
66     (#1)\@spaces\@spaces\@spaces\@spaces
67   }{%
68     [CoCoTeX #1\if!#2!\else\space#2\fi] #3%
69   }%
70 }

```

While the macros defined above are meant to be used in all CoCoTeX modules, the following is only for the Kernel.

`\cc@debug@message` is a generic debug message that is displayed whenever the debug option is set. Otherwise, it gobbles its argument.

```

71 \ifcc@debug
72   \def\do#1{\csgdef{cc@debug@#1@message}##1{\typeout{[CoCo #1 Debug]\space##1}}}%
73   \dolistloop{\debug@domain@list}%
74   \def\cc@debug@message#1{\typeout{[CoCo Debug]\space#1}}
75 \else
76   \let\cc@debug@message\gobble
77 \fi

```

`\ccDebugMsg` prints a domain specific debug message.

`[#1]` is the debug domain for filtering  
`{#2}` is the message

The whole mechanism works by *dynamicly* defining the underlying, domain specific, debug message macros, which all follow the scheme `cc@debug@<domain>@message`. If the debug domain is requested via the debug-domains class option, the macros whose domain is listed in this key are defined on runtime. Otherways, the mandatory argument gets gobbled.

The following debug domain filters are currently used:

a11y Messages related to PDF tagging and accessibility features  
 eval final values of Type expansions  
 inheritance Inheritance mechanism of Types

```

78 \def\ccDebugMsg{\cc@opt@empty\cc@debug@msg}
79 \def\cc@debug@msg[#1]#2{%
80   \expandafter\ifx\csname cc@debug@#1@message\endcsname\relax
81     \gobble{#2}%
82   \else
83     \csname cc@debug@#1@message\endcsname{#2}%
84   \fi
85 }

```

### 3 Global Switches

`\ccPrefix` is the prefix that is added to Component macros and (some) Container environments.

This has mostly historic reasons: back when CoCoTeX was specific to the *xerif* typesetting automaton, all macros produced by the xml converter had a `tp` prefix (from *transp*ect, the XML conversion tool in the backend of *xerif*). After CoCoTeX became stand-alone, the `tp` prefix became obsolete, but the converters running at the time needed to be backward-compatible. Therefore, all *xerif*-bound CoCoTeX instances still set this macro to ensure user-level macros bear the `tp`-prefix.





`\ccDeclareType` Each Container is defined by the data types it provides. These data types are declared with this macro.

`{#1}` is the name of the data type

`{#2}` is code that is specific to this type, usually something like Component or Property declarations, handlers, and so forth

```
102 \long\def\ccDeclareType##1##2{\csgappto{cc@type@##1@#1}{##2}\ignorespaces}%
```

`\ccDeclareEnv` Each container usually is realised as a L<sup>A</sup>T<sub>E</sub>X environment. The `\ccDeclareEnv` macro is used to set up this environment. Usually, the environment has *the same name as the Container*.

`[#1]` overrides the environment's name. However, keep in mind that the Container's name is not changed by re-naming the corresponding environment.

`{#2}` is used for the stuff done at the environment's beginning

`{#3}` is the stuff done at the environment's end

In the `begin` part, the Types declared in the Container declaration's body should be evaluated using the `\ccEvalType` macro, see below.

```
103 \def\ccDeclareEnv{\@ifnextchar [{\cc@declare@env}{\cc@declare@env[#1]}}%
104 \def\cc@declare@env[#1]##2##3{%
105   \csgdef{\ccPrefix ##1}{\global\let\reserved@cont\cc@cur@cont\def\cc@cur@cont{#1}##2}%
106   \csgdef{end\ccPrefix ##1}{##3}\global\let\cc@cur@cont\reserved@cont}%
```

The body of the Container is expanded last and should make use of the macros defined above.

```
107 \ccDeclareType{Attributes}{}%
108 #2%
109 \endgroup
110 \ignorespaces}
```

`\ccSetContainer` is used to change the currently active (Sub-)Container.

`{#1}` is the name of the new active Container

```
111 \def\ccSetContainer#1{\def\cc@cur@cont{#1}\ignorespaces}
```

`\ccAddToType` add additional content (i.e., the next token) to a Type `{#1}` of a previously declared Container `{#2}`.

```
112 \def\ccAddToType#1#2{\csgappto{cc@type@#1@#2}}
```

`\ccEvalType` calls the declaration list for Data Type `{#2}`. With optional `[#1]`, the Type's Container name can be overridden locally.

```
113 \def\ccEvalType{\cc@opt@curcont\cc@eval@type}
114 \def\cc@eval@type[#1]#2{%
115   \expandafter\ifx\csname cc@type@#2@#1\endcsname\relax
116     \ccPackageError{Kernel}{Class}
117     {Data Type #2 in Container #1 undefined!}
118     {You try to evaluate a data type `#2' from container `#1', but that data type has not been
      declared.}%
119   \else
120     \ccDebugMsg[eval]{Evaluating cc@type@#2@#1:^^J \csmeaning{cc@type@#2@#1}}%
121     \csname cc@type@#2@#1\endcsname
122   \fi\ignorespaces}
```

`\ccCheckParent` checks if a Container `{#1}` is declared so that another Container `{#2}` can inherit.

```

123 \def\ccCheckParent#1#2{%
124   \expandafter\ifx\curname cc@container@#1\endcsname\relax
125   \ccPackageError{Kernel}{Class}
126   {Parent Container `#1' undeclared}
127   {You tried to make a Container named `#2' inherit from a Container named `#1', but a
128     Container with that name does not exist.\MessageBreak
129     Please make sure that parent Containers are declared before their descendents.}%
129   \else
130     \csgdef{cc@parent@#2}{#1}%
131   \fi
132   \ignorespaces}

```

`\cc@inherit` is the low-level inherit function.

`{#1}` is a comma-separated list of things to be inherited

`{#2}` is the Container-list that should be inherited from

`{#3}` is the name of the inheriting Container

```

133 \def\cc@inherit#1#2#3{\cc@parse@inherit #1,,\@nil #2,,\@nil #3\@@nil}

```

`\cc@parse@inherit` is a low-level function to recursively parse the parameters of the `\cc@inherit` macro, above.

```

134 \def\cc@parse@inherit #1,#2,\@nil #3,#4,\@nil #5\@@nil{%
135   \let\next\relax
136   \if!#1!\else
137     \if!#3!\else
138       \cc@do@inherit{#1}{#3}{#5}%
139       \def\@argii{#2}\def\@argiv{#4}%
140       \ifx\@argii\@empty
141         \ifx\@argiv\@empty\else
142           \def\next{\cc@parse@inherit #1,,\@nil #4,\@nil #5\@@nil}%
143         \fi
144       \else
145         \ifx\@argiv\@empty
146           \def\next{\cc@parse@inherit #2,\@nil #3,,\@nil #5\@@nil}%
147         \else
148           \def\next{%
149             \cc@parse@inherit #1,,\@nil #4,\@nil #5\@@nil
150             \cc@parse@inherit #2,\@nil #3,#4,\@nil #5\@@nil
151           }%
152         \fi\fi\fi\fi
153   \next}

```

`\cc@do@inherit` is the macro that causes the parent's (unexpanded) Type list to be appended to the child's type list.

`{#1}` is the name of a Type

`{#2}` is the name of the Container that Type `{#1}` is *inherited* from

`{#3}` is the name of the Container that *inherits* Type `{#1}`

```

154 \def\cc@do@inherit#1#2#3{%
155   \ccDebugMsg[inheritance]{#3 inherits #1 from #2.}%
156   \ccCheckParent{#2}{#3}%
157   \expandafter\ifx\curname cc@type@#1@#2\endcsname\relax
158   \ccPackageError{Kernel}{Type}{Type `#1' was not declared for
159     Container `#2'.}%
159   \else
160     \edef\x{\noexpand\csgappto{cc@type@#1@#3}}%

```

```

161 \expandafter\x\expandafter{\csname cc@type@#1@#2\endcsname}%
162 \ccDebugMsg[inheritance]{value cc@type@#1@#3:~^J \expandafter\meaning\csname cc@type@#1@#3\
    endcsname}%
163 \fi
164 \ignorespaces}

```

## 5 Components

### 5.1 Simple Components

“Simple Components” are basically data storages. They are used within Containers to obtain data and store them for further processing at the end of the Container, or even beyond.

`\ccDeclareComponent` defines a simple Component macro. The internal macro that is used to store the Component’s value is `\csname cc@<current Container name>@<#1>\endcsname`.

`[#1]` is the Component’s identifier. If omitted, `{#1}` is the same as `{#2}`.

`{#2}` is the Component’s name

`{#3}` is code that is executed *before* assignment of the user’s value

`{#4}` is code that is executed *after* assignment of the user’s value

```

165 \def\ccDeclareComponent{\cc@opt@second\cc@declare@comp}
166 \def\cc@declare@comp[#1]#2#3#4{%
167 \ltx@LocalExpandAfter\global\expandafter\let\csname cc@\cc@cur@cont @#1\endcsname\relax
168 \expandafter\long\expandafter\def\csname \ccPrefix#2\endcsname##1{%
169 #3\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#1\endcsname{##1}#4\ignorespaces
    }%
170 \ignorespaces}

```

`\ccDeclareGlobalComponent` is a shortcut to declare simple, globally available Components with the name `{#2}` and an optional initial value `[#1]`. They are usually empty.

```

171 \def\ccDeclareGlobalComponent{\cc@opt@empty\cc@declare@global@comp}%
172 \def\cc@declare@global@comp[#1]#2{%
173 \ccDeclareComponent{#2}{\expandafter\global}{}}%
174 \if!#1!\else\csname \ccPrefix #2\endcsname{#1}\fi%
175 \ignorespaces}

```

Once declared, a component can be set in two ways: The first way is to use `\ccPrefix<name>` with one argument for its value. The second, preferred, way is to use the `\ccComponent` macro:

`\ccComponent` is the preferred way to fill a Component with content.

`{#1}` is the Component’s name

`{#2}` is the Instance value.

```

176 \long\protected\def\ccComponent#1#2{%
177 \ifx\cc@is@counted\relax
178 \ifcsdef{cc@\cc@cur@cont @#1}{-}
179 {\cc@def@counted@comp{\cc@counted@comp@scheme{#1}}{#1}{-}}%
180 \csgdef{cc@\cc@cur@cont @\cc@counted@comp@scheme{#1}}{#2}%
181 \else
182 \ifcsdef{cc@\cc@cur@cont @#1}{-}{\ccDeclareComponent{#1}{-}}%
183 \csdef{cc@\cc@cur@cont @#1}{#2}%
184 \fi\ignorespaces}

```

`\ccGlobalComponent` is a global variant of `\ccComponent`.

`{#1}` is the Component's name

`{#2}` is the Instance value.

```

185 \long\protected\global\def\ccGlobalComponent#1#2{%
186   \ifx\cc@is@counted\relax
187     \ifcsdef{cc@\cc@cur@cont @#1}{%
188       {\cc@def@counted@comp{\cc@counted@comp@scheme{#1}}{#1}{}}}%
189     \csgdef{cc@\cc@cur@cont @\cc@counted@comp@scheme{#1}}{#2}%
190   \else
191     \ifcsdef{cc@\cc@cur@cont @#1}{\ccDeclareGlobalComponent{#1}{}}}%
192     \csgdef{cc@\cc@cur@cont @#1}{#2}%
193   \fi\ignorespaces}

```

`\ccComponentEA` is a variant of `\ccComponent` but it expands the Content in `{#2}` once before it is assigned to the Component `{#1}`.

```

194 \long\protected\def\ccComponentEA#1#2{%
195   \def\x{\ccComponent{#1}}\expandafter\x\expandafter{#2}%
196   \ignorespaces}

```

`\ccUseComp` is a high level command to return (or print) the material stored as a Component with the name `{#1}`.

```

197 \def\ccUseComp#1{\csname cc@\cc@cur@cont @#1\endcsname}

```

`\ccdefFromComp` is a user-level command to store the value of a Component `{#2}` into a CS token `{#1}`.

```

198 \def\ccdefFromComp#1#2{\cc@store@comp{e}#1{#2}}

```

`\ccgdefFromComp` is the global variant of `\ccdefFromComp`.

```

199 \def\ccgdefFromComp#1#2{\cc@store@comp{x}#1{#2}}

```

`\ccpgdefFromComp` is a global variant of `\ccdefFromComp` that takes a CS name as `{#1}` and prepends the `\ccPrefix` before assigning the Contents of Component `{#2}` to the resulting CS token.

```

200 \def\ccpgdefFromComp#1#2{\def\x{\cc@store@comp{x}}\expandafter\x\csname\ccPrefix #1\endcsname
    {#2}}

```

`\strip@longprefix` is a helper macro to strip the prefix from the `\meaning` of a `\long` macro.

```

201 \def\strip@longprefix#1\long macro:->#2{#2}

```

`\cc@store@comp` is a generalized macro to store a component's unexpanded internal definition in a TeX macro.

`{#1}` is a scope quantifier (either 'e' or 'x')

`{#2}` is a CS token

`{#3}` is the name of a component

```

202 \long\def\cc@store@comp#1#2#3{%
203   \edef\@tempa{\expandonce{\csname protected@#1def\endcsname}\noexpand#2}%
204   \protected@edef\@tempb{\csname cc@\cc@cur@cont @#3\endcsname}%
205   \ifx\@tempb\relax
206     \let#2\relax
207   \else
208     \expandafter\@tempa\expandafter{\@tempb}%

```

```

209 \fi
210 \ignorespaces}

```

`\ccUseComponentFrom` is a high level command to return (or print) the material stored as a global Component from the Container `{#1}` with the name `{#2}`.

```

211 \def\ccUseComponentFrom#1#2{\csname cc@#1@#2\endcsname}

```

`\ccGetComp*` is a user-level command to return the contents stored in a Component of name `{#1}` as a paragraph iff the Component is neither empty nor `\relax`. If Accessibility features are activated, the returned content of the Component is automatically tagged with a `Para` tag.

The starred version of `\ccGetComp` suppresses automated tagging for that Component when the accessibility features are active.

```

212 \def\ccGetComp{\@ifstar\cc@sget@comp\cc@get@comp}
213 \def\cc@get@comp#1{%
214   \ccWhenComp{#1}
215   {\ccWhenAlly{\ccaStructStart{Para}}}%
216   \ccUseComp{#1}%
217   \ccWhenAlly{\ccaStructEnd{Para}}}%
218   \par}}
219 \def\cc@sget@comp#1{\ccWhenComp{#1}{\ccUseComp{#1}\par}}

```

`\ccIfComp` is a high level macro that executes `{#2}` if the Component macro `{#1}` is used in a Container (empty or non-empty), and `{#3}` if not.

```

220 \long\def\ccIfComp#1#2#3{\expandafter\ifx\csname cc@cc@cur@cont @#1\endcsname\relax#3\else#2\fi}

```

`\ccWhenComp` is a high level variant of `\ccIfComp` that omits the `else`-branch.

`{#1}` is the name of the Component

`{#2}` is code that is expanded when the Component `{#1}` is used in a container (empty or non-empty)

```

221 \long\def\ccWhenComp#1#2{\expandafter\ifx\csname cc@cc@cur@cont @#1\endcsname\relax\else#2\fi}

```

`\ccUnlessComp` is a high level variant of `\ccIfComp` that omits the `then`-branch.

`{#1}` is the name of the Component

`{#2}` is the code that is expanded when a Container `{#1}` is *not* used in a Container (neither empty nor non-empty)

```

222 \long\def\ccUnlessComp#1#2{\expandafter\ifx\csname cc@cc@cur@cont @#1\endcsname\relax#2\fi}

```

`\ccIfCompFrom` is the global variant of `\ccIfComp`.

`{#1}` is the name of the Container

`{#2}` is the name of the Component

`{#3}` is the then branch

`{#4}` is the else branch

```

223 \long\def\ccIfCompFrom#1#2#3#4{\expandafter\ifx\csname cc@#1@#2\endcsname\relax#4\else#3\fi}

```

`\ccIfCompFromVal` is a conditional to check if a Global Component holds a specific value.

`{#1}` is the name of the Container

`{#2}` is the name of the Component

{#3} is the comparison value  
 {#4} is the then branch  
 {#5} is the else branch

```
224 \long\def\ccIfCompFromVal#1#2#3#4#5{\protected@edef\@argiii{#3}\expandafter\ifx\csname cc@#1@#2\endcsname\@argiii#4\else#5\fi}
```

\cc@long@empty is a helper macro used as comparator when checking whether a \long macro is empty or not.

```
225 \long\def\cc@long@empty{}
```

\ccIfCompEmpty is a high level macro that executes {#2} if the Component macro {#1} is empty (or {}) within its Container, and {#3} if it is either not existant or non-empty.

```
226 \long\def\ccIfCompEmpty#1#2#3{\expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\cc@long@empty#2\else#3\fi}
```

\ccIfCompFromEmpty is a global variant of \ccIfCompEmpty.

{#1} is the name of the Container  
 {#2} is the name of the Component  
 {#3} is the then-branch  
 {#4} is the else-branch

```
227 \long\def\ccIfCompFromEmpty#1#2#3#4{\expandafter\ifx\csname cc@#1@#2\endcsname\cc@long@empty#3\else#4\fi}
```

\cc@check@empty handles the distinction between empty and un-used components: First, check if #4#3 is set (i. e., anything but \relax). If it is set, check if it is empty. If empty, set #4#3 to \relax, meaning further occurrences of \ccIfComp{#4#3} will execute the else branch. If #4#3 is non-empty, do nothing.

If #4#3 is already \relax, check if the fallback #1#3 is set. If so, make #4#3 an alias of #1#3. If not, do nothing.

[#1] is the prefix of the fallback component  
 {#2} is the Container name  
 {#3} is the name of the Component  
 {#4} is the Override's prefix

```
228 \def\cc@check@empty{\cc@opt@empty\cc@check@empty}%]
229 \def\cc@check@empty[#1]#2#3#4{%
230   \ccIfComp{#4#3}
231   {\ccIfCompEmpty{#4#3}
232     {\expandafter\global\expandafter\let\csname cc@#2@#4#3\endcsname\relax}
233     {}}
234   {\ccIfComp{#1#3}
235     {\expandafter\expandafter\expandafter\let\expandafter\csname cc@#2@#4#3\expandafter\endcsname\csname cc@#2@#1#3\endcsname}
236     {}}}
```

## 5.2 Counted Components

Counted Components are Components that may occur in the same parent Container multiple times. They may be multiple instances of single-macro Components, or recurring collections of multiple Components, called **Component Groups**.

## Component Groups

`\ccDeclareComponentGroup` is a user-level macro to declare a new Component Group with the name `{#2}` and the body `{#3}`. Optional `[#1]` holds Attribute handlers specific to that Component Group's Instances.

```

237 \def\ccDeclareComponentGroup{\cc@opt@empty\cc@declare@component@group}%
238 \def\cc@declare@component@group[#1]#2#3{%
239   \csnumgdef{cc#2Cnt}{\z@}%
240   \def\@argi{#1}\ifx\@argi@empty
241     \csgdef{cc@type@Attributes@\cc@cur@cont @cc#2}{}%
242   \else
243     \csgappto{cc@type@Attributes@\cc@cur@cont @cc#2}{#1}%
244   \fi
245   \long\csdef{\ccPrefix#2}{\cc@opt@empty{\csname cc@group@#2\endcsname}}%
246   \long\csdef{cc@group@#2}[##1]{%
247     \def\cc@cnt@grp{cc#2}%
248     \csxdef{cc#2Cnt}{\expandafter\the\expandafter\numexpr\csname cc#2Cnt\endcsname+\@ne\relax}%
249     \if!##1!\else
250       \ccEvalAttributes[\cc@cur@cont @cc#2]{##1}%
251       \csgdef{cc@\cc@cur@cont @#2-\csname cc#2Cnt\endcsname @attrs}{##1}%
252     \fi
253     #3%
254     \csname cc@group@#2@hook\endcsname
255     \ignorespaces
256   }%
257   \csdef{end\ccPrefix#2}{\ccToggleCountedConditionals\csuse{cc@compose@group@#2}\aftergroup\
    ignorespaces}%
258 }
```

`\ccAddToComponentGroup` can be used to locally add code `{#2}` to a previously defined Component Group `{#1}`.

```

259 \def\ccAddToComponentGroup#1#2{\csappto{cc@group@#1@hook}{#2}}
```

`\ccDeclareGroupHandler` is used to declare a new group handler. A Group Handler is a hook for code `{#2}` that is expanded at the end of a Component Group `{#1}`'s environment. It is mostly used to process Components within a Group instance and store the result in their own components. For instance, a Group Handler can be used to combine a First Name and a Surname to a combined Component "FullName".

```

260 \def\ccDeclareGroupHandler#1#2{%
261   \ifcsdef{cc@group@#1}
262     {\ifcsdef{cc@compose@group@#1}
263       {\csgappto{cc@compose@group@#1}{#2}}
264       {\csgdef{cc@compose@group@#1}{#2}}}
265     {\ccPackageError{Kernel}{Type}{Component Group `#1' unknown!}{You tried to declare a Group
      Handler for a Component Group that has not been declared, yet! Use \string\
      ccDeclareComponentGroup{#1}{ } to declare the Component Group first.}}%
266 \ignorespaces}
```

`\cc@cnt@grp` is a designated group name. Counted Components of the same group use the same counter.

```

267 \let\cc@cnt@grp@empty
```

`\ccUseCompByIndex` picks a Component with name `{#3}` and index `{#2}` from a group `{#1}`.

```

268 \def\ccUseCompByIndex#1#2#3{\csname cc@\cc@cur@cont @#1-#3-#2\endcsname}
```

`\ccUsePropFrom` picks a Counted Component with the index `{#2}` from a Group `{#1}` and renders it using Property `{#3}`.

```

269 \def\ccUsePropFrom#1#2#3{%
270   \begingroup
271   \@tempcnta\numexpr#2\relax
272   \letcs\ccTotalCount{cc#1Cnt}%
273   \def\cc@cnt@grp{cc#1}%
274   \ccToggleCountedConditionals
275   \csnumdef{cc#1Cnt}{\the\@tempcnta}%
276   \ccCurCount=\the\@tempcnta\relax%
277   \csname cc@\cc@cur@cont @#3\endcsname%
278   \endgroup}

```

### Iterating over Component Groups

The following two macros iterate over all instances of a Component Group `{#1}` in the current Container and applies for each instance the Property `{#2}`. The result is appended to the the Collector Component `{#3}`, if and only if that Component is not yet set for the current Container at the time of the first iteration.

While the first macro only writes the Property *definition* into the Collector Component, the second fully expands the macros inside the Property and stores the result in Component `{#3}`.

Use the former to print and the latter to further process the respective results.

`\ccCurCount` stores the number of the current instance of a Counted Component. Use this in the declarations of Properties that are expanded within the Component Group.

```

279 \newcount\ccCurCount

```

`\cc@assign@res` assigns the result of the Component collection to a control sequence with the name `{#1}` and resets the temporary storage.

```

280 \def\cc@assign@res#1{%
281   \ifx\cc@iterate@res\relax
282     \cslet{#1}\relax
283   \else
284     \expandafter\csname #1\expandafter\endcsname\expandafter{\cc@iterate@res}%
285   \fi
286   \global\let\cc@iterate@res\relax
287 }

```

`\ccIfComponentOverride` is a switch to apply either `{#2}`, if the Collection Component `{#1}` has been set manually within a container; or `{#3}`, if it has been generated from Counted Components.

```

288 \def\ccIfComponentOverride#1#2#3{\expandafter\ifx\csname cc@used@#1@override\endcsname \@empty#2\
    else#3\fi}

```

`\ccComposeCollection` is used to create an unexpanded Collection Component `{#3}` from all instances of Component Group `{#1}` using the instructions given by property `{#2}`.

```

289 \def\ccComposeCollection#1#2#3{%
290   \ccIfComp{#3}
291   {\cslet{cc@used@#3@override}\@empty}
292   {\cc@compose@collection{#1}{#2}%
293     \cc@assign@res{\ccPrefix#3}}

```

`\ccApplyCollection` is an alternative version of `\ccComposeCollection` and fully expands the result of the application of Property `{#2}` before it is stored inside the Component `{#3}`.



```

294 \def\ccApplyCollection#1#2#3{%
295   \ccIfComp{#3}
296   {\cslet{cc@used@#3@override}\@empty}
297   {\cc@apply@collection{#1}{#2}%
298    \cc@assign@res{\ccPrefix#3}}

```

`\cc@compose@collection` is a low-level macro used to compose a resource object from the unexpanded values of a Component Group {#1} using Property {#2}. The result is stored in `\cc@iterate@res` and can be retrieved with `\cc@assign@res`.

```

299 \def\cc@compose@collection#1#2{%
300   \expandafter\ifnum\csname cc#1Cnt\endcsname > \z@\relax
301   \edef\cc@iterate@res{%
302     \noexpand\bgroup
303     \noexpand\def\noexpand\ccTotalCount{\csname cc#1Cnt\endcsname}%
304     \noexpand\ccToggleCountedConditionals
305     \noexpand\def\noexpand\cc@cur@cont{\cc@cur@cont}%
306     \noexpand\def\noexpand\cc@cnt@grp{cc#1}%
307     \expandafter\@tempcntb=\csname cc#1Cnt\endcsname\relax
308     \cc@iterate{\@tempcnta}{\@ne}{\@tempcntb}{%
309       \edef\@tempb{%
310         \noexpand\beginingroup
311         %% top-level counter for user interaction
312         \noexpand\ccCurCount=\the\@tempcnta\relax
313         %% evaluating the current group's Attributes
314         \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}
315         {\noexpand\ccEvalAttributes[\cc@cur@cont @cc#1]
316          {\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}}%
317         %% internal counter for macro grabbing
318         \noexpand\csnumdef{cc#1Cnt}{\ccCurCount}%
319         \noexpand\ccUseProperty{#2}%
320         \noexpand\endingroup
321       }%
322       \expandafter\expandafter\expandafter\def
323       \expandafter\expandafter\expandafter\cc@iterate@res
324       \expandafter\expandafter\expandafter{\expandafter\cc@iterate@res\@tempb}%
325     }%
326     \expandafter\def\expandafter\cc@iterate@res\expandafter{\cc@iterate@res\egroup}%
327   \fi}

```

`\cc@apply@collection` is the low-level macro used to fully expand a Component Group {#1} using Property {#2}. The result is stored in `\cc@iterate@res` and can be retrieved with `\cc@assign@res`.

```

328 \def\cc@apply@collection#1#2{%
329   \beginingroup
330   \global\let\cc@iterate@res\relax
331   \letcs\ccTotalCount{cc#1Cnt}%
332   \cc@iterate{\@tempcnta}{\@ne}{\ccTotalCount}{%
333     \bgroup
334     \ccToggleCountedConditionals
335     \def\cc@cnt@grp{cc#1}%
336     \csnumdef{cc#1Cnt}{\the\@tempcnta}%
337     \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}
338     {\ccEvalAttributes[\cc@cur@cont @cc#1]
339      {\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}}%
340     \ccCurCount=\the\@tempcnta
341     \protected@xdef\@tempb{\csname cc@\cc@cur@cont @#2\endcsname}%
342     \@temptokena \expandafter{\@tempb}%
343     \def\@tempc{\csgappto{\cc@iterate@res}}%

```

```

344 \expandafter\@tempc\expandafter{\@tempb}%
345 \egroup
346 }%
347 \endgroup
348 }

```

`\cc@comp@edef` is used to pass a Counted Component into a TeX macro.

`{#1}` is a prefix to the def command, e.g., `\global` or `\protected`

`{#2}` is a CS token

`{#3}` is the Name of the Counted Component

`{#4}` is the Property that should be applied to all Members of the Counted Component

```

349 \def\cc@comp@edef{\cc@opt@empty\cc@comp@edef}
350 \def\cc@comp@edef[#1]#2#3#4{%
351 \cc@apply@collection{#3}{#4}%
352 \ifx\cc@iterate@res\relax
353 #1\let#2\relax%
354 \else
355 \def\@tempa{#1\def#2}%
356 \cc@assign@res{\@tempa}%
357 \fi
358 }

```

```

359 \def\cc@comp@def{\cc@opt@empty\cc@comp@def}
360 \def\cc@comp@def[#1]#2#3#4{%
361 \cc@compose@collection{#3}{#4}%
362 \ifx\cc@iterate@res\relax
363 #1\let#2\relax%
364 \else
365 \def\@tempa{#1\def#2}%
366 \cc@assign@res{\@tempa}%
367 \fi
368 }

```

```

369 \def\ccdefFromCountedComp{\cc@comp@def}
370 \def\ccgdefFromCountedComp{\cc@comp@def[\global]}

```

`\ccedefFromCountedComp` is the user-level command for *local* `\cc@comp@edef`.

```

371 \def\ccedefFromCountedComp{\cc@comp@edef}

```

`\ccxdefFromCountedComp` is the user-level command for *global* `\cc@comp@edef`.

```

372 \def\ccxdefFromCountedComp{\cc@comp@edef[\global]}

```

`\ccpxdefFromCountedComp` is the user-level command for *global* `\cc@comp@edef`. In contrast to `\ccxdefFromCountedComp`, it takes a CS name as first argument and prepends the `\ccPrefix` to the CS token to be defined.

```

373 \def\ccpxdefFromCountedComp#1{\expandafter\ccxdefFromCountedComp\csname \ccPrefix #1\endcsname}

```

## Declaring Counted Component

`\cc@counted@comp@scheme` gives the scheme how counted components are defined internally.

`{#1}` the name of the Counted Component.

```
374 \def\cc@counted@comp@scheme#1{\cc@cnt@grp-#1-\csname \cc@cnt@grp Cnt\endcsname}
```

`\ccDeclareCountedComponent` is a user-level macro to create a new Counted Component.

`{#1}` is the user-level name of the Component

```
375 \def\ccDeclareCountedComponent#1{%
376   \cc@def@counted@comp
377   {\cc@counted@comp@scheme{#1}}
378   {#1}
379   {}
380   {\expandafter\global}%
381   \ignorespaces}
```

`\cc@def@counted@comp` is used to declare Counted Components.

`{#1}` is the internal name of the Component which is composed out of the group name, the value of the group counter and the user-level macro name `{#2}`

`{#2}` is the name of the Counted Component

`{#3}` is some custom code passed to the second argument of `\ccDeclareComponent`

`{#4}` is a modifier to the internal macro definition.

```
382 \def\cc@def@counted@comp#1#2#3#4{%
383   \ccDeclareComponent[#1]{#2}
384   {\bgroup#3\expandafter\global}
385   {\def\@tempa{\@cc@reset@components@\cc@cur@cont}}%
386   \edef\@tempb{\noexpand\csgundef{\cc@noexpand\cc@cur@cont @#1}}%
387   \expandafter\expandafter\expandafter\csgappto\expandafter\@tempa\expandafter{\@tempb}%
388   \egroup}%
389   \ignorespaces
390   #4\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#2\endcsname{\csname cc@
391     cc@cur@cont @#1\endcsname}%
392   \ignorespaces}
```

### Resetting Counted Component

`\cc@reset@components` is used to reset Counted Components to prevent later Containers of a given type to feed the components from the previous Container of the same type. Usually, this is prevented by keeping Component definitions strictly local.

In some cases, however, Components may be declared globally, i.e., they may be re-used after the Container is ended. In this so-called Asynchronous Processing of Components, the reset should be done at the very beginning of the next instance of the container type to prevent bleeding of one container's components into the next one, specifically if a container occurs more than once in the same document.

`{#1}` is the name of the Component Group

```
392 \def\cc@reset@components#1{%
393   \csname @cc@reset@components@#1\endcsname
394   \global\cslet{\cc@reset@components@#1}\relax%
395 }
```

### Toggling Conditionals for Counted Components

`\ccToggleCountedConditionals` In order to process Counted Components, we need to re-define the Conditionals in a way such that the Component is expanded twice before the comparison takes place to correctly resolve the Component counter.

**Warning!** Use this macro only within local groups!

```
396 \long\def\ccToggleCountedConditionals{%
397   \let\cc@is@counted\relax
```

This re-definitions of `\ccIfComp` cannot use `etoolbox`'s `\cs...` macros since the conditional can be embedded inside itself. If an inner `csname` is undefined, the condition for the outer one would be reset before it can be expanded by `\ifx`.

```
398 \long\def\ccIfComp##1{%
399   \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
      csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@secondoftwo\else\expandafter\
      @firstoftwo\fi%
400 }%
401 \long\def\ccWhenComp##1{%
402   \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
      csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@gobble\else\expandafter\
      @firstofone\fi%
403 }%
404 \long\def\ccUnlessComp##1{%
405   \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
      csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@firstofone\else\expandafter\
      @gobble\fi%
406 }%
407 \long\def\ccIfCompEmpty##1{%
408   \expandafter\expandafter\expandafter\ifx\csname cc@\cc@cur@cont @##1\endcsname\cc@long@empty
      \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}%
409 \ccToggleCountedConditionalsHook% legacy
410 }
```

## 6 Properties

### 6.1 Setting Properties

`\cc SetProperty` is a user-level macro that provides the Property–Value interface for Containers.

`{#1}` is the name of the Property

`{#2}` is the Value assigned to that Property.

```
411 \long\def\ccSetProperty#1#2{\long\csdef{cc@\cc@cur@cont @#1}{#2}\ignorespaces}
```

`\cc AppToProp` can be used add material to the *end* of an existing Property vaue.

`{#1}` is the name of the Property

`{#2}` is the material to be added to previous value of that Property

```
412 \def\ccAppToProp#1#2{%
413   \long\csappto{cc@\cc@cur@cont @#1}{#2}%
414   \ignorespaces}
```

`\cc PreToProp` can be used add material to the *beginning* of an existing Property.

`{#1}` is the name of the Property

`{#2}` is the material to be inserted before the previous value of that Property

```
415 \def\ccPreToProp#1#2{%
```

```

416 \long\cspreto{cc@cc@cur@cont @#1}{#2}%
417 \ignorespaces}

```

`\ccPropertyLet` can be used to create an alias Property `{#1}` of a given Property `{#2}`. Is is equivalent to `\ccSetProperty{#1}{\ccUseProperty{#2}}`.

```

418 \long\def\ccPropertyLet#1#2{\long\csedef{cc@cc@cur@cont @#1}{\expandonce{\csname cc@
cc@cur@cont @#2\endcsname}}\ignorespaces}

```

`\ccPropertyLetX` creates a Property `{#1}` with the fully expanded value of another Property `{#2}`. Is is equivalent to `\ccSetPropertyX{#1}{\ccUseProperty{#2}}`.

```

419 \long\def\ccPropertyLetX#1#2{\long\csedef{cc@cc@cur@cont @#1}{\csname cc@cc@cur@cont @#2\
endcsname}\ignorespaces}

```

`\ccSetPropertyVal` is a variant of `\ccSetProperty` that expands the value `{#2}` *once* before assigning it to the Property macro with the name `{#1}`. This can be used to assign the current value of a variable macro, dimension, counter or length to a Property.

```

420 \long\def\ccSetPropertyVal#1#2{\def\@tempa{\ccSetProperty{#1}}\expandafter\@tempa\expandafter
{#2}\ignorespaces}

```

`\ccSetPropertyX` is another variant of `\ccSetProperty`, but it *fully expands* the value (using `\edef`) defined in `{#2}` before the Property is stored in the Property macro named `{#1}`. Use this if you need to use conditionals to determine the actual values of Properties that otherwise expect fixed named or dimensional values.

```

421 \long\def\ccSetPropertyX#1#2{\long\csedef{cc@cc@cur@cont @#1}{#2}\ignorespaces}

```

`\ccAddToProperties` adds the material in `{#2}` to a Container of name `{#1}`'s Properties List.

```

422 \long\def\ccAddToProperties#1#2{\ccAddToType{Properties}{#1}{#2}\ignorespaces}

```

## 6.2 Using Properties

`\ccUseProperty` is a user-level command to directly access a previously set Property with the name `{#1}`.

```

423 \def\ccUseProperty#1{\csuse{cc@cc@cur@cont @#1}}

```

`\cc@store@prop` stores the result of the application of property `{#3}` in the control sequence `{#2}`. The optional `[#1]` can hold a definition modifier like `\global` or `\long`.

```

424 \def\cc@store@prop{\cc@opt@empty\cc@store@prop}%
425 \long\def\cc@store@prop[#1]#2#3{%
426 \protected@edef\@tempa{\ccUseProperty{#3}}%
427 #1\expandafter\def\expandafter#2\expandafter{\@tempa}%
428 \ignorespaces}

```

`\ccdefFromProperty` expands an (implicit) Property `{#2}` and stores the result in (implicit) control sequence `{#1}`.

```

429 \def\ccdefFromProperty{\cc@store@prop}

```

`\ccgdefFromProperty` is the `\global` variant of `\ccdefFromProperty`.

```

430 \def\ccgdefFromProperty{\cc@store@prop[\global]}

```

`\ccpgdefFromProperty` is a `\global` variant of `\ccdefFromProperty` that takes a CS name as `{#1}` and prepends the `\ccPrefix` to it before assigning the Property result to the macro.

```
431 \def\ccpgdefFromProperty#1{\expandafter\ccgdefFromProperty\csname \ccPrefix #1\endcsname}
```

`\ccUsePropertyEnv` is a user-level command to access a previously set Property and make it an environment accessible to Property specific processing instructions (see below).

```
432 \def\ccUsePropertyEnv#1{\cslet{cc@#1@active}{\relax}\csuse{cc@\cc@cur@cont @#1}\csundef{cc@#1@active}}
```

`\ccIfStrEqual` is a variant of etoolbox's `\ifstrequal` that first fully expands both arguments `{#1}` and `{#2}` (using `\edef`) before comparing them.

```
433 \def\ccIfStrEqual#1#2{%
434   \edef\@argi{#1}\edef\@argii{#2}%
435   \expandafter\expandafter\expandafter\ifstrequal
436     \expandafter\expandafter\expandafter{\expandafter\@argi\expandafter}%
437     \expandafter{\@argii}}
```

## Local Property Overrides

`\cc@set@property@local` is a low-level macro to locally manipulate Properties.

`{#1}` is the CS token representing a method to alter the property (`\ccSetProperty`, `\ccAppToProp`, or `\ccPreToProp`)

`{#2}` is the name of the Property to be altered

`{#3}` is the new (or added) Value

```
438 \def\cc@set@property@locally#1#2#3{%
439   \let\@cc@cur@cont\cc@cur@cont
440   \ifdefstring\@cc@cur@cont{Heading}{\let\@cc@cur@cont\ccCurSecName}{}%
441   \csappto{cc@type@Properties@\@cc@cur@cont}{#1{#2}{#3}}%
442 }
```

The User level macros are Prefix sensitive. They exist in three flavours depending on whether the global Value of a Property should be kept or be replaced.

They all take two arguments:

`{#1}` is the name of the Property

`{#2}` is the value to be set, appended, or prepended to that Property, respectively.

`\ccSetPropLocal` sets a Property `{#1}` to a new value `{#2}`.

```
443 \def\ccSetPropLocal{\cc@set@property@locally\ccSetProperty}
444 \cslet{\ccPrefix SetPropLocal}\ccSetPropLocal%
```

`\ccAppPropLocal` appends the value `{#2}` to the *end* of an existing Property `{#1}`.

```
445 \def\ccAppPropLocal{\cc@set@property@locally\ccAppToProp}
446 \cslet{\ccPrefix AppPropLocal}\ccAppPropLocal%
```

`\ccPrePropLocal` appends the value `{#2}` to the *beginning* of an existing Property `{#1}`.

```
447 \def\ccPrePropLocal{\cc@set@property@locally\ccPreToProp}
448 \cslet{\ccPrefix PrePropLocal}\ccPrePropLocal%
```

## 6.3 Processing Instructions

In general, processing instructions are commands that are only visible to a specific process and ignored by others. In CoCoT<sub>E</sub>X, Processing Instructions (PIs) are commands placed inside a Component that should only take effect when that Component is processed through a specific Property.

`\ccPI` is a Processing Instruction that executes `{#2}` when a Property with the name `{#1}` is currently processed with the `\ccUsePropertyEnv` macro.

```
449 \DeclareRobustCommand\ccPI[2]{\ifcsdef{cc@#1@active}{#2}{}}
```

## 6.4 Property Conditionals

`\ccIfProp` checks if a Property with the name `{#1}` is defined and non-empty. If so, do `{#2}`, otherwise do `{#3}`.

```
450 \long\def\ccIfProp#1#2#3{%
451   \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\relax#3\else
452     \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\cc@long@empty #3\else#2\fi
453   \fi
454   \ignorespaces}
```

`\ccIfPropVal` checks if a Property `{#1}` expands to `{#2}`. If so, do `{#3}`, otherwise do `{#4}`.

**Warning:** Do not use this conditional in Properties that are used in `\ccApplyCollection`!

```
455 \long\def\ccIfPropVal#1#2#3#4{\long\def\@tempa{#2}%
456   \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\@tempa\relax#3\else#4\fi\ignorespaces}
```

# 7 Helper macros

## 7.1 Handling of Optional Arguments

Two simple internal macros to ease up the handling of optional arguments.

`\ccOpt@curcont` overrides stores the currently active Container name as future `#1`, unless the control sequence `{#1}` is called with an optional argument. In this case, the future `#1` is the value of that optional argument.

```
457 \long\def\ccOpt@curcont#1{\@ifnextchar[{#1}-{#1[\cc@cur@cont]}]{%}
```

`\ccOpt@empty` passes an empty string as future `#1` if the optional argument is missing.

```
458 \long\def\ccOpt@empty#1{\@ifnextchar[{#1}-{#1[]]}{%}
```

`\ccOpt@second` passes the first *mandatory* argument as value to the *optional* argument if the latter is missing.

```
459 \let\ccOpt@second\@dblarg
```

## 7.2 Iterators

`\cc@iterate` traverses in `[#1]`-th steps (defaults to +1) through counter `{#2}`, starting at number `{#3}` until and including number `{#4}` and does `{#5}` at every iteration (from `forloop.sty`, **Be aware** that incrementation of counter `{#2}` takes place after `{#5}` is called!):

```

460 \long\def\cc@iterate{\@ifnextchar[{ {\@cc@iterate}{\@cc@iterate[\@one]}}}%]
461 \long\def\@cc@iterate[#1]#2#3#4#5{%
462   #2=#3\relax%
463   \expandafter\ifnum#2>#4\relax%
464   \else
465     #5%
466     \advance#2 by #1\relax
467     \cc@iterate[#1]{#2}{\the#2}{#4}{#5}%
468   \fi}%

```

### 7.3 Attributes

Many macros and environments deal with optional arguments that are used to alter the behaviour of that macro or environment. The combination of a parameter and its set of possible values are called **Attributes**. In this section, we define the parsers for those parameters.

In order to catch the `babel` package's messing with the quote symbol, we make sure it has the correct cat-code.

```

469 \begingroup
470 \catcode`"=12

```

`\ccParseAttributes` High level wrapper for the attribute parser.

`{#1}` is the domain of the attribute

`{#2}` is the raw attribute list

```

471 \gdef\ccParseAttributes#1#2{%
472   \if!#1!\else
473     \if!#2!\else
474       \def\cc@cur@domain{#1}%
475       \cc@parse@attributes #2,,\@nil
476     \fi\fi}

```

The actual, recursively applying, parser comes in two parts:

`\cc@parse@attributes` parses the single attributes in an optional argument,

```

477 \gdef\cc@parse@attributes #1,#2,\@nil{%
478   \if!#1!\else
479     \cc@parse@kv#1==\@nil
480     \if!#2!\else
481       \cc@parse@attributes#2,\@nil
482     \fi\fi}

```

and

`\cc@parse@kv` distinguishes between the attribute name and its value(s).

```

483 \gdef\cc@parse@kv#1=#2=#3\@nil{%
484   \edef\@argii{#2}%
485   \ifx\@argii\@empty
486     \expandafter\let\csname cc@\cc@cur@domain @attr@#1\endcsname\@empty%
487   \else
488     \ifx #2 =\else
489       \expandafter\def\csname cc@\cc@cur@domain @attr@#1\endcsname{#2}%
490     \fi
491   \fi}

```



`\cc@parse@csv` takes a fallback macro `{#1}` and feeds it as argument to each item of the comma-separated list in the control sequence `{#2}`. The macro `{#1}` is stored internally as `\cc@parser@callback`.

```

492 \gdef\cc@parse@csv#1#2{%
493   \if!#1!\else
494     \let\cc@parser@callback#1%
495     \edef\cc@tempa{\csname #2\endcsname}%
496     \ifx\cc@tempa\@empty\else
497       \expandafter\cc@parse@csv\cc@tempa,,\@nil
498     \fi
499   \fi}

```

`\cc@parse@csv` applies `\cc@parser@callback` to the first item of a comma-separated pair and feeds the second item to itself.

```

500 \gdef\cc@parse@csv #1,#2,\@nil{%
501   \if!#1!\else
502     \cc@parser@callback{#1}%
503   \fi
504   \if!#2!\else
505     \cc@parse@csv#2,\@nil
506   \fi
507   \ignorespaces}
508 \endgroup

```

`\ccEvalAttributes` is a special Type Evaluator for Containers that define their Instance's attributes as Data Type. The Type then contains a list of `\ccDeclareAttributeHandler` statements for each of the allowed attributes.

`[#1]` is the Attribute Domain (defaults to the current Container name)

`{#2}` is the Container Instance's raw Attribute list.

```

509 \def\ccEvalAttributes{\cc@opt@curcont\cc@eval@attributes}%
510 \def\cc@eval@attributes[#1]#2{%

```

First we check if the Container Instance has a dedicated Attribute Type defined

```

511   \expandafter\ifx\csname cc@type@#1@Attributes\endcsname\relax

```

If so, we parse the Attribute list.

```

512   \ccParseAttributes{#1}{#2}%

```

After reading the Attribute list, we prepare unpacking the Attribute Data Type. Usually, the Type contains of a list of `\ccDeclareAttributeHandler` statements, but it can also handle the Attributes directly. The Attribute handler macro is defined locally:

`\ccDeclareAttributeHandler*` declares an Attribute handler. The *starred* version is for Attributes that are not expected to hold a value (i. e., switches), while the *non-starred* version is for Attributes that hold a value (key-value pairs). The value(s) for each matching Attribute is stored in `\ccAttrVal`. You may want to copy that value into another macro inside the third argument of the Handler macro for later evaluation, as it will be redefined by an Attribute Handler that is further down the Handler list.

`{#1}` is the name of the attribute (i. e., the part before the '=')

`[#2]` is code that is called when the Attribute *does not* occur in the Attribute list `{#1}`

`{#3}` is code that is called when the Attribute *does* occur in the Attribute list `{#1}`.

```

513   \def\ccDeclareAttributeHandler{%
514     \let\cc@is@starred\@undefined
515     \ifstar

```

```

516     {\let\cc@is@starred\relax\cc@declare@attribute@handler}
517     {\cc@declare@attribute@handler}}}%
518     \def\cc@declare@attribute@handler##1{\cc@opt@empty{\cc@declare@attribute@handler{##1}}}%
519     \def\cc@declare@attribute@handler##1[##2]##3{%
520         \let\ccAttrVal\relax
521         \ifx\cc@is@starred\relax
522             \ccIfAttrIsSet{#1}{##1}{##3}{##2}%
523         \else
524             \ccIfAttr{#1}{##1}
525             {\letcs\ccAttrVal{cc@#1@attr@##1}##3}
526             {##2}%
527         \fi\ignorespaces
528     }%

```

With the Handler macro in place, we evaluate the Attributes data Type, thus parsing the Attributes.

```

529     \ccEvalType[##1]{Attributes}%
530     \else

```

If the Container has no Attributes type defined, we check if the Container instance has, in fact, Attributes

```

531     \if!##2!\else

```

If so, we issue a warning since we cannot know how to deal with the Attributes.

```

532     \ccPackageWarning{Kernel}{Attribute}
533     {Container instance on line \inputlineno\space has Attributes,^^Jbut Container `##1'
534     provides no Attribute handlers!}
535     \fi
536     \fi
537     \ignorespaces}

```

`\ccGetAttribute` returns the value of an attribute.

`{##1}` is the attribute domain

`{##2}` is the attribute name

```

537 \def\ccGetAttribute##1##2{\csuse{cc@##1@attr@##2}}

```

`\ccIfAttr` can be used to call macros depending on whether an attribute is set, or not.

`{##1}` is the attribute domain

`{##2}` is the attribute name

`{##3}` is the then case

`{##4}` is the else case

```

538 \def\ccIfAttr##1##2##3##4{\ifcsdef{cc@##1@attr@##2}{##3}{##4}}

```

`\ccWhenAttr` is a variant of `\ccIfAttr` that omits the *else* branch.

`{##1}` is the attribute domain

`{##2}` is the attribute name

`{##3}` is the then case

```

539 \def\ccWhenAttr##1##2##3{\ifcsdef{cc@##1@attr@##2}{##3}{}}

```

`\ccUnlessAttr` is a variant of `\ccIfAttr` that omits the *then* branch.

`{#1}` is the attribute domain  
`{#2}` is the attribute name  
`{#3}` is the else case

```
540 \def\ccUnlessAttr#1#2#3{\ifcsdef{cc@#1@attr@#2}{#3}}
```

`\ccIfAttrIsStr` can be used to call macros depending if an attribute is set to the current (sub)container or group and what value it has.

`{#1}` is the attribute domain  
`{#2}` is the attribute name  
`{#3}` is the comparing value  
`{#4}` is the then case  
`{#5}` is the else case

```
541 \def\ccIfAttrIsStr#1#2#3#4#5{\ccIfAttr{#1}{#2}{\ifcsstring{cc@#1@attr@#2}{#3}{#4}{#5}}{#5}}
```

`\ccIfAttrIsSet` can be used to check if a value-less attribute has been set (i.e., it expands to `\@empty`).

`{#1}` is the attribute domain  
`{#2}` is the attribute name  
`{#3}` is the then case  
`{#4}` is the else case

```
542 \def\ccIfAttrIsSet#1#2#3#4{\ccIfAttr{#1}{#2}{\expandafter\ifx\cename cc@#1@attr@#2\endcsname\@empty#3\else#4\fi}{#4}}
```

## 7.4 Style Classes

Style Classes are locally usable sub-Containers.

**TODO**  
 Style Classes should be container-dependend.  
 Better, yet, incorporate style classes into the new Attribute Type!

`\ccDeclareClass` The top-level macro `\ccDeclareClass[#1]{#2}[#3]{#4}` has four arguments, two of which are optional. `{#2}` is the name of the class. If this argument is empty, the special class name `default` is used. `{#4}` is the declaration block of the class. This argument usually contains a set of Property assignments using the `\ccSetProperty{<prop>}{<val>}` macro, see Sect. 6. The first optional argument `[#1]` is the Style Class' parent Container. Using parent Containers, you can have Style Classes of the same name for different (sub-)Containers, e.g., a `default` class for each float and heading Container. The second optional argument `[#3]` is the parent Style Class. Properties from that Style Class are loaded automatically prior to the loading of the current Style Class's Properties. This applies recursively allowing for a cascading of property values, as in CSS.

```
543 \long\def\ccDeclareClass{\@ifnextchar [{\@cc@set@class}{\@cc@set@class[default]}}%
544 \long\def\@cc@set@class[#1]#2{\cc@opt@empty{\cc@set@class[#1]{#2}}}%
545 \long\gdef\cc@default@class@default{}
546 \long\def\cc@set@class[#1]#2[#3]#4{%
547   \def\@argii{#2}\ifx\@argii\@empty\let\@argii\cc@str@default\fi%
548   \if!#3!\else
549     \expandafter\long\expandafter\def\cename cc@#1@class@\@argii @parent\endcsname{#3}%
550   \fi
```

```

551 \expandafter\long\expandafter\def\csname cc@#1@class@{@argii\endcsname{#4}%
552 \ignorespaces}

```

`\ccUseStyleClass` is a user-level macro to expand and “activate” a Style Class’ Properties, those of its recursive ancestor Style Classes, and the default Style Class respecting the current Container.

`{#1}` is the Style Class name

`{#2}` is the Container name

```

553 \def\ccUseStyleClass#1#2{%
554 \expandafter\ifx\csname cc@#2@class@#1\endcsname\relax
555 \expandafter\ifx\csname cc@default@class@#1\endcsname\relax
556 \PackageError{cocotex.cls}{Class `#1' with scope `#2' not defined!}{Please declare the
    class `#1'!}%
557 \else
558 \PackageWarning{cocotex.cls}{Class `#1' with scope `#2' not defined! Reverting to default.}
    %
559 \fi
560 \fi
561 \csname cc@default@class@#1\endcsname%
562 \expandafter\ifx\csname cc@#2@class@#1@parent\endcsname\relax\else
563 \expandafter\ccUseStyleClass\expandafter{\csname cc@#2@class@#1@parent\endcsname}{#2}%
564 \fi
565 \csname cc@#2@class@#1\endcsname\ignorespaces}

```

## 7.5 The CoCoTeX Logo

`\CoCoTeX` the CoCoTeX Logo.

```

566 \DeclareRobustCommand\CoCoTeX{\texorpdfstring{{C\kern-.1em o\kern-.033emC\kern-.1em o}\kern-.133
    em\TeX}{CoCoTeX}}

```

```
</kernel>
```

## Module 3

# coco-common.dtx

---

```
<*common>
```

This file provides some macros that are used in more than one CoCoT<sub>E</sub>X module.

```
23 %%
24 %% module for CoCoTEX that provides some commonly used base macros.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-common}
32 [2024/07/16 0.5.0 CoCoTEX common module]
```

Load key/value option parser packages in case coco-common is used without the cls.

```
33 \RequirePackage{kvoptions-patch}
34 \RequirePackage{xkeyval}
35 \RequirePackage{iftex}
```

Default hook label for CoCoT<sub>E</sub>X modules:

```
36 \SetDefaultHookLabel{cc}
```

## 1 Package options

### 1.1 Accessibility Features

Default color encoding passed as option to the `xcolor` package.

```
37 \def\cc@str@cmyk{cmyk}
38 \def\cc@str@rgb{rgb}
39 \let\cc@color@enc\cc@str@cmyk
40 \define@choicekey{coco-common.sty}{color-enc}[\cc@color@enc\nr]{srgb,rgb,gray,cmy,cmyk,natural,
  none}[cmyk]{%
41   \let\cc@color@enc\cc@color@enc
42   \ifcase\nr\relax% srgb
43     \global\let\cc@color@enc\cc@str@rgb
44   \or% rgb
45   \or% gray
46   \or% cmy
47     \global\let\cc@color@enc\cc@str@cmyk
48   \or% cmyk
49   \or% natural, i.e. no conversion of color spaces takes place
```

```

50 \else%none
51 \global\let\cc@color@enc\relax
52 \fi
53 }
54 \ProcessOptionsX
55 \ifx\cc@color@enc\relax\else\PassOptionsToPackage{\cc@color@enc}{xcolor}\fi%

```

`\ccIfPreamble` is true as long as there has not been a `\begin{document}`.

```

56 \def\cc@if@preamble{\ifx\nodocument\relax\expandafter\@secondoftwo\else\expandafter\@firstoftwo
    \fi}
57 \let\ccIfPreamble\cc@if@preamble

```

## 2 Commonly Used Low-Level Macros and Registers

If CoCoTeX is used in conjunction with `xerif`<sup>1</sup>, we include the `coco-xerif` module, which, albeit not an official part of the CoCoTeX framework, is essential for the Framework to work with `xerif` generated `.tex` files.

```

58 \IfFileExists{coco-xerif.sty}{\RequirePackage{coco-xerif}}{}

```

The `coco-kernel` module contains the core functions of the CoCoTeX framework.

```

59 \RequirePackage{coco-kernel}

```

### 2.1 Hard Dependencies

Hard requirements for all CoCoTeX modules:

```

60 \RequirePackage{xcolor}

```

Including the `graphicx` package and catching case-insensitive graphics file's endings from Word:

```

61 \RequirePackage{graphicx}
62 \DeclareGraphicsRule{.EPS}{eps}{.EPS}{}

```

### 2.2 Common Variables

#### String Variables for Value Comparisons

`\cc@str@default` is a CS token that holds the string “default” for comparisons.

```

63 \def\cc@str@default{default}

```

`\cc@str@table` is a CS token that holds the string “table” for comparisons.

```

64 \def\cc@str@table{table}

```

`\cc@str@figure` is a CS token that holds the string “figure” for comparisons.

```

65 \def\cc@str@figure{figure}

```

<sup>1</sup>See <https://github.com/transpect/xerif/>

`\cc@str@top` is a CS token that holds the string “top” for comparisons.

```
66 \def\cc@str@top{top}
```

`\cc@str@bottom` is a CS token that holds the string “bottom” for comparisons.

```
67 \def\cc@str@bottom{bottom}
```

### Box Registers

Some temporary boxes that won’t interfere with LaTeX’s temporary boxes.

`\cc@tempboxa` is a temporary box register used throughout CoCoTeX.

```
68 \newbox\cc@tempboxa
```

`\cc@tempboxb` is another temporary box register used throughout CoCoTeX.

```
69 \newbox\cc@tempboxb
```

### Temporary Length and Skip Registers

`\cc@tempskipa` is a temporary skip register used throughout CoCoTeX.

```
70 \newskip\cc@tempskipa
```

## 2.3 Helper macros

`\cc@topstrut` is a `\strut` that has the height of `\topskip` and the depth of the difference between the `\baselineskip` and `\topskip`.

```
71 \def\cc@topstrut{\vrule\@width\z@\@height\topskip\@depth\dimexpr\baselineskip-\topskip\relax}
```

`\cc@afterbox` prevents indentation and additional spacing after environments. Intended to be used in combination with `\aftergroup`.

```
72 \def\cc@afterbox{%
73   \everypar{%
74     \if@nobreak
75       \nobreakfalse
76       \clubpenalty \@M
77       \if@afterindent \else
78         {\setbox\z@\lastbox}%
79         \everypar{}%
80       \fi
81     \else
82       \clubpenalty \@clubpenalty
83       {\setbox\z@\lastbox}%
84       \everypar{}%
85     \fi}}
```

## 2.4 Masks

These macros are intended to mask non-content markup, like page- or line breaking commands in order to find and remove or alter them easier.

`\hack` intended to mask line breaking macros.

```
86 \let\hack\@firstofone
```

`\hackfor` intended to hide line breaking macros.

```
87 \let\hackfor\@gobble
```

`\Hack` intended to mask page breaking macros.

```
88 \let\Hack\@firstofone
```

`\Hackfor` intended to hide page breaking macros.

```
89 \let\Hackfor\@gobble
```

`\@gobbleopt` intended to nullify a macro's argument with a possible optional argument interfering.

Use it like this: `\let\yourMacroWithOptArg\@gobbleopt`

```
90 \long\def\@gobbleopt{\@ifnextchar[\@gobbleopt{\@gobbleopt[]}}%
91 \long\def\@gobbleopt[#1]#2{%
```

`\ccGobble` is used to de-activate certain macros to prevent them from being called multiple times while processing contents. An example is a footnote inside a caption while calculating the height of the caption. In this case, we need the space the footnote symbol requires without the actual footnote being written into the footnote insert, since that should happen when we actually print the caption.

```
92 \def\ccGobble{%
93   \renewcommand\footnote[2][\the\c@footnote]{\def\@thefnmark{##1}\@makefnmark}%
94   \renewcommand\index[2][]{}%
95   \renewcommand\marginpar[2][]{}%
96   \renewcommand\glossary[2][]{}%
97   \let\hypertarget\@gobbletwo
98   \let\label\@gobble
99 }%
```

## 2.5 Arithmetics

`\CalcRatio` is used to calculate the ratio between two integers `{#1}` and `{#2}`.

```
100 \def\CalcRatio#1#2{\strip@pt\dimexpr\number\numexpr\number\dimexpr#1\relax*65536/\number\dimexpr
    #2\relax\relax sp}
```

`\CalcModulo` is used to calculate the remainder of integer division of `{#1}` by `{#2}`. This needs a different approach than the common modulo definition, which would return negative results in some cases, as TeX rounds up the quotient of `{#1}` and `{#2}` if the first decimal place is equal to or greater 5.

```
101 \def\CalcModulo#1#2{\number\numexpr#1+#2-((#1+#2/2)/#2)*#2\relax}
```

`\minusvspace` Counterpart to L<sup>A</sup>T<sub>E</sub>X's `\addvspace`: if the value of `\minusvspace` is larger than `\lastskip`, `\lastskip` is used. Otherwise, the value of `\minusvspace` is used.



```

102 \def\@xminusvskip{%
103   \ifdim\lastskip<\@tempskipb
104   \else
105     \ifdim\lastskip<\z@
106     \else
107       \ifdim\@tempskipb<\z@
108       \advance\@tempskipb\lastskip
109       \fi
110       \vskip-\lastskip
111       \vskip \@tempskipb
112     \fi
113   \fi}
114 \def\minusvspace#1{%
115   \ifvmode
116     \if@minipage\else
117       \ifdim \lastskip =\z@

```

Compatibility to texlive pre 2020:

```

118   \ifx\@vspace@calcify\@undefined
119     \vskip #1\relax
120   \else
121     \@vspace@calcify{#1}%
122   \fi
123   \else
124     \setlength\@tempskipb{#1}%
125     \@xminusvskip
126   \fi
127 \fi
128 \else
129   \@noitemerr
130 \fi}

```

## 2.6 Determine actual page number

We need to determine the real page a floating object is printed. This mechanism is largely an adaption of the mechanism used in the `marginnote` package.

Counting absolute page numbers, however, may be misleading when the `coco-title` module is loaded and the cover page is not followed by an empty page. Therefore, we save the default page counter from L<sup>A</sup>T<sub>E</sub>X to evaluate it independently from the actual manner of counting.

`\the@cc@thispage` temporarily stores the current page number.

```

131 \def\the@cc@thispage{%

```

`\cc@abspage` is a counter for the absolute page number.

```

132 \newcount\cc@abspage \cc@abspage\z@

```

`\thecc@abspage` is the output formatter for the `\cc@abspage` counter.

```

133 \def\thecc@abspage{\the\cc@abspage}

```

`\ifcc@odd` is a conditional that is set to true if the current absolute page number is not divisible by 2.

```
134 \newif\if@cc@odd \@cc@oddtrue
```

The absolute page counter is injected directly into L<sup>A</sup>T<sub>E</sub>X's output routine:

```
135 \AtBeginDocument{%
136   \global\cc@abspage=\c@page\relax%
137   \g@addto@macro\@outputpage{\global\cc@abspage\c@page}%
138 }
```

We split the testing mechanism into two parts.

`\ccTestPage` is run before the floating object is placed. It will store the page according to the placement in the tex source code.

```
139 \def\ccTestPage{%
140   \expandafter\ifx\csname the@cc@thispage\endcsname\@empty
141   \gdef\the@cc@atthispage{1}%
142   \else
143   \expandafter\ifnum \the@cc@thispage=\cc@abspage%
144     \begingroup
145     \@tempcnta\the@cc@atthispage\relax
146     \advance\@tempcnta\@ne\relax
147     \xdef\the@cc@atthispage{\the\@tempcnta}%
148   \endgroup
149   \else
150   \gdef\the@cc@atthispage{1}%
151   \fi
152 \fi
153 \xdef\the@cc@thispage{\the\cc@abspage}%
154 \let\@cc@curpage\relax
155 \expandafter\ifx\csname \cc@cur@cont-\the@cc@thispage-\the@cc@atthispage\endcsname\relax
156   \ifodd\cc@abspage\relax\@cc@oddtrue\else\@cc@oddfalse\fi
157 \else
158   \edef\@cc@curpage{\expandafter\expandafter\expandafter\@firstofone\csname \cc@cur@cont-\the@cc@thispage-\the@cc@atthispage\endcsname}%
159   \ifodd\@cc@curpage\relax\@cc@oddtrue\else\@cc@oddfalse\fi
160 \fi
161 }
```

`\ccSavePage` is the second macro, which writes the actual page number into the aux files.

```
162 \def\ccSavePage{%
163   \protected@write\@auxout{\def\the@cc@cur@cont{\cc@cur@cont}\let\thecc@abspage\relax}{%
164     \string\expandafter\string\gdef\string\csname\space \cc@cur@cont-\the@cc@thispage-\the@cc@atthispage\string\endcsname{\thecc@abspage}}%
165 }
```

## 3 Re-Thinking L<sup>A</sup>T<sub>E</sub>X Core Functions

### 3.1 Keeping .aux-Files Up-to-Date

`\ccBreak` is a general line break macro intended to be re-defined if necessary without touching L<sup>A</sup>T<sub>E</sub>X's kernel page and line breaking macros.

```
166 \DeclareRobustCommand*\ccBreak{\hfill\break}
```

```
167 \ifx\ccPrefix\@empty\else\cslet{\ccPrefix break}\ccBreak\fi
```

## 3.2 Content lists

### Default L<sup>A</sup>T<sub>E</sub>X Content Lists

This part contains macros to ‘simplify’ the generation of content lists like the Table of Contents or List of Figures/Tables, etc.

Entries in the list-files (e.g., `\jobname.toc`, `\jobname.lof`, etc.) usually contain `\contentsline` macros that expand to `l@<level>`. Whenever a level of Components that are to be written into content lists is declared, the package automatically generates a `\cc@l@<level>` macro for this level of entries. The content-baring argument of `\ccContentsline` (or `\cc@l@<level>`, resp.) contains Components.

Once a list file is read, those `\cc@l@<level>` macros are expanded in two steps. Each entry constitutes a Container in its own right. It therefore can have multiple Components. The first step is the extraction phase, where the entry’s Container is dynamically declared, the corresponding properties are initialised, and its Components are extracted

`\cc@init@l@` is a low-level macro used to dynamically define `\cc@l@<level>` macros.

[#1] is an override for counters that have to be restored  
 {#2} is the list file ending (raw entries being stored in a file `\jobname.\#2`)  
 {#3} is a number that indicated the nesting depth  
 {#4} is the nested level’s unique name.

```
168 \def\cc@init@l@{\cc@opt@empty\@cc@init@l@}%
169 \def\cc@init@l@[#1]#2#3#4{%
170   \expandafter\ifx\csname c@#2depth\endcsname\relax
171     \expandafter\global\expandafter\newcount\csname c@#2depth\endcsname
172     \expandafter\global\csname c@#2depth\endcsname=0\relax
173   \fi
174   \expandafter\ifx\csname cc@#2@extract@data\endcsname\relax
175     \expandafter\let\csname cc@#2@extract@data\endcsname\cc@extract@generic
176   \fi
177   \expandafter\ifx\csname cc@#2@print@entry\endcsname\relax
178     \expandafter\let\csname cc@#2@print@entry\endcsname\cc@print@generic
179   \fi
180   \expandafter\long\expandafter\gdef\csname cc@l@#4\endcsname##1##2{%
181     \ifLuaTeX\suppresslongerror=1\fi
182     \expandafter\ifnum \csname c@#2depth\endcsname<#3\relax
183       \else
184         \bgroup
```

`\ccTocLink` is used to link list entries to their destination.

```
185 \long\def\ccTocLink####1{\hyper@linkstart{link}{\@contentsline@destination}{####1}\hyper@linkend}%
```

```
186 \csname cc@#2@extract@data\endcsname{#3}{#4}{##1}{##2}%
187 \csname cc@#2@print@entry\endcsname{#4}%
188 \egroup
189 \fi
190 \ifLuaTeX\suppresslongerror=0\fi
191 }}
```

`\ccContentsline` is our version of L<sup>A</sup>T<sub>E</sub>X’s `\contentsline`.

{#1} is the name of the list counter

{#2} is the name of the list entry  
 {#3} is the page number  
 {#4} is the hyperref destination

```
192 \long\def\ccContentsline#1#2#3#4{\gdef\@contentsline@destination{#4}%
193 \csname cc@l@#1\endcsname{#2}{#3}}
```

`\cc@extract@generic` is a fallback extractor for a list entry. It is used when the list handler does not provide a case-specific extractor for the entries.

{#1} is the name of the list counter  
 {#2} is the name of the list entry  
 {#3} is the page number  
 {#4} is the hyperref destination

```
194 \def\cc@extract@generic#1#2#3#4{}
```

`\cc@print@generic` is the fallback output generator for the composed list entry {#1}.

```
195 \def\cc@print@generic#1{}
```

`\cc@expand@l@contents` expands the content of the `cc@l@<level>` macro and contains some code to catch and handle standard L<sup>A</sup>T<sub>E</sub>X headings.

{#1} is the content of the `cc@l@`-Macro  
 {#2} is the name of the handling Container  
 {#3} is the Component prefix  
 {#4} is the name of the Content component

```
196 \def\cc@expand@l@contents#1#2#3#4{%
197 \global\let\cc@tempa\relax
198 \sbox\z@{\def\numberline##1{\xdef\cc@tempa{\noexpand\csdef{cc@#2@#3Number}{##1}}{#1}}%
199 \ifdim\wd\z@>\z@
200 \let\numberline\@gobble%
201 \protected\csdef{cc@#2@#3#4}{#1}%
202 \cc@tempa
203 \else
204 #1%
205 \fi
206 \global\let\cc@tempa\relax
207 }
```

## Custom Content Lists

`\ccDeclareContentList` provides an interface for additional content lists.

{#1} is the name of the custom content  
 {#2} is a comma separated list of container names the instances of which should be listed in the custom contents list

```
208 \def\ccDeclareContentList#1#2{%
209 \def\cc@add@extra@cl##1{%
210 \expandafter\ifx\csname cc@##1@extra@cl\endcsname\relax
211 \csgdef{cc@##1@extra@cl}{#1}%
212 \else
213 \csgappto{cc@##1@extra@cl}{, #1}%
214 \fi}%
215 \edef\@argii{#2}%
216 \cc@parse@csv\cc@add@extra@cl{\@argii}%
```

```

217 \expandafter\newwrite\csname cc@cl@#1\endcsname\relax
218 }

```

`\ccCreateContentListEntries` creates entries for Custom Content Lists. It is called during the processing of a container's instance.

`{#1}` is the name of the calling Container  
`{#2}` is the name of the file stream  
`{#3}` is the level of the entry  
`{#4}` is the current page counter  
`{#5}` is the current hyperref label

```

219 \def\ccCreateContentListEntries#1#2#3#4#5{%
220   \def\cc@add@extra@cl##1{%
221     \expandafter\protected@write\csname cc@cl@##1\endcsname
222       {\ccGobble}%
223     {\protect\ccContentsline{#2}{#3}{#4}{#5}\protected@file@percent}\relax
224   }%
225   \ifcsdef{cc@#1@extra@cl}{%
226     \cc@parse@csv\cc@add@extra@cl{cc@#1@extra@cl}}{%
227   }

```

### 3.3 Indentation and Left Margins of Potentially Numbered Items

The **left margin** means the space between the left border of the page area and the imaginary line that multi-line text aligns to. The **indent** is the offset of the very first line of that block of text relative to that value.

If the **indent** is a negative value you'll get a hanging indent; if it is positive, you get a paragraph style indent, and if it is set to **Opt**, you get a clean alignment of the whole item.

CoCoTeX provides a feature that allows the indentation of counted elements to be just as wide as the widest Number of the same level (if **indent** is set to **auto**), as well as a feature that allows the indent to be as wide as all Numbers of the same container type (if **indent** is set to **auto-global**).

The approach to set the **indent**, **margin-left** and the position of the Number Component in numbered items such as Headings, entries in ToC and listof-X, captions, etc. is to store the maximum width for each level and the maximum width across all Numbers of a Container Type in the .aux file at the very end of the compilation after it has been constantly updated during the entire L<sup>A</sup>T<sub>E</sub>X runtime. That way, for the next L<sup>A</sup>T<sub>E</sub>X run, the maximum values are available immediately and can be used to fortify those parameters.

`\cc@store@latest` is a low-level macro that stores the maximum value of a dimension Property `{#1}`. An internal Property `\#1-local` is constantly updated whenever the macro is called and the previously stored value is lower than the one given in `{#2}`.

The first call of the macro for a given Property triggers an addendum to the `\@enddocumenthook` which causes the last value for that dimension to be stored in the .aux file. If the Property hasn't been set from a previous L<sup>A</sup>T<sub>E</sub>X run or a previous call to the `\cc@store@latest` macro for the same Property and the same level, it is set to `{#2}`.

`{#1}` is the internal name of the property  
`{#2}` is the check value.

```

228 \def\cc@store@latest#1#2{%
229   \expandafter\ifx\csname cc-\cc@cur@cont-#1\endcsname\relax
230     \csxdef{cc-\cc@cur@cont-#1}{#2}%
231   \else
232     \expandafter\ifdim\csname cc-\cc@cur@cont-#1\endcsname<#2\relax
233       \csxdef{cc-\cc@cur@cont-#1}{#2}%
234     \fi
235   \fi

```

```

236 \expandafter\ifx\csname cc-\cc@cur@cont-#1-local\endcsname\relax
237 \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
238 \else
239 \expandafter\ifdim\csname cc-\cc@cur@cont-#1-local\endcsname<#2\relax
240 \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
241 \fi
242 \fi

```

The second step is to store the highest values in the .aux file for later LaTeX runs. A `\write\@auxout` command for the storage macro is therefore added to the `\@enddocumenthook` and a flag is set that indicates that the write command has already been added to the hook, since that needs to be done only once for each to-be-stored dimension.

Note that the value that is eventually stored, is the updated *local* maximum, not the value that is retrieved at the beginning of the run. This allows the values to be down-graded if the LaTeX source changed during two consecutive runs. However, if values change, you still need to do at least two more  $\LaTeX$  runs before the values stabilize.

```

243 \ifcsdef{cc-\cc@cur@cont-#1-stored-trigger}{%
244   {\edef\@tempa{%
245     \noexpand\immediate\noexpand\write\noexpand\@auxout{%
246       \noexpand\string\noexpand\csgdef{cc-\cc@cur@cont-#1}{%
247         \noexpand\csname cc-\cc@cur@cont-#1-local\noexpand\endcsname}}}%
248     \expandafter\AtEndDocument\expandafter{\@tempa}%
249     \csgdef{cc-\cc@cur@cont-#1-stored-trigger}{\@empty}}}%

```

`\cc@format@number` calculates number widths and prepares macros to be used by the user.

`{#1}` is the internal Property prefix  
`{#2}` is the user-level Component prefix  
`{#3}` is the numerical list level.

```

250 \def\cc@format@number#1#2#3{%
251   \ccSetPropertyVal{#1curr-number-level}{#3}%

```

*First step:* measuring the natural width of the Number if it exists for the current item.

```

252 \ccIfComp{#2Number}
253   {\sbox\z@{\ccUseProperty{#1number-format}}}
254   {\sbox\z@{}}%

```

*Second step:* we store the width of `\box0` if it is wider than the previously stored width for that level. The end value will be written into the .aux file during expansion of the `\@enddocumenthook`. We do the same for the maximum across *all* levels of the same Container Type.

```

255 \cc@store@latest{#1number-#3-maxwd}{\the\wd\z@}%
256 \cc@store@latest{#1number-maxwd}{\the\wd\z@}%

```

We provide the maximum level as a user-level Property `#1number-width-level-max`, the global maximum across all levels as `#1number-width-max`, and the width of the current number as `#1number-width`.

```

257 \ccSetPropertyVal{#1number-width-level-max}{\csname cc-\cc@cur@cont-#1number-#3-maxwd\
  endcsname}%
258 \ccSetPropertyVal{#1number-width-max}{\csname cc-\cc@cur@cont-#1number-maxwd\endcsname}%
259 \ccSetPropertyVal{#1number-width}{\the\wd\z@}%

```

*Third step:* we calculate and fortify the actual `#1margin-left` (i.e., the overall left indent of the whole item) and `#1indent` (offset of the first line) of the entry.

```

260 \cc@get@indent{#1}{#3}%
261 \cc@set@hang{#1}%
262 }

```

`\cc@set@hang` determines and sets the hanging indent of a counter.

`{#1}` is the internal Property prefix

```
263 \def\cc@set@hang#1{%
```

First, we set the `#1hang-number` to be an alias of `#1number-format` as fallback.

```
264 \ccPropertyLet{#1hang-number}{#1number-format}%
```

Then, we check for `#1indent`.

```
265 \ccIfProp{#1indent}  
266 {\ifdim\ccUseProperty{#1indent}<\z@
```

If it is set and negative, we alter the `#1hang-number` Property in such a way that it is shifted to the left by `#1indent` amount and put into a hbox of `-#1indent` width (remember that the value is negative).

```
267 \ccSetProperty{#1hang-number}{%  
268 \hskip\ccUseProperty{#1indent}%  
269 \hbox to -\ccUseProperty{#1indent}{%  
270 \ccIfPropVal{#1number-align}{left}{\hss}%  
271 \ccUseProperty{#1number-format}%  
272 \ccIfPropVal{#1number-align}{right}{\hss}}%  
273 \fi}{}}
```

In all other cases, we stick to the default (`#1number-format`) we set in the first step.

`\cc@calc@margin@left` determines the left margin of the current level by subtracting the current level's indent from the left margin of the next-higher level. “Next-higher” meaning “hierarchically”, i.e., the level counter is *lower*. Remember that for hang indent, the indent is negative, so `margin-left` grows larger.

`{#1}` is the Property prefix

`{#2}` is the current numerical list level.

```
274 \def\cc@calc@margin@left#1#2{%  
275 \@tempcnta\numexpr#2-\@ne\relax  
276 \expandafter\ifx\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname\relax  
277 \@tempdima=-\ccUseProperty{#1indent}\relax%  
278 \else  
279 \@tempdima=\dimexpr\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname-\  
ccUseProperty{#1indent}\relax  
280 \fi  
281 \cc@store@latest{#1#2-margin-left}{\the\@tempdima}%  
282 \ccSetProperty{#1margin-left}{\the\@tempdima}}
```

`\cc@get@indent` Eventually, write the actually used values for `margin-left` and `indent` into the current container's Property list.

`{#1}` is the CS token of a method that is called to calculate the actual left margin of the list item. It defaults to above's

`\cc@calc@margin@left` and is fed the two mandatory arguments of the `\cc@get@indent` macro, namely

`{#2}` for the internal property prefix, and

`{#3}` for the numerical list level.

The callback method should set and store the `#2margin-left` Property.

```
283 \def\cc@get@indent{\@ifnextchar[{\cc@get@indent}{\cc@get@indent[\cc@calc@margin@left]}}  
284 \def\cc@get@indent[#1]#2#3{%
```

First, we need to store the initial values for both `#2margin-left` and `#2indent` since, first their values might be non-dimensional, and second, they will be altered during macro expansion to ultimately being passed to `\hskip`.

```

285 \ccPropertyLetX{int-#2margin-left}{#2margin-left}%
286 \ccPropertyLetX{int-#2indent}{#2indent}%
287 \ccIfPropVal{#2indent}{auto-global}

```

If `#2indent` is set to `auto-global`, the item gets an `indent` that is set to the negative value of the maximum width of all numbers across all Levels of the same Container Type. The same maximum is added to the user-set value of `margin-left`.

```

288 {\ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-max}}}%

```

If the user has set `#2margin-left` to `auto`, we reset it to empty.

```

289 \ccIfPropVal{#2margin-left}{auto}{\ccSetProperty{#2margin-left}{}}{}%

```

If the user has not set `margin-left`, we set it to `\z@`.

```

290 \ccIfPropVal{#2margin-left}{
291   {\ccSetProperty{int-#2margin-left}{\z@}}
292   {\ccPropertyLetX{int-#2margin-left}{#2margin-left}}%
293   \ccSetPropertyX{#2margin-left}{\dimexpr\ccUseProperty{#2number-width-max}+\ccUseProperty{int
    -#2margin-left}\relax}}

```

Next, we check if `#2margin-left` is set to `auto`.

```

294 {\ccIfPropVal{int-#2margin-left}{auto}

```

If `#2margin-left` is set to `auto`, all items of the same level get the same left margin that is determined by the sums of the indents of all higher levels.

```

295 {\ccIfPropVal{int-#2indent}{auto}

```

if `#2indent` is also set to `auto`, the `indent` of the current item is set to the widest Number of the same level.

```

296 {\ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}

```

otherwise it is set to the value of `indent`, or `Opt` if it was not set at all.

```

297 {\ccIfProp{int-#2indent}
298   {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}
299   {\ccSetProperty{#2indent}{\z@}}}%

```

the final value for `margin-left` is calculated. If no optional argument is given, the method called is the `\cc@calc@margin@left` macro, above.

```

300 #1{#2}{#3}}

```

This branch is reached when the left margin is not set to `auto`.

```

301 {\ccIfProp{int-#2margin-left}
302   {\ccIfPropVal{int-#2indent}{auto}

```

If `margin-left` is set to a specific value and `indent` is set to `auto`, set the actual indent to the width of the level's widest Number.

```

303 {\ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}
304 {\ccIfProp{int-#2indent}

```

Otherwise, if `indent` is set to a specific width, apply that value, or else set the inden to `Opt`.



```

305 {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}
306 {\ccSetProperty{#2indent}{\z@}}}
```

If `margin-left` is not set,

```

307 {\ccIfPropVal{int-#2indent}{auto}}
```

and `indent` is set to `auto`, set `margin-left` to the width of the level's widest Number and the actual `indent` to the negative of that.

```

308 {\ccPropertyLetX{#2margin-left}{#2number-width-level-max}%
309 \ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}
310 {\ccIfProp{int-#2indent}}
```

If `margin-left` is not set, and `indent` is set to a specific value, apply that value for `indent` and set `margin-left` to `Opt`. In this branch, `indent` should have a positive value, otherwise the content would probably lap over the left edge of the type area.

```

311 {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}%
312 \ccSetProperty{#2margin-left}{\z@}}
```

otherwise set both `indent` nad `margin-left` to `Opt`.

```

313 {\ccSetProperty{#2indent}{\z@}%
314 \ccSetProperty{#2margin-left}{\z@}}}}}
```

### 3.4 Labelling and Cross referencing

CoCoT<sub>E</sub>X provides two ways to put labels on Container instances: one via the label attribute at the begin of a (Sub-)Containers corresponding environment, or via the `RefLabel` Component inside the (Sub-)Container.

```

315 \AtBeginDocument{%
```

Storing the final definitions of `\label`

`\cc@ltx@label` stores the definition of LaTeX's `\label` macro at the beginning of the document.

```

316 \global\let\cc@ltx@label\label
317 }
```

`\ccCreateLabel` is a high level macro to generate hyperref anchors and/or ref targets.

`{#1}` is the type of anchor

This macro looks for both the label attribute in the begin of a Container's environment, as well as for a `RefLabel` Components inside the environment. If both exist, both apply. If none exists, we adopt the generic anchor point generated by the `hyperref` package.

**TODO**  
Check if the hyperref  
macros need to be  
configured in any way for  
various reference types!

```

318 \def\ccCreateLabel#1{%
319 \ifx\Hy@MakeCurrentHrefAuto\undefined\else
320 \Hy@MakeCurrentHrefAuto{cc:#1}%
```

```

321 \Hy@raisedlink{\hyper@anchorstart{\@currentHref}\hyper@anchorend}%
322 \fi
323 \let\cc@ref@label\relax
324 \ccWhenComp{RefLabel}
325 {\ccgdefFromComp\cc@ref@label{RefLabel}%
326 \expandafter\cc@create@label\expandafter{\cc@ref@label}}%
327 \ccIfAttr{\cc@cur@cont}{label}
328 {\cc@parse@csv\cc@create@label{\cc@cur@cont @attr@label}}%
329 {\ifx\cc@ref@label\relax\cc@create@label{\@currentHref}\fi}}

```

`\cc@create@label` generates the actual anchor for document-internal cross-references (i.e., a  $\LaTeX$  `\label`).

`{#1}` is the label ID

```

330 \def\cc@create@label#1{%
331 \ccIfComp{Number}
332 {\ifx\cc@labelname@comp\@undefined
333 \def\cc@labelname@comp{Title}%
334 \fi
335 \begingroup
336 \ccGobble
337 \ccgdefFromComp\@currentlabel{Number}%
338 \ccgdefFromComp\@currentlabelname{\cc@labelname@comp}%
339 \endgroup}%
340 {\cc@fallback@anchor}%
341 %% leaving this will generate lots of "duplicate destination"
342 %% messages from pdfbackend
343 %\expandafter\hypertarget\expandafter{#1}{}%
344 \expandafter\label\expandafter{#1}%
345 }
346 \def\cc@fallback@anchor{\phantomsection}%

```

### 3.5 Linguistic Name generation and selection

`\ccSetBabelLabel` defined a language-dependent string macro for German and English varieties.

`{#1}` is the language

`{#2}` is the internal reference name

`{#3}` is the language specific label

```

347 \def\ccSetBabelLabel#1#2#3{%
348 \def\ccc@lang{#1}%
349 \expandafter\def\expandafter\ccc@tempa\expandafter{\expandafter\def\csname #2name\endcsname
350 {#3}}%
351 \ifdefstring\ccc@lang{german}{%
352 \expandafter\addto\expandafter\captionsgerman\expandafter{\ccc@tempa}%
353 \expandafter\addto\expandafter\captionsgerman\expandafter{\ccc@tempa}%
354 }\relax%
355 \ifdefstring\ccc@lang{english}{%
356 \expandafter\addto\expandafter\captionsbritish\expandafter{\ccc@tempa}%
357 \expandafter\addto\expandafter\captionsUKenglish\expandafter{\ccc@tempa}%
358 \expandafter\addto\expandafter\captionsenglish\expandafter{\ccc@tempa}%
359 \expandafter\addto\expandafter\captionsamerican\expandafter{\ccc@tempa}%
360 \expandafter\addto\expandafter\captionsUSenglish\expandafter{\ccc@tempa}%
361 }\relax%
362 }

```

### 3.6 Link Generation

`\ccCompLink` creates a hyperlink with the target taken from Component with the name `{#1}` and the label `{#2}`.

```
362 \def\ccCompLink#1#2{%
363   \protected@edef\@argi{\expandonce{\ccUseComp{#1}}}%
364   \expandafter\href\expandafter{\@argi}{#2}%
365 }
```

`\ccPageLabel` enables referencing pages via `\pageref` by using `\phantomsection` to create a hyperref anchor for label `{#1}`.

```
366 \def\ccPageLabel#1{\phantomsection\label{#1}}
```

```
</common>
```



## Module 4

# coco-accessibility.dtx

---

This file provides code for the interaction between the CoCoTeX framework and the `ltpdfa` package.

**Please consider this module as highly experimental!**

There are two files created from this dtx: one `coco-accessibility.sty` and one `coco-accessibility.lua`.

## 1 LaTeX code

```
<*ally-sty>
```

### 1.1 General Processing

The `coco-accessibility.sty` starts with some general package information like name, current version and date of last changes.

```

23 %%
24 %% Accessibility features for \textit{xerif} projects.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2018
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-accessibility}
32   [2024/07/16 0.5.0 CoCoTeX accessibility module]
33 \RequirePackage{kvoptions-patch}
34 \RequirePackage{xkeyval}
35 \RequirePackage{atbegshi}
36 \RequirePackage{xparse}

```

The `ltpdfa` package re-defines too many standard LaTeX macros, so we only use its lua code and define the interface ourselves. For that, we use `etoolbox`'s patch commands to inject our tagging code into the standard macros rather than to create hard copies. This should increase compatibility with other packages and make all our lives easier.

We start with adopting `ltpdfa`'s package options.

`\cca@lang@id` is the ISO 639-2 code for the document's main language. As default, we assume Modern English.

```

37 \def\cca@lang@id{eng}%
38 \DeclareOptionX{lang-id}{\gdef\cca@lang@id{#1}}

39 \DeclareOptionX{init}{\global\let\cc@do@ally\relax}

```

`\cca@do@nodetree` if `\relax`, show the node tree in the log and in the shell output.

```
40 \DeclareOptionX{nodetree}{\let\cca@do@nodetree\relax}
```

`\cca@do@showspaces` if `\relax`, show spaces in the pdf.

```
41 \DeclareOptionX{show-spaces}{\let\cca@do@showspaces\relax}
```

`\cca@do@dospaces` if `\relax`, add ASCII space characters to the PDF. L<sup>A</sup>T<sub>E</sub>X doesn't write physical spaces into the output document but moves letters via skips, which allows variable word spacing beyond a font's space width definition, but it is a hard barrier for screen readers which rely on real space characters. This options causes the `ltpdfa` package to insert real space characters that are immediately followed by a negative skip by the font-dependent width of that space to keep L<sup>A</sup>T<sub>E</sub>X's typeface intact. This is activated by default.

```
42 \let\cca@do@dospaces\relax
43 \DeclareOptionX{no-spaces}{\let\cca@do@dospaces\@undefined}
```

`\cca@do@doparas` if `\relax`, add paragraph tagging.

```
44 \let\cca@do@doparas\relax
45 \DeclareOptionX{no-paras}{\let\cca@do@doparas\@undefined}
```

Processing the options.

```
46 \ProcessOptionsX
```

`\cca@patch@error` is a generic error message that is thrown whenever a L<sup>A</sup>T<sub>E</sub>X kernel macro could not be patched. This is usually the case when the macro definition does not match coco-accessibility's expectation, e.g., when another package messes with the macro's original definition. #1 is the CS token of the un-patchable macro.

```
47 \def\cca@patch@error#1{%
48   \ccPackageError{a11y}{compatibility}
49   {Could not patch \noexpand#1}
50   {You probably use a LaTeX package that re-defines the \noexpand#1 control sequence. It is
    apparently not compatbile with coco-accessibility.sty. Sorry}}
```

## 1.2 Activating and Deactivating Accessibility Features

`\ccIfAlly` is a switch to distinct between compilation with (implicit #1) or without (implicit #2) activated accessibility features.

```
51 \def\cc@if@ally{\ifx\cc@do@ally\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
52 \let\ccIfAlly\cc@if@ally
```

`\ccWhenAlly` is a variant of `\ccIfAlly` that omits the else branch.

```
53 \def\ccWhenAlly{\ifx\cc@do@ally\relax\expandafter\@firstofone\else\expandafter\@gobble\fi}
```

`\ccUnlessAlly` is a varant of `\ccIfAlly` that omits the then branch.

```
54 \def\ccUnlessAlly{\ifx\cc@do@ally\relax\expandafter\@gobble\else\expandafter\@firstofone\fi}
```

## 1.3 Accessibility-specific additions

### Loading Further Dependencies

Activated coco-accessibility requires two packages: `luatexbase-attr` (possibly deprecated?) provides an interface to add attributes to lua code; `atveryend` provides a hook to inject code to the final stages of PDF rendering.

```

55 \ccWhenAllly{%
56   \ifluatex\else
57     \ccPackageError{a11y}{engine}
58       {accessibility features require luatex!}
59       {You tried to use the accessibility features of CoCoTeX with an other TeX engine than
        luatex. This will not work; luatex is a hard requirement. Sorry.}
60   \fi
61   \RequirePackage{luatexbase-attr}
62   \RequirePackage{atveryend}

```

### Additional Hyperref Setup

Additional hyperref setup to be executed at the very end of the preamble.

```

63 \AtBeginDocument{%
64   \hypersetup{%
65     % pdfa=true% already set elsewhere
66     ,unicode=true%
67     ,pdfinfo={}%
68     % ,pdfpagelabels=true% already set elsewhere
69     ,pageanchor=true%
70   }%
71   \Hy@pdfatru
72 }

```

### Loading and Configuring ltpdfa's Lua Modules

Now, we set the configuration of the `ltpdfa` lua facility by passing some of the `coco-accessibility` package options:

```

73 \directlua{ltpdfa = require('ltpdfa')}
74 \directlua{ltpdfa.config.final = true}
75 \ifinlist{ltpdfa}\debug@domain@list
76   {\directlua{ltpdfa.config.debug = true}}
77   {\directlua{ltpdfa.config.debug = false}}
78 \directlua{ltpdfa.config.nodetree = \ifx\cca@do@nodetree\relax true\else false\fi}
79 \directlua{ltpdfa.config.showspace = \ifx\cca@do@spaces\relax true\else false\fi}
80 \directlua{ltpdfa.config.dospaces = \ifx\cca@do@dospaces\relax true\else false\fi}
81 \directlua{ltpdfa.config.doparas = \ifx\cca@do@doparas\relax true\else false\fi}

```

`ltpdfa` provides two ways to tag heading heads. One by tagging headers as `H1..H6`, and one where all headings are tagged as `H` and a heading's depth is implied by nesting. Since most of our projects require way more than 6 heading levels, we hard-code the nesting approach:

```

82 \directlua{ltpdfa.config.headnums = false}

```

CoCoTeX with accessibility support is `\luaTeX` only, so we hard-code `pdfTeX` as render engine:

```

83 \directlua{ltpdfa.config.driver = "\luaescapestring{pdfTeX}"}
84 \directlua{ltpdfa.config.lang = '\luaescapestring{\cca@lang{id}'} }
85 \directlua{ltpdfa.init()}%

```

Initial setup of `ltpdfa`

```

86 \edef\@ltpdfa@pattr{\directlua{ltpdfa.getAttribute('\luaescapestring{parentattr}')}}}
87 \edef\@ltpdfa@tattr{\directlua{ltpdfa.getAttribute('\luaescapestring{typeattr}')}}}
88 \attributedef\@ltpdfa@typeattr=\@ltpdfa@tattr
89 \attributedef\@ltpdfa@parentattr=\@ltpdfa@pattr
90 \def\ltpdfa@last@page{\ifx\r@LTLastPage\undefined\@empty\else\expandafter\@secondoftwo\
    r@LTLastPage\fi}%

```

We need the absolute last page of the document

```

91 \AfterLastShipout{\immediate\write\@mainaux{\string\newlabel{LTLastPage}{\LTLastPage}{\
    directlua{ltpdfa.getPageNum()}}}}}%
92 }%</ccWhenAlly

```

## 1.4 Generic Macro to Declare Accessibility Features

In order to selectively enable and (temporarily) disable accessibility macros during runtime, we need each tagging markup macro to exist in three states: one where they trigger tagging into the pdf, one where they do nothing, and one where they can be rescued for later expansion.

The enabled, disabled and protected versions of each macro are stored inside three separate lists:

`\cca@relaxed@defs` is the list that stores the *disabled* ltpdfa interface command variants,

```

93 \def\cca@relaxed@defs{}

```

`\cca@saved@defs` is a list that stores the *enabled* ltpdfa interface command variants.

```

94 \def\cca@saved@defs{}

```

`\cca@protected@defs` stores the protected macros where they do nothing when called, but also are not removed from the token list.

```

95 \def\cca@protected@defs{}

```

The next three macros are used to disable, enable, and protect accessibility markup locally, iff accessibility features are globally activated:

`\ccaDisable` disables all ltpdfa commands

```

96 \def\ccaDisable{\ccWhenAlly{\cca@relaxed@defs}}

```

and

`\ccaEnable` enables all ltpdfa commands.

```

97 \def\ccaEnable{\ccWhenAlly{\cca@saved@defs}}

```

`\ccaProtect` protects accessibility commands such that they expand to itself (useful when material containing tagging macros are stored in a macro using `\protected@edef`.)

```

98 \def\ccaProtect{\ccWhenAlly{\cca@protected@defs}}

```



`\CsToStr` is a xparse helper macro which returns the name of a control sequence #1.

```
99 \ExplSyntaxOn
100 \newcommand{\CsToStr}[1]{\cs_to_str:N #1}
101 \ExplSyntaxOff
```

`\DeclareAccessibilityCommand` is the wrapper for our interface macros. It has the same argument signature as L<sup>A</sup>T<sub>E</sub>X's `\newcommand*`, albeit without the whole checking for already defined control sequences.

```
102 \def\DeclareAccessibilityCommand#1{\ifnextchar[{\cca@declare@cmd@firstopt#1}{\cca@declare@cmd
    #1}}}%]
```

First, we need to take care of the optional arguments:

`\cca@temp@signature` is the temporary storage for the argument signature.

```
103 \let\cca@temp@signature\empty
```

`\cca@declare@cmd@firstopt` is the handler for the first optional argument, which holds the overall number of the arguments of our interface macro:

```
104 \def\cca@declare@cmd@firstopt#1[#2]{\edef\cca@temp@signature{[\unexpanded{#2}]}%
105 \ifnextchar[{\cca@declare@cmd@secopt#1}{\cca@declare@cmd#1}}}%]
```

`\cca@declare@cmd@secopt` is the handler for the second optional argument, which indicates that the first of the first-level arguments is optional and which itself holds the default value for that optional argument. Its unexpanded value is added to the argument signature.

```
106 \def\cca@declare@cmd@secopt#1[#2]{\eappto\cca@temp@signature{[\unexpanded{#2}]} \cca@declare@cmd
    #1}
```

`\cca@declare@cmd`, eventually, is the actual wrapper for the newcommand calls.

```
107 \def\cca@declare@cmd#1#2{%
```

First, we create a string `\savedDef` that includes the *active* definition of our interface macro and store it in an internal macro named `\cc@saved@#1`. This macro is immediately called.

```
108 \edef\savedDef{\noexpand\newcommand*\expandafter\noexpand\csname cc@saved@\CsToStr{#1}\
    endcsname\expandonce{\cca@temp@signature}{\unexpanded{#2}}}\savedDef%
```

Then, we create a `\let` sequence that maps the plain CS name #1 onto that newly created internal macro. The String containing the let-sequence is then stored in the `\cca@saved@defs` list, so whenever this list is expanded, the desired CS-token “#1” is defined to the active definition.

```
109 \edef\x{\noexpand\let\noexpand#1\expandafter\noexpand\csname cc@saved@\CsToStr{#1}\endcsname}%
110 \global\expandafter\appto\expandafter\cca@saved@defs\expandafter{x}%
```

Then, we repeat the same procedure, but this time, we define the whole internal CS token with the same argument structure to expand to `\relax`.

```
111 \edef\relaxDef{\noexpand\newcommand*\expandafter\noexpand\csname cc@no@\CsToStr{#1}\endcsname\
    expandonce{\cca@temp@signature}{\relax}}\relaxDef%
```

The whole `\let` sequence for the `\relax` version of our internal macro is then stored in the `\cca@relaxed@defs` list.

```
112 \edef\y{\noexpand\let\noexpand#1\expandafter\noexpand\csname cc@no@\CsToStr{#1}\endcsname}%
113 \global\expandafter\appto\expandafter\cca@relaxed@defs\expandafter{y}%
```

Now, we can decide which of the two `\let`-sequences should be the used to define the initial value of the #1 CS token, depending on the value of the `\ccIfAlly` conditional:

```
114 \ccIfAlly{\x}{\y}%
```

Eventually, we define the protected version of the CS token `{#1}` which expands to itself, and thus is protected from expansion inside `\edef`, et al.

```
115 \edef\z{\noexpand\protected\noexpand\def\noexpand#1{\noexpand#1}}%
116 \global\expandafter\appto\expandafter\cca@protected@defs\expandafter{\z}%
```

Finally, we reset the macro that contained the argument signature.

```
117 \let\cca@temp@signature\@empty
118 }
```

Some macros from `ltpdfa.sty`:

```
119 \DeclareAccessibilityCommand{\ccaAddToConfig}[2]{\directlua{ltpdfa.addToConfig('\luaescapestring
{#1}', '\luaescapestring{#2}')}}
120 \DeclareAccessibilityCommand{\ccaAddKeyword}[1]{\directlua{ltpdfa.tagger.addToKeywords('\
luaescapestring{#1}')}}
121 \DeclareAccessibilityCommand{\ccaAddAuthor}[1]{\directlua{ltpdfa.tagger.addToAuthors('\
luaescapestring{#1}')}}
122 \@onlypreamble\ccaAddToConfig
```

`\ccaStructStart` inserts a structural tag with the name #2. Optional #1 is the name of a forced parent.

This tagging macro inserts `\bgroup` at the start of the tagged area.

```
123 \DeclareAccessibilityCommand{\ccaStructStart}[2][\ifcc@is@final\directlua{ltpdfa.tagger.
structStart('\luaescapestring{#2}', '\luaescapestring{#1}'))}\fi}
```

`\ccaStructEnd` inserts the an `\egroup` and an end tag with the name #1.

```
124 \DeclareAccessibilityCommand{\ccaStructEnd}[1]{\ifcc@is@final\directlua{ltpdfa.tagger.structEnd
('\luaescapestring{#1}'))}\fi}
```

`\ccaVstructStart` is the same as `\ccaStructStart`, but without inserting a group at the beginning of the tagging area.

```
125 \DeclareAccessibilityCommand{\ccaVstructStart}[2][\ifcc@is@final\directlua{ltpdfa.tagger.
vstructStart('\luaescapestring{#2}', '\luaescapestring{#1}'))}\fi}
```

`\ccaVstructEnd` ends an ungrouped tagging area. #1 is the name of the tag.

```
126 \DeclareAccessibilityCommand{\ccaVstructEnd}[1]{\ifcc@is@final\directlua{ltpdfa.tagger.
vstructEnd('\luaescapestring{#1}'))}\fi}
```

`\ccaPstructStart` is the same as `\ccaStructStart` but no grouping and no setting of any attributes applies. Implies that the element has no content children, at all.

```
127 \DeclareAccessibilityCommand{\ccaPstructStart}[2][\directlua{ltpdfa.tagger.pstructStart('\
luaescapestring{#2}', '\luaescapestring{#1}'))}
```

`\ccaPstructEnd` ends an unattributed tagging area.

```
128 \DeclareAccessibilityCommand{\ccaPstructEnd}[1]{\directlua{ltpdfa.tagger.pstructEnd('\luaescapestring{#1}')}}}
```

`\ccaGetCurStruct` returns the internal ID of the currently open structural element. #1 is table attribute that should be returned. The following code gives an example on how to use the macro:

```
\ccaStructStart{Leela}
\edef\LeelaID{\ccaGetCurStruct{idx}}%
\ccaStructEnd{Leela}
```

This stores the internal node index of the `Leela` tag node in the `\LeelaID` macro so it can be referenced by other lua interface macros like `\ccaAddToStruct` or `\ccaMoveStruct`, as shown below.

```
129 \DeclareAccessibilityCommand{\ccaGetCurStruct}[1]{\directlua{ltpdfa.tagger.getCurrentStruct('\luaescapestring{#1}')}}}
130 \DeclareAccessibilityCommand{\ccaSaveCurStruct}[1]{\protected@csxdef{#1}{\ccaGetCurStruct{idx}}}
```

`\ccaAddToStruct` adds the current structural element to the structural element #1 previously retrieved using `\ccaGetCurStruct`, e.g.,

```
% \ccaStructStart{Fry}
% \edef\FryID{\ccaGetCurStruct{idx}}%
% \ccaStructEnd{Fry}
% \ccaStructStart{Hubert}
% \ccaAddToStruct{\FryID}%
% \ccaStructEnd{Hubert}
```

makes `Hubert` into a child node of `Fry` and detaches it from its current parent node (which, in this case, is also the current parent of `Foo`). Note that the parent has to be tagged *before* the child node.

```
131 \DeclareAccessibilityCommand{\ccaAddToStruct}[1]{\directlua{ltpdfa.tagger.addToStruct('\luaescapestring{#1}')}}}
```

`\ccaMoveStruct` removes the Node with the ID #1 from its current parent and attaches it as child to the current node. `\ccaMoveStruct` is the logical counter-part of above's `\ccaAddToStruct`. The child's node ID can be retrieved with the `\ccaGetCurrentStruct` command, for example:

```
\ccaStructStart{Hubert}
\edef\HubertID{\ccaGetCurStruct{idx}}
\ccaStructEnd{Hubert}
\structStart{Fry}
\ccaMoveStruct{\HubertID}
\structEnd{Fry}
```

This will make `Hubert` a child of `Fry`. In contrast to `\ccaAddToStruct`, this allows to attach a previously tagged child node to a later tagged parent node.

```
132 \DeclareAccessibilityCommand{\ccaMoveStruct}[1]{\relax\directlua{ltpdfa.tagger.moveStruct('\luaescapestring{#1}')}}}
```

`\ccaReplaceStruct` takes a previously added tag node with the index #1 and *replaces* it with the current tag node.

```
133 \DeclareAccessibilityCommand{\ccaReplaceStruct}[1]{\relax\directlua{ltpdfa.tagger.replaceStruct('\luaescapestring{#1}')}}}
```

`\ccaAddID` renames the index attribute of the current tag node to #1. If #1 is “auto”, the index is calculated by ltpdfa.

```
134 \DeclareAccessibilityCommand{\ccaAddID}[1]{\directlua{ltpdfa.tagger.addID('\luaescapestring
    {#1}')}}}
```

`\cca@set@docinfo` sets the PDF docinfo. #2 is a key, #3 is the value, optional #1 is an encoding.

```
135 \DeclareAccessibilityCommand{\ccaSetDocinfo}[3][\directlua{ltpdfa.setDocInfo('\luaescapestring
    {#2}', '\luaescapestring{#3}', '\luaescapestring{#1}')}}}
```

`\ccaAddRolemap` is used to map a custom LaTeX tag to a well-defined PDF tag. #1 is the name of the LaTeX Tag, #2 is the name of the PDF role.

```
136 \DeclareAccessibilityCommand{\ccaAddRolemap}[2]{\directlua{ltpdfa.tagger.addRolemap('\
    luaescapestring{#1}', '\luaescapestring{#2}')}}}
```

`\ccaAddPlacement` tells the tagger if a floating object is placed as a “Block” or “Inline”.

```
137 \DeclareAccessibilityCommand{\ccaAddPlacement}[1]{\directlua{ltpdfa.tagger.addPlacement('\
    luaescapestring{#1}')}}}
```

`\ccaAddNumbering` ???

```
138 \DeclareAccessibilityCommand{\ccaAddNumbering}[1]{\directlua{ltpdfa.tagger.addNumbering('\
    luaescapestring{#1}')}}}
```

## 1.5 Lua injection

Some features are realized by Lua code, so we tell LuaLaTeX to include the code that is generated from material later in this source file:

```
139 \ccWhenAlly{\directlua{ally = require('coco-accessibility')}}}
```

## 1.6 Hyperlink handling

To tag hyperlinks, we define some ltpdfa interface macros.

`\ccaAddAltText` is used to add an Alternative Text node, given in #1, to the PDF structTree.

```
140 \DeclareAccessibilityCommand{\ccaAddAltText}[1]{\directlua{ltpdfa.tagger.addAltText('\
    luaescapestring{#1}')}}}
```

`\cca@Gin@alt` is the captured value of the `alt` key from the optional argument of the `\includegraphics` command. This can be used to pass its value to the `\ccaAddAltText` macro defined above.

```
141 \define@key{Gin}{alt}{\gdef\cca@Gin@alt{#1}}
```

`\ccaAddLastLink` adds the last Link node to the PDF structTree.

```
142 \DeclareAccessibilityCommand{\ccaAddLastLink}{\directlua{ltpdfa.tagger.addLastLink()}}
```

`\ccaGetStructParent` returns the current parent structure. This is needed in case a link breaks across columns (or pages).

```
143 \DeclareAccessibilityCommand{\ccaGetStructParent}{\directlua{lua.tagger.getStructParent()}}
```

We prepare the link interface macros to be patched into `hyperref` at the begin document hook if accessibility features are activated.

First we add the start tag for a Link node.

```
144 \begingroup
145 \makeoother\#
146 \ccWhenAlly{%
147 \AtBeginDocument{%
148   \patchcmd\Hy@StartlinkName
149     {\pdfstartlink}
150     {\pdfstringdef\@argii{#2}\ccaStructStart{Link}\ccaAddAltText{#2}\edef\@ltpdfmy@parent{\
151       \pdfstartlink}
152     }{\cca@patch@error\Hy@StartlinkName}
```

and the parent node inside the link attribute:

```
153   \patchcmd\Hy@StartlinkName
154     {#1}
155     {#1 /StructParent \@ltpdfmy@parent /Contents(\@argii)}
156     {}{\cca@patch@error\Hy@StartlinkName}
```

then we patch `hyperref`'s general link macro, twice. Once for the Link's start tag

```
157   \patchcmd\hyper@linkurl
158     {\pdfstartlink}
159     {\pdfstringdef\@argii{#2}\ccaStructStart{Link}\ccaAddAltText{#2}\edef\@ltpdfmy@parent{\
160       \pdfstartlink}
161     {}{\cca@patch@error\hyper@linkurl}
```

and secondly for the Parent:

```
162   \patchcmd\hyper@linkurl
163     {/C[\@urlbordercolor]%
164     \fi
165     }
166     {/C[\@urlbordercolor]%
167     \fi
168     /StructParent \@ltpdfmy@parent%
169     /Contents(\@argii)
170     }{\cca@patch@error\hyper@linkurl}
171   % \patchcmd\hyper@linkurl
172   % {\Hy@href@nextactionraw}
173   % {
174   % \Hy@href@nextactionraw}
175   % {}{\cca@patch@error\hyper@linkurl}
```

finally, we patch the end tag for the link node into the `\close@pdflink` macro:

```
176   \patchcmd\close@pdflink
177     {\pdfendlink}
178     {\pdfendlink
179     \ccaAddLastLink\ccaStructEnd{Link}}
180     {}{\cca@patch@error\close@pdflink}
```

For internal references, we patch the tagging into the `\@setref` macro. Unfortunately, `hyperref` redefines this macro and links to both the original version (when `\ref*` is used), and its own re-definition (else), so we need to patch both versions. We start by resetting `\@setref` to its vanilla state and inject our tagging, once for the start tag and a second time for the end tag:

```

181 \let\cca@hy@setref\@setref
182 \let\@setref\real@setref
183 \patchcmd\@setref
184   {\else}
185   {\else\ccaStructStart{Reference}}
186   {}{\cca@patch@error\orig@setref@new}%
187 \patchcmd\@setref
188   {\fi}
189   {\ccaStructEnd{Reference}\fi}
190   {}{\cca@patch@error\orig@setref@new}%

```

Now, we restore `hyperref`'s version and inject the tagging there as well:

```

191 \let\real@setref\@setref
192 \let\@setref\cca@hy@setref
193 \patchcmd\@setref
194   {\expandafter\Hy@setref@link}
195   {\ccaStructStart{Reference}\expandafter\Hy@setref@link}
196   {}{\cca@patch@error\@setref}
197 \patchcmd\@setref
198   {{#2}}
199   {{#2}\ccaStructEnd{Reference}}
200   {}{\cca@patch@error\@setref}
201 }% /AtBeginDocument
202 }% /ccWhenAlly
203 \endgroup

```

## 1.7 Tagging Page Styles as Artifacts

Page styles, i.e., headers and footers, need to be tagged as artifacts unless they contain semantic information. To avoid inserting the tagging by hand into each publisher's page style definitions, we inject the tagging automatically by using `etoolbox`'s patch commands to insert the start and end tags inside the internal header and footer macros, respectively.

`\ccaPagestyleArtifacts` contains the code to patch the `\@oddhead`, `\@evenhead`, `\@oddfoot` and `\@evenfoot` macros.

```

204 \DeclareAccessibilityCommand{\ccaPagestyleArtifacts}{%
205   \ifx\@oddhead\@empty\else
206     \pretocmd\@oddhead{\ccaStructStart[document]{header}}{}{}%
207     \apptocmd\@oddhead{\ccaStructEnd{header}}{}{}%
208   \fi
209   \ifx\@evenhead\@empty\else
210     \pretocmd\@evenhead{\ccaStructStart[document]{header}}{}{}%
211     \apptocmd\@evenhead{\ccaStructEnd{header}}{}{}%
212   \fi
213   \ifx\@oddfoot\@empty\else
214     \pretocmd\@oddfoot{\ccaStructStart[document]{footer}}{}{}%
215     \apptocmd\@oddfoot{\ccaStructEnd{footer}}{}{}%
216   \fi
217   \ifx\@evenfoot\@empty\else
218     \pretocmd\@evenfoot{\ccaStructStart[document]{footer}}{}{}%
219     \apptocmd\@evenfoot{\ccaStructEnd{footer}}{}{}%

```

```
220 \fi}
```

The standard pagestyles from the L<sup>A</sup>T<sub>E</sub>X kernel are patched by the module.

```
221 \apptocmd\ps@empty{\ccaPagestyleArtifacts}{}{}
222 \apptocmd\ps@plain{\ccaPagestyleArtifacts}{}{}
223 \apptocmd\ps@headings{\ccaPagestyleArtifacts}{}{}
224 \apptocmd\ps@myheadings{\ccaPagestyleArtifacts}{}{}

```

Finally, we register the `footer` and `header` PDF tags as `artifacts` with `ltpdfa`:

```
225 \ccWhenAlly{%
226   \ccaAddToConfig{artifact}{header={Type:Pageination}{Subtype:Header}}
227   \ccaAddToConfig{artifact}{footer={Type:Pageination}{Subtype:Footer}}

```

## 1.8 generic artifacts

```
228 \ccaAddToConfig{artifact}{leaders={Type:Layout}}
229 \ccaAddToConfig{artifact}{footnoterule={Type:Layout}}
230 \ccaAddToConfig{artifact}{Rule={Type:Layout}}
231 \ccaAddToConfig{artifact}{Artifact={Type:Layout}}
232 }

```

`\ccaArtifact` starts an Artifact environment within which all Tagging is disabled.

```
233 \def\ccaArtifact{\ccaStructStart[document]{Artifact}\ccaDisable}

```

`\endccaArtifact` ends an Artifact environment.

```
234 \def\endccaArtifact{\ccaEnable\ccaStructEnd{Artifact}}

```

## 1.9 Tagging for Floats

### Taggin for Figures

`\ccaAddFigure` #1, #2, #3, and #4 are the x and y coordinates of the image, first x and y of the lower left corner, then x and y of the upper right corner; #5 and #6 are the *x* and *y* scales, respectively; and #7 is “true” or “false” depending on whether or not the clipping option is active.

```
235 \DeclareAccessibilityCommand{\ccaAddFigure}[7]{\directlua{ltpdfa.tagger.addFigure(
236   '\luaescapestring{#1}',
237   '\luaescapestring{#2}',
238   '\luaescapestring{#3}',
239   '\luaescapestring{#4}',
240   '\luaescapestring{#5}',
241   '\luaescapestring{#6}',
242   '\luaescapestring{#7}')}}

```

`\ccaFigureStart` injects the starting tag for images to the pdf

```
243 \DeclareAccessibilityCommand{\ccaFigureStart}[1]{\directlua{ltpdfa.tagger.figureStart('\luaescapestring{#1}')}}

```

`\ccaFigureEnd` injects the ending tag for images

```
244 \DeclareAccessibilityCommand{\ccaFigureEnd}[1]{\directlua{ltpdfa.tagger.figureEnd('\luaescapestring{#1}')}}}
```

which we add to the beginning and the end of `graphics` package's `\Gininclude@graphics` macro, respectively.

```
245 \AtBeginDocument{%
246   \let\ltx@Gincludel@graphics\Gininclude@graphics
247   \def\Gininclude@graphics#1{\ifcc@is@final\ccaFigureStart{}\fi\ltx@Gincludel@graphics{#1}\ifcc@is@final\ccaFigureEnd{}\fi}%
248 }
```

```
249 \apptocmd\Gininclude@pdfptex{\ifcc@is@final%
250   \def\@tempa{!}%
251   \ccaAddFigure{\Gin@llx}{\Gin@lly}{\Gin@urx}{\Gin@ury}
252   {\ifx\Gin@scalex\@tempa\else \Gin@scalex\fi}
253   {\ifx\Gin@scaley\@tempa\else \Gin@scaley\fi}
254   {\ifGin@clip true\else false\fi}\fi}%rwi/rhi
255   {}{}
256 \AtBeginDocument{%
257   \@ifpackageloaded{htmltabs}{%
258     \let\ltx@ht@valign@box\ht@valign@box
259     \def\ht@valign@box{\if@ht@final@render\cc@is@finaltrue\fi\ltx@ht@valign@box}
260     \let\ltx@ht@RenderCell\ht@RenderCell
261     \def\ltx@ht@RenderCell{\@cc@is@finalfalse\ltx@ht@RenderCell}}{}}
```

## Tagging for Tables

`\ccaAddScope` is used to indicate the scope of in table's head cells. The value should be either `Column` or `Row`.

```
262 \DeclareAccessibilityCommand{\ccaAddScope}[1]{\relax\directlua{ltpdfa.tagger.addScope('\luaescapestring{#1}')}}}
```

`\ccaAddColSpan` is used to mark a cell to span horizontally over #1 columns (including it's own).

```
263 \DeclareAccessibilityCommand{\ccaAddColSpan}[1]{\relax\directlua{ltpdfa.tagger.addColSpan('\luaescapestring{#1}')}}}
```

`\ccaAddRowSpan` is used to mark a cell to span vertically over #1 rows (including it's own).

```
264 \DeclareAccessibilityCommand{\ccaAddRowSpan}[1]{\relax\directlua{ltpdfa.tagger.addRowSpan('\luaescapestring{#1}')}}}
```

`\ccaAddKeep` is inserted into empty Tags to tell the ltpdfa-tagger to not remove the Tag even if it may be empty.

```
265 \DeclareAccessibilityCommand{\ccaAddKeep}{\relax\directlua{ltpdfa.tagger.addKeep()}}
```

## 1.10 Transformation of Typographic Unicode characters

In order for screen readers to work correctly, some unicode characters that mask purely typographic glyphs (e.g., ligatures) need to be mapped to their underlaying orthographic characters. This is done via pdfTeX's `glyphtounicode` tables:

```
266 \ifx\pdfextension\@undefined\else
267 \protected\def\pdfglyphtounicode{\pdfextension glyphtounicode}
```



```

268 \input glyphtounicode
269 \edef\pdfgentounicode{\pdfvariable gentounicode}
270 \pdfgentounicode = 1
271 \fi

```

`\ccaConvertPdfString` takes a `utf-16` string as `{#1}` and converts it to `utf-8`. This is intended to resolve octal tokens in the output of `hyperref`'s `\pdfstringdef`, which `ltpdfa`'s `ltpdfa.setDocInfo()` method does not seem to handle well.

```

272 \DeclareAccessibilityCommand{\ccaConvertPdfString}[1]{\directlua{tex.print(ltpdfa.metadata.utf16
    ToUtf8('\luaescapestring{#1}'))}}

```

## 1.11 Automatic PDF Tagging

### Document Root Node

The following code causes the `ltpdfa` package to tag the `document` environment as the structural representation's root node:

```

273 \ccWhenAlly{%

```

`cca/document/begin` is a hook to add accessibility specific declarations right before begin document.

```

274 \NewHook{cca/document/begin}%
275 \AtBeginDocument{%
276   \directlua{ltpdfa.beginDocument('\luaescapestring{ltpdfa@last@page}')}
277   \UseHook{cca/document/begin}%
278   \directlua{ltpdfa.configAutoclose()}
279   \ccaVstructStart{document}%

```

`\PDF@FinishDoc` is re-defined to disable `hyperref` filling the PDF's DocumentInfo with empty strings, as they are written by `ltpdfa`, instead.

```

280 \renewcommand\PDF@FinishDoc{%
281   \Hy@DisableOption{pdfauthor}%
282   \Hy@DisableOption{pdftitle}%
283   \Hy@DisableOption{pdfsubject}%
284   \Hy@DisableOption{pdfcreator}%
285   \Hy@DisableOption{pdfcreationdate}%
286   \Hy@DisableOption{pdfmoddate}%
287   \Hy@DisableOption{pdfproducer}%
288   \Hy@DisableOption{pdfkeywords}%
289   \Hy@DisableOption{pdftrapped}%
290   \Hy@DisableOption{pdfinfo}%
291 }%
292 }
293 \AtEndDocument{%
294   \ccaVstructEnd{document}
295   \directlua{ltpdfa.endDocument()}%
296 }
297 }

```

## 1.12 Math Tagging

```

298 \AtBeginDocument{%
299   \apptocmd{\({}\}{\ccaStructStart{Formula}}{}{\cca@patch@error\{}}
300   \pretocmd{\)}{\ccaStructEnd{Formula}}{}{\cca@patch@error\}}
301   \apptocmd{\[{}]{\ccaVstructStart{Para}\ccaStructStart{Formula}}{}{\cca@patch@error\[{}]}
302   \pretocmd{\]}{\ccaStructEnd{Formula}\ccaVstructEnd{Para}}{}{\cca@patch@error\]}
303   \pretocmd{\start@align}{\ccaVstructStart{Para}\ccaStructStart{Formula}}{}{\cca@patch@error\
304     start@align}%
305   \apptocmd{\endalign}{\ccaStructEnd{Formula}\ccaVstructEnd{Para}}{}{\cca@patch@error\endalign}%
306 }

```

### 1.13 Default Role Mapping

Note that this section contains only the role mappings that didn't thematically fit into other CoCoT<sub>E</sub>X modules.

```

306 \ccaAddRolemap{document}{Document}
307 \ccaAddRolemap{Para}{P}

```

Finally, we hook `ltpdfa`'s page processor into `AtBeginShipoutBox`:

```

308 \ccWhenAlly{\AtBeginShipout{\directlua{ltpdfa.pageprocessor(tex.box["AtBeginShipoutBox"])}}}%

```

End of T<sub>E</sub>X source code.

```
</a11y-sty>
```

## 2 Lua code

```
<*a11y-lua>
```

### 2.1 Local Variables and Tables

`ltpdfa` is an instance of the `ltpdfa` Lua table.

```

311 local ltpdfa = require('ltpdfa')

```

### 2.2 Meta Data Extraction

`meta` is a table that holds the metadata that are extracted from the `\jobname.xmp` file via its `extract` member.

```

312 meta = {
313   Author = '',
314   Title = '',
315   Creator = '',
316   Producer = '',
317   Keywords = '',

```

The method `meta.extract()` reads the meta data from the `\jobname.xmp` and stores certain values to be accessed by LaTeX. This is used to fill the DocumentInfo when a xmp file is available during the expansion of `\cct@writepdf@meta` from the `coco-title` module (see Sect. 2).

```

318 extract = function(self)
319     local xmpfile = ltpdfa.metadata.xmphandler.fromFile(ltpdfa.config.metadata.xmpfile)
320     local f = io.open(xmpfile, "rb")
321     local content = f:read("*all")
322     f:close()

```

First, we extract the document title.

```

323     if (content:find('<dc:title>')) then
324         local title = content:gsub('.*<dc:title>[~]*<rdf:Alt>[~]*<rdf:li>[~]*>(.)</rdf:li>
325         >[~]*</rdf:Alt>[~]*</dc:title>.*', "%1")
326         self.Title = title
327     end

```

Then, we extract the authors from the `dc:creator` list.

```

327     local authors
328     local author = {}
329     if (content:find('<dc:creator>')) then
330         authors = content:gsub('.*<dc:creator>[~]*<rdf:Seq>(.)</rdf:Seq>[~]*</dc:creator>.*', "
331         %1")
332         for k in string.gmatch(authors, "<rdf:li>([~]+)</rdf:li>") do
333             table.insert(author, k)
334         end
335         self.Author = table.concat(author, '\\and ')
336     end

```

Then, we extract the keywords from the `dc:subject` list. If that doesn't exist, we try to extract the keywords from the `pdf:Keywords` Element, instead.

```

336     local keywords
337     local keyword = {}
338     if (content:find('dc:subject')) then
339         keywords = content:gsub('.*<dc:subject>[~]*<rdf:Bag>(.)</rdf:Bag>[~]*</dc:subject>.*', "
340         %1")
341         for k in string.gmatch(keywords, "<rdf:li>([~]+)</rdf:li>") do
342             table.insert(keyword, k)
343         end
344         self.Keywords = table.concat(keyword, '\\and ')
345     elseif (content:find('pdf:Keywords')) then
346         local keyword = content:gsub('.*<pdf:keywords>(.)</pdf:Keywords>.*', "%1")
347         self.Keywords = keyword
348     end

```

Then, we extract the PDF producer from the `pdf:Producer` element, if it exists:

```

348     if (content:find('<pdf:Producer>')) then
349         local prod = content:gsub('.*<pdf:Producer>(.)</pdf:Producer>.*', "%1")
350         self.Producer = prod
351     end

```

Finally, we extract the PDF CreatorTool from the `xmp:CreatorTool` element, if it exists:

```

352     if (content:find('<xmp:CreatorTool>')) then
353         local creatortool = content:gsub('.*<xmp:CreatorTool>(.)</xmp:CreatorTool>.*', "%1")
354         self.Creator = creatortool
355     end
356 end
357 }

```

## 2.3 Public Methods

`cocotex` is the base table that contains all public methods and sub-tables available in the CoCoTeX framework. Here, it is defined unless it is already defined elsewhere.

```
358 if type(cocotex) ~= 'table' then
359   cocotex = {}
360 end
```

`cocotex.ally` is a globally available namespace for coco-accessibility specific lua tables.

```
361 cocotex.ally = {
362   meta = meta
363 }
```

After loading `coco-accessibility.lua` via the `require()` method, a `cocotex.ally` table is returned.

```
364 return cocotex.ally
```

no more lua code.

```
</ally-lua>
```

## Module 5

# coco-meta.dtx

```
<*meta>
```

This file provides some macros that are used to process meta data, both for the whole document, as well as parts of a document.

File preamble

```
23 %%
24 %% module for CoCoTeX that provides handling of a document's meta data.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-meta}
32   [2024/07/16 0.5.0 CoCoTeX meta module]
33 \RequirePackage{coco-common}
```

**CommonMeta** is an abstract Container for commonly used meta data, both for whole documents as well as parts of documents.

```
34 \ccDeclareContainer{CommonMeta}{%
35   \ccDeclareType{Components}{%
36     \ccDeclareRole[author]{Author}%
37     \ccm@declare@comp
38     \ccm@extended@common@macros
39     \ccm@declare@affils
40   }%
41   \ccDeclareType{Properties}{}%
42 }
```

## 1 Counted Container Handlers

### 1.1 Generic Blocks

`\ccm@generic@comp` is used to define a generic meta data block. It provides two Components for each instance, one for the block's Heading or Label, and one for its Content.

```
43 \def\ccm@generic@comp{%
```

`CC` is the formatted list of all GenericMeta components.

```
44   \ccDeclareComponent{GenericMetaBlock}{\expandafter\global}{}%
```

```
45 \ccDeclareComponentGroup{GenericMeta}{%
```

`Heading` is the label of a generic meta datum

```
46 \ccDeclareCountedComponent{Heading}%
```

`Heading` is the content of a generic meta datum

```
47 \ccDeclareCountedComponent{Content}%
```

```
48 }}
```

`\ccm@generic@eval` evaluates the Components and tells the Framework how the generic counted Sub-Containers should be rendered.

```
49 \def\ccm@generic@eval{%
50   \def\cc@cur@cont{titlepage}%
51   \ccComposeCollection{GenericMeta}{generic-meta-format}{GenericMetaBlock}
52 }
```

## 1.2 Contributor Roles

Contributors are Counted Containers that represent the meta data of people that share a role in contributing content to a document. Examples for such roles are an article/chapter/book's authors, or a collection/series' editors.

## 1.3 Declaring Contributor Role Blocks

`\ccDeclareRoleBlock` is used to create a new Collection Container (named `\ccPrefix#2#3`) for a Role with the name `{#2}`. A Role Block is a Component of the Parent Container, which contains a formatted list of certain Components of all members of the Role. Format and selection of the utilised Components are specified via the Property given in `{#4}`. Role Blocks can also be directly used inside the Parent Container as Overrides.

The optional argument `[#1]` tells the evaluator in the Parent Container's `end` macro how the collector is to be composed. Valid values are `compose` (default), which uses `\ccComposeCollection` to compose the Collection Component, or `apply`, which uses `\ccApplyCollection`, instead.

```
53 \def\ccDeclareRoleBlock{\@ifnextchar[\cc@declare@role@block{\cc@declare@role@block[compose]}}%
54 \def\cc@declare@role@block[#1]#2#3#4{%
55   \ifcsdef{ccm@role@#1}
56     {\ccDeclareComponent{#2#3}{\expandafter\global}{}%
57     \csgdef{ccm@role@\cc@cur@cont @#2@#3}{#4}%
58     \csappto{ccm@role@eval@\cc@cur@cont @#2}
59     {\csname ccm@role@#1\endcsname{#2}{#3}}
60     {\ccPackageError{Meta}{Argument}
61     {Invalid optional argument in \string\ccDeclareRoleBlock!}
62     {Only 'apply' or 'compose' are allowed as values^^Jin the optional argument of \string\
        ccDeclareRoleBlock!}}}%

```

`\ccm@role@eval` creates the name lists for the role. `{#1}` is the name of the role.

```
63 \def\ccm@role@eval#1{\csname @ccm@role@eval@\cc@cur@cont @#1\endcsname}
```

`\ccm@eval@role` is the name of the macro used to compose the Collection (either `\ccComposeCollection`, or `\ccApplyCollection`)  
`{#2}` is the name of the role  
`{#3}` is the name of the list.

The access Component is `\ccPrefix #2#3`, i.e., the prefix and both arguments together.

```
64 \def\ccm@eval@role#1#2#3{%
```

First, we check if the Collection Component has already been set in the input. If so, we set an internal flag to indicate that the Collection Component has been filled manually.

```
65 \ccIfComp{#2#3}{\cslet{cc@used@#2#3@override}\@empty}{%
```

Second, we check if the counter for the Role is defined and greater than 0. If neither is the case, this means that the Group does not occur in the input, at all, so we don't need to do anything.

```
66 \ifcsdef{cc#2Cnt}  
67 {\expandafter\ifnum\csname cc#2Cnt\endcsname>\z@
```

otherwise, we call the Property that is stored in `\ccm@role@\cc@cur@cont @#2@#3` and store the result in the Component `#2#3`.

```
68 #1{#2}{\csname ccm@role@\cc@cur@cont @#2@#3\endcsname}{#2#3}%  
69 \fi  
70 }{}}}
```

`\ccm@role@apply` #1 is the name of the role and #2 is the name of the composition. This macro applies (i.e. *fully expands*) the `\ccm@role@\cc@cur@cont @#1@#2` Property and stores the result in the `#1#2` Component.

```
71 \def\ccm@role@apply#1#2{\ccm@eval@role\ccApplyCollection{#1}{#2}}
```

`\ccm@role@compose` #1 is the name of the role and #2 is the name of the composition. This stores the *unexpanded* contents of the `\ccm@role@\cc@cur@cont @#1@#2` Property in the `#1#2` Component.

```
72 \def\ccm@role@compose#1#2{\ccm@eval@role\ccComposeCollection{#1}{#2}}
```

## Declaring Contributor Roles

`\ccDeclareRole` is used to declare the Components that belong to each member of a contributor role.

`[#1]` is the internal name of the Role's formatting Property. If omitted, it is the same as `{#2}`  
`{#2}` is the name of the role

The output of all members of a role is controlled by a Component called “`<role>NameList`” that is formatted according to the `<role>-format` Property. For reasons of naming conventions, the role names for a Component and its respective Property do not necessarily need to be identical.

```
73 \def\ccDeclareRole{\cc@opt@second\cc@declare@role}%  
74 \def\cc@declare@role[#1]#2{%-  
75 \ccDeclareComponentGroup{%-  
76 {\ccDeclareAttributeHandler*{corresp}{\ccSetProperty{is-corresp}{true}}}%  
77 }{#2}{%-
```

## Contributor Role Components

`FullNameOR` is the full name of the Role member. If omitted, it is calculated by the `role-full-name-format` Property.

```
78 \ccDeclareCountedComponent{FullName}%
```

**CiteNameOR** is the Full Role member name that is used for citation advices. If omitted, it is calculated by the `role-cite-name-format` Property.

```
79 \ccDeclareCountedComponent{CiteName}%
```

**ShortCiteNameOR** is a shortened version of the `CiteName` Component. If omitted, it is calculated by the `role-short-cite-name-format` Property.

```
80 \ccDeclareCountedComponent{ShortCiteName}%
```

**PDFInfoNameOR** is the version of the Role member name that is used in the PDF Info dictionary. If omitted, it is calculated by the `role-pdfinfo-name-format` Property.

```
81 \ccDeclareCountedComponent{PDFInfoName}%
```

**InitialOR** holds the initials of the Role member. If omitted, the initials are attempted to be calculated from the `FirstName` and `MidName` Components via the `initials-format` Property.

```
82 \ccDeclareCountedComponent{Initial}%
```

**LastName** is the surname of the Role member.

```
83 \ccDeclareCountedComponent{LastName}%
```

**FirstName** is the first name of the Role member.

```
84 \ccDeclareCountedComponent{FirstName}%
```

**MidName** is/are the middle name(s) of the Role member.

```
85 \ccDeclareCountedComponent{MidName}%
```

**Honorific** is a other honorific title for the Role member.

```
86 \ccDeclareCountedComponent{Honorific}%
```

**Lineage** is the name suffix, typically something like “jr.” or “the 3rd”.

```
87 \ccDeclareCountedComponent{Lineage}%
```

**ORCID** is the ORCID (Open Researcher and Contributor ID) of the Role member. Depending on the publisher style, this can be a full URL or just the identifier.

```
88 \ccDeclareCountedComponent{ORCID}%
```

**AffilRef** is the ID of an entry in the Affil Component Group.

```
89 \ccDeclareCountedComponent{AffilRef}% for references to the Affil Group
```

**Affiliation** is the Affiliation of the Role member.

**Note** that only one `AffilRef` or `Affiliation` should be used for any Role member, not both at the same time.

```
90 \ccDeclareCountedComponent{Affiliation}% for affiliations as direct Author meta data
```



`Email` is the email/contact address of the Role member.

```
91 \ccDeclareCountedComponent{Email}{%
```

`CorrespondenceAsOR` is how the Role member is to be addressed when she is the corresponding Role member. If omitted, it is calculated by the `role-correspondence-as-format` Property.

```
92 \ccDeclareCountedComponent{CorrespondenceAs}{%
```

### Contributor Role Group Handlers

The Group Handlers fill the previously defined Override Components when they are not explicitly given inside the Component Group.

```
93 }%
94 \ccDeclareGroupHandler{#2}{%
95   \ccUnlessComp{FullName}{\ccComponent{FullName}{\ccUseProperty{#1-full-name-format}}}%
96   \ccUnlessComp{Initial}{\ccComponent{Initial}{\ccUseProperty{initials-format}}}%
97   \ccUnlessComp{CiteName}{\ccComponent{CiteName}{\ccUseProperty{#1-cite-name-format}}}%
98   \ccUnlessComp{ShortCiteName}{\ccComponent{ShortCiteName}{\ccUseProperty{#1-short-cite-name-
99     format}}}%
100   \ccUnlessComp{PDFInfoName}{\ccComponent{PDFInfoName}{\ccUseProperty{#1-pdfinfo-name-format
101     }}}%
102   \ccUnlessComp{CorrespondenceAs}{\ccComponent{CorrespondenceAs}{\ccUseProperty{#1-
103     correspondence-as-format}}}%
104   \ccWhenComp{AffilRef}{\ccWhenComp{Affiliation}{%
105     \ccPackageError{Meta}{Ambiguity}
106     {You cannot use both Containers AffilRef and Affiliation in the same '\ccPrefix#2' Sub-
107     Container}
108     {At least one '\ccPrefix#2' Sub-Container contains both AffilRef and Affiliation. This
109     is not allowed. Please decide for one affiliation strategy: Either two lists with
110     cross-references, or affiliations directly as an author's meta-data.}}}%
111 }
```

### Declaring the Contributor Role's Collection Components

Recall that the Collection Component's name are all prefixed by the Role's name, e.g., the actual Collection Component `➡NameList` of a Role named "Author" is accessed by calling the `➡AuthorNameList` Component.

`NameListCL` is the formatted list of all Role member's `➡FullName` Components according to the `role-block-print-format` Property.

```
106 \ccDeclareRoleBlock{#2}{NameList}{#1-list-print-format}%
```

`CitationListCL` is the formatted list of all Role member's `➡CiteName` according to the `role-block-cite-format` Property.

```
107 \ccDeclareRoleBlock{#2}{CitationList}{#1-list-cite-format}%
```

`ShortCitationListCL` is the formatted list of all Role member's `➡ShortCiteName` Component `role-block-short-cite-format` Property.

```
108 \ccDeclareRoleBlock{#2}{ShortCitationList}{#1-list-short-cite-format}%
```

`PDFInfoCL` is the formatted string that is sent to the PDF's Info dictionary. Its format is determined by the `role-block-pdfinfo-format` Property.

```
109 \ccDeclareRoleBlock[apply]{#2}{PDFInfo}{#1-list-pdfinfo-format}%
```

`\CorrespondenceCL` is the list of all Role member's `\CorrespondanceAs` Components according to the `\role-block-correspondence-format` Property.

```
110 \ccDeclareRoleBlock{#2}{Correspondence}{#1-list-correspondence-format}%
111 }
```

`\ccAddToRole` appends another Component declaration block `{#2}` to a pre-defined Role `{#1}`.

TODO  
make into LaTeX kernel  
hook

```
112 \def\ccAddToRole#1#2{\csgappto{cc@group@#1@hook}{#2}}
```

## 2 Labeled Components

Labeled Components are two Components, one for the Content and one for the Label.

`\ccDeclareLabeledComp` declares two Components: one named `\ccPrefix #2` for the value, and another one named `\ccPrefix #2Label` for its corresponding label. `#3` is used for Property overrides. The optional Argument `#1` allows to set a default value for the Label.

```
113 \def\ccDeclareLabeledComp{\cc@opt@empty\cc@declare@labeled@comp}
114 \def\cc@declare@labeled@comp[#1]#2#3{%
115   \ccDeclareComponent{#2}{\expandafter\global}{}%
116   \ccDeclareComponent{#2Label}{\expandafter\global}{}%
117   \csxdef{labeled-meta-property-infix-\cc@cur@cont-#2}{#3}%
118   \if!#1!\else
119     \long\csgdef{cc@\cc@cur@cont @#2Label}{#1}%
120   \fi\ignorespaces}
```

`\ccUseLabeledComp` returns the Labeled Component with its label. The starred version omits automatic Tagging if the `coco-accessibility` module is active.

```
121 \def\ccUseLabeledComp{\@ifstar{\global\let\ccm@no@tag\relax\cc@use@labeled@comp}{\
  cc@use@labeled@comp}}
122 \def\cc@use@labeled@comp#1{%
123   \ccWhenComp{#1}{%
```

`\ccCurInfix` stores the currently active Property infix for the Labeled Component. Is is used to call the right format Property for the Labeled Component, which defaults to `\labeled-meta-[ccCurInfix]-format`. If this Property doesn't exists, formatting falls back to `\labeled-meta-format`.

```
124 \letcs\ccCurInfix{labeled-meta-property-infix-\cc@cur@cont-#1}%
```

`\ccCurComp` stores the currently active name of the Labeled Component, which is used in the generic `\labeled-meta-format` Property.

```
125 \def\ccCurComp{#1}%
```

```
126 \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatum}\fi
127 \ccIfProp{labeled-meta-\ccCurInfix-format}
128   {\ccUseProperty{labeled-meta-\ccCurInfix-format}}
129   {\ccUseProperty{labeled-meta-format}}%
```

```

130 \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatum}\fi
131 }\global\let\ccm@no@tag\@undefined}

```

### 3 Meta Data Rolemaps for Tagged PDFs

Role mapping for accessibility tagging:

```

132 \ccaAddRolemap{Authors}{P}
133 \ccaAddRolemap{Affiliations}{P}
134 \ccaAddRolemap{MetaDatum}{Div}
135 \ccaAddRolemap{MetaDatumLabel}{P}
136 \ccaAddRolemap{MetaDatumValue}{P}
137 \ccaAddRolemap{MetaDatumBlock}{Div}
138 \ccaAddRolemap{Abstract}{Div}
139 \ccaAddRolemap{AbstractLabel}{P}
140 \ccaAddRolemap{AbstractText}{Div}
141 \ccaAddRolemap{Keywords}{Div}
142 \ccaAddRolemap{KeywordsLabel}{P}
143 \ccaAddRolemap{KeywordsText}{Div}

```

### 4 Common Meta Data

`\ccm@declare@comp` defines some commonly used meta Components

```

144 \def\ccm@declare@comp{%

```

`Copyright` holds the Copyright notice.

```

145 \ccDeclareComponent{Copyright}{\expandafter\global}{}% Copyright text

```

`LicenceLogo` is a component for a license logo. This usually contains an `\includegraphics`.

```

146 \ccDeclareComponent{LicenceLogo}%

```

`LicenceName` is the name of the license.

```

147 \ccDeclareComponent{LicenceName}%

```

```

148 }%

```

`article-meta` is an abstract Container that holds meta data specific to a journal's Article.

```

149 %% for single articles
150 \ccDeclareContainer{article-meta}{%
151 \ccDeclareType{Components}{%

```

`StartPage` is the number of the starting page of an article

```

152 \ccDeclareGlobalComponent{StartPage}

```

`EndPage` is the number of the ending page of an article

```
153 \ccDeclareGlobalComponent{EndPage}
```

`CiteAs` holds a string as to how the article should be cited in other publications.

```
154 \ccDeclareLabeledComp[Cite as]{CiteAs}{cite-as}
```

`Submitted` holds the date when the article was submitted to the journal.

```
155 \ccDeclareLabeledComp[Submitted]{Submitted}{submitted}
```

`Received` holds the date when the article was received by the journal

```
156 \ccDeclareLabeledComp[Received]{Received}{received}
```

`Revised` holds the date when the article was revised by its author(s)

```
157 \ccDeclareLabeledComp[Revised]{Revised}{revised}
```

`Reviewed` holds the date when the article was reviewed by the editors or reviewers.

```
158 \ccDeclareLabeledComp[Reviewed]{Reviewed}{reviewed}
```

`Accepted` holds the date when the article was accepted for publication by the journal.

```
159 \ccDeclareLabeledComp[Accepted]{Accepted}{accepted}
```

`Published` holds the date when the article is due to be published.

```
160 \ccDeclareLabeledComp[Published]{Published}{published}
```

`COIStatement` holds the author's Conflict of Interest statement

```
161 \ccDeclareLabeledComp[Conflict of Interest]{COIStatement}{coi-statement}
```

```
162 }}
```

`\ccm@extended@common@macros` provides some extended markup. Some headings use these Components for compilations of contributions by different authors. They are also loaded by article title pages.

```
163 \def\ccm@extended@common@macros{%
```

`Abstract` holds the contribution's abstract or content summary.

```
164 \ccDeclareLabeledComp[Abstract]{Abstract}{abstract}%
```

`Keywords` holds a list of keywords related to the contribution.

```
165 \ccDeclareLabeledComp[Keywords]{Keywords}{keyword}%
```

**DOI** holds the Digital Object Identifier. Depending on the Publisher style, this may be the full URI, or just the identifier.

```
166 \ccDeclareLabeledComp{DOI}{doi}%
```

**TitleEn** holds the English title of the publication when the contribution's main language is *not* english.

```
167 \ccDeclareLabeledComp{TitleEn}{title-en}%
```

```
168 \ccm@generic@comp
169 }
```

## 4.1 Affiliations

`\ccm@declare@affils` is a wrapper that creates the user-level macros for the affiliations.

```
170 \def\ccm@declare@affils{%
```

**AffilBlockCC** is the Collection Component for the contribution's Affiliations list. **Note** that the [↗AffilBlock](#) itself is not generated in this module. The two modules `coco-headings` and `coco-title` that both depend on the `coco-meta` module have their own mechanisms to build their respective [↗AffilBlock](#) Collection Components.

```
171 \ccDeclareComponent{AffilBlock}{\expandafter\global}{}%
```

```
172 \ccDeclareComponentGroup{Affil}{%
```

**AffiliationOR** is the fully formatted Affiliation string. If omitted, the Component is built using the [⚙️affiliation-format](#) Property.

```
173 \ccDeclareCountedComponent{Affiliation}%
```

**Address** is the address where the Role member is working.

```
174 \ccDeclareCountedComponent{Address}%
```

**Institute** is the name of the university, department or institution where the Role member is working

```
175 \ccDeclareCountedComponent{Institute}%
```

**Country** is the country where the institution is located in.

```
176 \ccDeclareCountedComponent{Country}%
```

**Department** is the department where the Role member is working.

```
177 \ccDeclareCountedComponent{Department}%
```

**AffilIDOR** is the internal identifier that is referenced by the Role member's [↗AffilRef](#) Component. If omitted, the ID is the value of an automatic counter that is incremented by one at the beginning of each Affil Group Container counter in the same Parent Container.

```
178 \ccDeclareCountedComponent{AffilID}%
```

```

179 }%
180 \ccDeclareGroupHandler{Affil}{%
181   \ccUnlessComp{AffilID}{\ccComponentEA{AffilID}{\ccAffilCnt}}}%
182   \ccUnlessComp{Affiliation}{\ccComponent{Affiliation}{\ccUseProperty{affiliation-format}}}%
183 }%
184 }

```

## 5 Meta Data Properties

```

185 \ccAddToType{Properties}{CommonMeta}{%

```

### 5.1 Initials

`initials-format` <any> generates an Role member's initials from the ➡`FirstName` and ➡`MidName` Components.

```

186 \ccSetProperty{initials-format}{%
187   \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
      cc@long@empty\else
188   \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
      relax\else
189   \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\
      the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
190   \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
      cc@long@empty\else
191   \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
      relax\else
192   \ccUseProperty{initials-sep}%
193   \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\
      the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
194   \fi\fi
195   \fi\fi
196 }

```

`initials-sep` <any> is the separator between two ➡`Initials`.

```

197 \ccSetProperty{initials-sep}{~}

```

`initials-period` <any> is the symbol that is inserted at the end of each ➡`Initial`.

```

198 \ccSetProperty{initials-period}{.}

```

### 5.2 Member Role Composition Properties

### 5.3 Overrides Within a Role Counted Component

The next Properties control how the Component Overrides within a single Role Counted Component are composed.

`role-full-name-format` <any> how the ➡`FullName` Component for each Role member is generated.

```

199 \ccSetProperty{role-full-name-format}{%
200   \if\ccUseComp{Honorific}\relax
201   \else

```

```

202     \ccUseComp{Honorific}\space
203     \fi
204     \ccUseComp{FirstName}\space
205     \if\ccUseComp{MidName}\relax
206     \else
207         \ccUseComp{MidName}\space
208     \fi
209     \ccUseComp{LastName}%
210     \if\ccUseComp{Lineage}\relax
211     \else
212         \space\ccUseComp{Lineage}%
213     \fi%
214 }%

```

`role-cite-name-format` <any> how the ➡CiteName for each Role member is formatted.

```

215 \ccSetProperty{role-cite-name-format}{\ccIfComp{LastName}{\ccUseComp{LastName},~\ccUseComp{
    Initial}}{\ccUseComp{FullName}}}% How CiteName for each name is built

```

`role-short-cite-name-format` <any> how the ➡ShortCiteName Component of a Role member is formatted

```

216 \ccSetProperty{role-short-cite-name-format}{\ccUseComp{LastName}}%

```

`role-pdfinfo-name-format` <any> how the ➡PDFInfoName of a Role member is formatted

```

217 \ccPropertyLet{role-pdfinfo-name-format}{role-cite-name-format}%

```

`role-correspondence-as-format` <any> How the ➡CorrespondenceAs string of a Role Member is formatted

```

218 \ccSetProperty{role-correspondence-as-format}{\ccUseComp{Email}}%

```

## 5.4 Format of Single Role Collection Component Items

the next properties control how single items in the Parent container's Collection Components are to be formatted.

`role-block-print-format` <any> How a single entry in the ➡NameList is formatted.

```

219 \ccSetProperty{role-block-print-format}{\ccUseComp{FullName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}%

```

`role-block-cite-format` <any> how a single entry in the ➡CitationList is formatted

```

220 \ccSetProperty{role-block-cite-format}{\ccUseComp{CiteName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}%

```

`role-block-short-cite-format` <any> how a single entry in the ➡ShortCitationList is formatted

```

221 \ccSetProperty{role-block-short-cite-format}{\ccUseComp{ShortCiteName}\ifnum\ccCurCount<\
    ccTotalCount\ccUseProperty{counted-name-sep}\fi}%

```

`role-block-pdfinfo-format` <any> how a single entry in the ➡PDFInfo Component is formatted

```

222 \ccSetProperty{role-block-pdfinfo-format}{\ccUseComp{PDFInfoName}\ifnum\ccCurCount<\
    ccTotalCount\and\fi}% How each item in the Component <Role>PDFInfo is formatted

```

**role-block-correspondence-format** <any> how a single entry in the ➡Correspondence Component is formatted

```

223 \ccSetProperty{role-block-correspondence-format}{%
224   \ccIfPropVal{is-corresp}{true}
225   {\ifx\is@first@corresp\relax
226     \ccUseProperty{corresp-sep}%
227   \else
228     \global\let\is@first@corresp\relax
229   \fi
230   \ccUseComp{CorrespondenceAs}%
231 }{}}%
```

**counted-name-sep** <any> how single entries in ➡NameList are separated

```

232 \ccSetProperty{counted-name-sep}{,\space}%
```

**name-and** <any> is a Property that can be used when composing Role specific Collection Components. Is is usually used between the penultimate and the last entry in the Collection Component.

```

233 \ccSetProperty{name-and}{\space and\space}%
```

**name-et al** <any> is a Property that can be used when composing Role specific Collection Components. Is is usually used after the first entry in the list, when the total number of entries is too large.

```

234 \ccSetProperty{name-et al}{\space et~al.}%
```

**name-sep** <any> is the default separator between entries in a Role specific Collection Component.

```

235 \ccSetProperty{name-sep}{,\space}%
```

**keywords-sep** <any> is the default separator between Entries in the Keywords list.

```

236 \ccSetProperty{keywords-sep}{,\space}%
```

**corresp-mark** <any> is the default marker for the “Correspondence” Role member, i.e., the Role member who is designated the primary contact person of a contribution.

```

237 \ccSetProperty{corresp-mark}{*}%
```

**corresp-sep** <any> is the default separator between entries in the ➡Correspondence Collection Component.

```

238 \ccSetProperty{corresp-sep}{,\space}%
```

## 5.5 Collection Component Properties Specific to Author Role

The Properties defined here are mostly aliases of the more generic Properties defined above.

### Author Instance Override Properties

**author-cite-name-format** <any> how an Author’s ➡CiteName is formatted.

```

239 \ccPropertyLet{author-cite-name-format}{role-cite-name-format}%
```

**author-short-cite-name-format** <any> how an author’s ➡ShortCiteName is formatted.

```

240 \ccPropertyLet{author-short-cite-name-format}{role-short-cite-name-format}%
```



**author-full-name-format** <any> how an author's [Full Name](#) Component is composed

241 \ccPropertyLet{author-full-name-format} {role-full-name-format}%

**author-pdfinfo-name-format** <any> how an author's [PDFInfoName](#) Component is composed

242 \ccPropertyLet{author-pdfinfo-name-format} {role-pdfinfo-name-format}%

**author-correspondence-as-format** <any> how an author's [CorrespondenceAs](#) entry is to be formatted

243 \ccPropertyLet{author-correspondence-as-format} {role-correspondence-as-format}%

### Author-Specific Collection Component Override Properties

**author-list-print-format** <any> is the format of each entry in the [AuthorNameList](#) Component.

244 \ccPropertyLet{author-list-print-format} {role-block-print-format}%

**author-list-cite-format** <any> is the format of each entry in the [AuthorCitationList](#) Component.

245 \ccPropertyLet{author-list-cite-format} {role-block-cite-format}%

**author-list-short-cite-format** <any> is the format of each entry in the [AuthorShortCitationList](#) Component.

246 \ccPropertyLet{author-list-short-cite-format} {role-block-short-cite-format}%

**author-list-pdfinfo-format** <any> is the format of each entry in the [AuthorPDFInfo](#) Component.

247 \ccPropertyLet{author-list-pdfinfo-format} {role-block-pdfinfo-format}%

**author-list-correspondence-format** <any> is the format of the [AuthorContribution](#) Collection Component.

248 \ccPropertyLet{author-list-correspondence-format} {role-block-correspondence-format}%

## 5.6 Format of Affiliation Lists

**affiliation-format** <any> is the format of the [Affiliation](#) Component for each [Affil](#) Instance.

249 \ccSetProperty{affiliation-format}{%  
250 \ccWhenComp{Institute}{\ccUseComp{Institute}}%  
251 \ccWhenComp{Department}{, \ccUseComp{Department}}%  
252 \ccWhenComp{Address}{, \ccUseComp{Address}}%  
253 }%

**affil-sep** <any> is the separator between the entries of the [AffilBlock](#) Collection Component.

254 \ccSetProperty{affil-sep}{\par}

**affil-block-item-face** <any> are the font parameters used to print each entry in the [AffilBlock](#) Collection Component.

255 \ccSetProperty{affil-block-item-face}{%}

`affil-block-item-format` is the format of each entry in the `AffilBlock` list

```

256 \ccSetProperty{affil-block-item-format}{%
257   \textsuperscript{\ccUseComp{AffilID}}%
258   \bgroup
259   \ccUseProperty{affil-block-item-face}%
260   \ccUseComp{Affiliation}
261   \egroup%
262   \ifnum\ccCurCount<\ccTotalCount\relax\ccUseProperty{affil-sep}\fi%
263 }

```

`affil-block-face` <any> is the font used to print the `AffilBlock` Collection Component.

```

264 \ccSetProperty{affil-block-face}{\small\normalfont}%

```

`affil-block-format` <any> prints the `AffilBlock` Collection Component.

```

265 \ccSetProperty{affil-block-format}{%
266   \ccWhenComp{AffilBlock}
267   {\bgroup
268     \ccUseProperty{affil-block-face}%
269     \ccUseComp{AffilBlock}%
270   \egroup
271   \par
272 }}

```

## 5.7 Properties for Labeled Componentns

`labeled-meta-format` <any> is the generic Property that determines how Labeled Components are composed. It checks for implicit formatting properties specific to each labeled Component and falls back to generic defaults if those are not defined by the user or publisher style.

```

273 \ccSetProperty{labeled-meta-format}{%
274   \ccIfProp{labeled-meta-before-\ccCurInfix}
275   {\ccUseProperty{labeled-meta-before-\ccCurInfix}}
276   {\ccUseProperty{labeled-meta-before}}%
277   \bgroup
278   \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumLabel}\fi
279   \ccIfProp{labeled-meta-\ccCurInfix-face}
280   {\ccUseProperty{labeled-meta-\ccCurInfix-face}}
281   {\ccUseProperty{labeled-meta-face}}%
282   \ccIfProp{labeled-meta-\ccCurInfix-label-format}
283   {\ccUseProperty{labeled-meta-\ccCurInfix-label-format}}
284   {\ccUseProperty{labeled-meta-label-format}}%
285   \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumLabel}\fi
286   \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumValue}\fi
287   \ccUseComp{\ccCurComp}%
288   \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumValue}\fi
289   \egroup
290   \ccIfProp{labeled-meta-after-\ccCurInfix}
291   {\ccUseProperty{labeled-meta-after-\ccCurInfix}}
292   {\ccUseProperty{labeled-meta-after}}%
293 }

```

`labeled-meta-label-format` <any> is the generic format of the label of a Labeled Component.

```

294 \ccSetProperty{labeled-meta-label-format}{%
295   \ccWhenComp{\ccCurComp Label}{%
296     \bgroup

```

```

297 \ccUseProperty{labeled-meta-before-\ccCurInfix-label}%
298 \ccIfProp{labeled-meta-\ccCurInfix-label-face}
299   {\ccUseProperty{labeled-meta-\ccCurInfix-label-face}}
300   {\ccUseProperty{labeled-meta-label-face}}%
301 \ccUseComp{\ccCurComp Label}%
302 \ccIfProp{labeled-meta-\ccCurInfix-label-sep}
303   {\ccUseProperty{labeled-meta-\ccCurInfix-label-sep}}
304   {\ccUseProperty{labeled-meta-label-sep}}%
305 \egroup
306 }

```

**labeled-meta-label-face** <any> is the font setting for the Label of a Labeled Component.

```

307 \ccSetProperty{labeled-meta-label-face}{\bfseries}

```

**labeled-meta-label-sep** <any> is the default and fallback separator between the Labeled Component's [Label](#) and its value.

```

308 \ccSetProperty{labeled-meta-label-sep}{:\enskip}

```

**labeled-meta-face** <any> is the face of a Labeled Component. It applies to both the Label and the Value, but can be locally overridden by the [labeled-meta-label-face](#) Property.

```

309 \ccSetProperty{labeled-meta-face}{}

```

**labeled-meta-before** <any> is the code expanded before the Labeled Component is printed.

**Note** that the Property is expanded *outside* the local group of the Labeled Component.

```

310 \ccSetProperty{labeled-meta-before}{}

```

**labeled-meta-after** <any> is the code expanded after the Labeled Component is printed.

**Note** that the Property is expanded *outside* the local group of the Labeled Component.

```

311 \ccSetProperty{labeled-meta-after}{\par}

```

```

312 }

```

```

</meta>

```



## Module 6

# coco-headings.dtx

```
<*headings>
```

This module provides handlers for headings like parts, chapters, sections, or inline headings common to all CoCoTeX projects.

```
23 %%
24 %% module for CoCoTeX that extends heading objects.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive >= 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-headings}
32   [2024/07/16 0.5.0 CoCoTeX headings module]
33 \RequirePackage{coco-meta}
```

Headings are handled differently with `cocotex.cls` compared to standard  $\text{\LaTeX}$ , since cocotex manuscripts tend to have a whole collection of additional information that are pressed into the headings, like subtitles or section authors down to subsection level, etc. Therefore, the `\@startsection` and `\@make[s]chapterhead` facilities from  $\text{\LaTeX}$  are no longer sufficient. At the same time, the package does not redefine those macros and keeps them available for backwards compatibility.

First, we load the `bookmark` package:

```
34 \RequirePackage{bookmark}%
```

Since we use our own heading levels, we disable all automatically generated bookmarks.

```
35 \hypersetup{bookmarksdepth=-999}%
```

## 1 Facility for declaring heading levels and their layouts

`Heading` is an abstract parent Container for headings. It inherits from `CommonMeta`.

```
36 \ccDeclareContainer{Heading}{%
37   \ccInherit{Components,Properties}{CommonMeta}%
38   \ccDeclareType{Parent}{}%
39   \ccDeclareType{Components}{%
```

We already have the `Author` Component inherited from the `CommonMeta` Container. We therefore just need to declare the overrides.

```
40   \cch@provide@authors%
```

The remaining Components are built as usual.

`Title` is the main title of the heading.

```
41 \cch@provide@comp{Title}%
```

`Subtitle` is an optional second-level title of the heading.

```
42 \cch@provide@comp{Subtitle}%
```

`Number` is the heading's counter.

```
43 \cch@provide@comp{Number}%
```

`RefLabel` is a unique ID of an heading. It is targeted by cross references and replaces L<sup>A</sup>T<sub>E</sub>X's `\label` command.

```
44 \ccDeclareComponent{RefLabel}{-}{-}%
```

```
45 \cch@provide@quotes
46 }%
47 \ccDeclareType{Attributes}{%
```

`class <string>` is the style class of the heading.

```
48 \ccDeclareAttributeHandler{class}{%
49 \let\cch@current@class\ccAttrVal
50 \expandafter\ccUseStyleClass\expandafter{\ccAttrVal}{Heading}%
51 }%
```

```
52 }
53 \ccDeclareType{Properties}{-}%
54 \ccDeclareEnv{\cch@heading}{\cch@end@heading}%
55 }
```

`\ccDeclareHeading` is the user-level macro to declare new headings. There also exists a *starred* version of this macro, which exempts the declared heading from auto-tagging.

- #1 (optional) inherit-from: load all properties from that heading level, first.
- #2 level: used for toc entries. -1 for part, 0 for chapter, 1 for section, etc.
- #3 name: part, chapter, section, etc, to be used in toc, head lines, bookmarks, etc.
- #4 Property definitions and switches

```
56 \long\def\ccDeclareHeading{\global\let\cch@star@hdg\@undefined\ifstar{\global\let\cch@star@hdg\
57 relax\cc@declare@heading}{\cc@declare@heading}}
58 \long\def\cc@declare@heading{\cc@opt@empty\cc@declare@heading}
59 \long\def\cc@declare@heading[#1]#2#3#4{%
```

First, we check if the heading has already been declared.

```
59 \ifcsdef{cc@container@#3}{%
```

If yes, then we check if the new declaration's parameters match with the pre-existing one. We start with the heading level.

```
60 \ccPackageInfo{Headings}{-}{Appending to `#3'}%
61 \ifcsstring{cch@#3@level}{#2}{-}{%
62 \ccPackageError{Headings}
63 {Level Mismatch}}
```

```

64     {Level of heading `#3' cannot be altered!}
65     {The already existing heading `#3' has toc level `\'csname cch@#3@level\endcsname', but
        your^^J%
66     re-declaration states `#2'.^^J%
67     ^^J%
68     Consider declaring a new heading altogether with `#3' as parent,^^J%
69     or add Properties to `#3' using \string\ccAddToType\string{Properties\string}\string
        {#3\string}.}%
70 }%

```

we also check the parent.

```

71 \if!#1!\else
72 \ifcsstring{cc@parent@#3}{#1}{}%
73 \ccPackageError{Headings}
74 {Parent Mismatch}
75 {Parent of heading `#3' cannot be altered!}
76 {The already existing heading `#3' inherits from `\'csname cc@parent@#3\endcsname',^^J%
77 but your re-declaration sets Parent to `#1'.^^J%
78 ^^J%
79 Consider declaring a new heading altogether with `#1' as parent.}%
80 }%
81 \fi

```

and finally pass the new Properties to the existing heading.

```

82 \ccAddToType{Properties}{#3}{#4}%

```

Finally, we need to re-define the `\ccUseHeading` macro so that changes to the heading's Property list will be taken into account for all dependend constructions like list-ofs and toc-entries.

```

83 \cch@declare@heading{#2}{#3}%
84 }{% \ifcsdef cc@container@#3 else

```

If the heading does not already exist, we build a new one.

Each new heading constitutes its own Sub-Container of the heading Container. The name of this Sub-Container is the headings name.

```

85 \ccDeclareContainer{#3}{%

```

`\cch@#3@level` stores the numeric heading level for the heading

```

86 \csgdef{cch@#3@level}{#2}%

```

`\cch@2@unique` is a unique name for the heading's level. Is is always the name of the *first* heading that is defined with a given heading level counter.

```

87 \ifcsdef{cch@#2@unique}{\csgdef{cch@#2@unique}{#3}}%

```

```

88 \ccPackageInfo{Headings}{Declaring heading `#3'}%
89 \edef\@argi{#1}%
90 \ccDeclareType{Parent}{\cch@create@parent{#1}{#3}}

```

We inherit everything from the heading levels parent, or from the default heading if no parent is present.

```

91 \ifx\@argi\@empty
92 \ccInherit{Components,Properties}{Heading}%
93 \else

```

```

94 \ccInherit{Components,Properties,Parent}{#1}%
95 \fi

```

The main body of the heading Declaration is a list of Property definitions which we append to the Sub-Container’s “Property” Type.

```

96 \ccDeclareType{Properties}{%
97   #4%
98 }%

```

For each heading we declare some common macros like the ToC entry handlers, the heading’s counters and its hooks.

```

99 \ccDeclareType{Init}{%
100   \cch@init@hooks{#3}%
101   \let\@cch@cur@cont\cc@cur@cont
102   \def\cc@cur@cont{Heading}%
103   \cc@init@l@{toc}{#2}{#3}%
104   \let\cc@cur@cont\@cch@cur@cont
105   \cch@init@cnt{#3}%
106 }%

```

Unlike other Sub-Containers, headings form no own L<sup>A</sup>T<sub>E</sub>X environment. Instead, headings are specifications of one common `\ccPrefix Heading` environment. Is is outsourced into the internal `\cch@declare@heading` macro, which is defined below.

The reason for that is that we don’t want to define versions of the same property macros for each and every single heading level. Instead, we locally re-define the general low-level macros that represent the heading’s properties for each instance of the generalised `Heading` container.

```

107 \cch@declare@heading{#2}{#3}%
108 }% \ccDeclareContainer{#3}
109 }% \ifcsdef cc@container@#3 fi

```

If CoCoT<sub>E</sub>X’s accessibility features are active and the heading is declared with the non-starred version of `\ccDeclareHeading`, we need to register each new heading with `ltpdfa`’s autoclose mechanism.

```

110 \ccWhenAlly{%
111   \ifx\cch@star@hdg\relax
112     \expandafter\global\expandafter\let\csname cch@notag@#3\endcsname\@empty
113   \else
114     \cch@add@autoclose{#2}{#3}%
115   \fi
116 }%
117 }% \cc@declare@heading

```

Each new heading level needs some configuration with the `ltpdfa` package in order to automatically close heading tags with the beginning of a new heading.

`\cch@add@autoclose` adds the new heading level to `ltpdfa`’s autoclose mechanism. #1 is the numeric level, #2 is the name of the heading. We do this inside the `cca/before/begin/document` hook, since we need to know *all* locally defined heading levels beforehand in order to build the Sectioning tree correctly.

Note that this whole method is only called for headings declared with the non-starred version of `\ccDeclareHeading`.

```

118 \newcount\cch@tempcnta \cch@tempcnta\z@
119 \def\cch@add@autoclose#1#2{%
120   \AddToHook{cca/document/begin}[cch/autoclose]{%

```

First, we assign the Sectioning tag and the tag for the section’s head itself to the `Sect` and `H#` tags, respectively.



```
121 \ccaAddRolemap{#2}{Sect}%
```

Then we determine the hierarchical heading level we need to assign to the PDF tags. H1 is always reserved for the entire document's title, so we need to calculate the difference of the lowest used value and 2 and add this to the actual level of the current heading.

```
122 \cch@tempcnta=\numexpr\tw@-\cch@highest@level\relax
123 \advance\cch@tempcnta by #1\relax
124 \ccaAddRolemap{#2head}{H\the\cch@tempcnta}%
```

**TODO**  
Mixing numbered and plain H-Tags like that should be incompatible to the PDF/UA standard! On the other hand, H7 and higher is not recognized by neither VeraPDF nor PAC.

```
125 \ifnum\cch@tempcnta>6\relax
126 \ccaAddRolemap{H\the\cch@tempcnta}{H}%
127 \fi
```

Next, we tell `ltpdfa` for each heading level which other heading level is the next down the Sectioning hierarchy. For that, we first put the current heading level in a calculable counter.

```
128 \cch@tempcnta=#1\relax
```

Then we catch the heading with the highest level (from the aux file) and set the `document` layer in the `ltpdfa`'s `Sectioning` table to have that heading as its child

```
129 \ccDebugMsg[a11y]{^^J==> Comparing #2 (\the\cch@tempcnta) with lowest \cch@lowest@level,
    and highest \cch@highest@level}%
130 \ifnum\cch@tempcnta=\cch@highest@level
131 \ccDebugMsg[a11y]{^^J==> #2 is highest, setting it as child of document}%
132 \edef\x{\noexpand\ccaAddToConfig{autoclose}{document}={Type:Sectioning}{Child:\csname cch@#1
    @unique\endcsname}{Egroup:false}}}\x%
133 \fi
```

Then, we catch the lowest level to tell `ltpdfa`'s `Sectioning` table that this level has no children. Another switch is made to distinguish first-born heading levels from aliases, since the `Sectioning` table can only hold one heading per level. All other headings of the same level are, per definition, Aliases of the one that has been defined first.

```
134 \ifnum\cch@tempcnta=\cch@lowest@level\relax
135 \ccDebugMsg[a11y]{^^J==> #2 is lowest, Setting Child to none}%
136 \ifcsstring{cch@#1@unique}{#2}
137 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2}={Type:Sectioning}{Child:none}{Egroup:
    false}}}\x}
138 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2}={Type:Sectioning}{Child:none}{Egroup:
    false}{Alias:\csname cch@#1@unique\endcsname}}}\x}%
139 \else
```

For all higher heading levels, we look for the next lower heading

```
140 \ifnum\the\cch@tempcnta<\cch@highest@level\relax
141 \@tempswafalse
```

```

142 \else
143 \ccDebugMsg[a11y]{^^J==> #2 goes into the Loop...}%
144 \@tempswatrue
145 \fi
146 \loop

```

by incrementing the heading level counter by one

```

147 \advance\cch@tempcnta\@ne\relax

```

and checking the variable `repeat` condition:

```

148 \if@tempswa

```

We don't go further when the current loop counter is already larger than the heading level with the highest level counter.

```

149 \ifnum\cch@tempcnta>\cch@lowest@level\relax
150 \ccDebugMsg[a11y]{^^J==> #2's next \csmeaning{cch@\the\cch@tempcnta @unique} is too
    low, breaking loop...}%
151 \@tempswafalse
152 \else

```

If we are below the highest level, we check if a heading with the current level is defined

```

153 \expandafter\ifx\csname cch@\the\cch@tempcnta @unique\endcsname\relax
154 \ccDebugMsg[a11y]{^^J==> #2's next \csmeaning{cch@\the\cch@tempcnta @unique} is in
    range, going on (cur: \the\cch@tempcnta)...}%

```

if not, we continue. This is the case, when heading levels are not sequentially numbered. Which does (and did) happen. For reasons unknown...

```

155 \@tempswatrue
156 \else

```

If the heading level is defined, we configure `autoclose` such that the level with the iterator counter is set to be the child of the current heading level in `ltpdfa`'s `Sectioning` table. As above, we distinguish between original headings and Aliases.

```

157 \ccDebugMsg[a11y]{^^J==> Setting \csmeaning{cch@\the\cch@tempcnta @unique} as Child
    of #2 with cnt: \the\cch@tempcnta...}%
158 \ifcsstring{cch@#1@unique}{#2}
159 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:\csname cch@
    \the\cch@tempcnta @unique\endcsname}{Egroup:false}}}\x}
160 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:\csname cch@
    \the\cch@tempcnta @unique\endcsname}{Egroup:false}{Alias:\csname cch@#1@unique\
    endcsname}}}\x}%
161 \@tempswafalse
162 \fi
163 \fi

```

We repeat this as long as `\@tempswa` is false. This ensures that all heading levels have exactly one child assigned to them.

```

164 \repeat
165 \fi
166 }}

```

`\cch@min@level` is a temporary counter that stores and constantly updates the *lowest* value for the used heading level.

```
167 \newcount\cch@min@level \cch@min@level=99\relax
```

`\cch@max@level` is a temporary counter that stores and constantly updates the *highest* value for the used heading level.

```
168 \newcount\cch@max@level \cch@max@level=-99\relax
```

`\cch@highest@level` stores the level number of the highest *used* heading level from the previous tex run.

```
169 \ifx\cch@highest@level\@undefined \def\cch@highest@level{-99}\fi
```

`\cch@lowest@level` stores the level number of the lowest *used* heading level from the previous tex run.

```
170 \ifx\cch@lowest@level\@undefined \def\cch@lowest@level{99}\fi
```

both temporary counters are written into the aux file at the very end of the document for consecutive tex runs.

```
171 \AtEndDocument{%
172   \immediate\write\@mainaux{\string\gdef\string\cch@highest@level{\the\cch@min@level}}%
173   \immediate\write\@mainaux{\string\gdef\string\cch@lowest@level{\the\cch@max@level}}%
174 }%
```

`\cch@create@parent` stores the heading level's name and its parent, if it exists.

```
175 \def\cch@create@parent#1#2{%
176   \def\ccCurSecName{#2}%
177   \if!#1!\else
178     \ccCheckParent{#1}{#2}%
179   \fi%
180 }
```

`\cch@declare@heading` consists of two parts: In the first part, the inheritance mechanism and the initializers for each new heading level are triggered.

#1 is the numeric heading level, #2 is the name of the heading.

```
181 \def\cch@declare@heading#1#2{%
182   \ccEvalType{Parent}%
183   \ccEvalType{Init}%
```

`\ccUseHeading` is defined as second step. It is called at the end of each `\ccPrefix Heading` environment to process the Components within the Container instance. Each heading level has its own “version” of this macro.

```
184 \csgdef\ccUseHeading#2{%
```

Since heading levels don't define their own environments, we make sure that `Heading` is the namespace we are working in.

```
185   \ccSetContainer{Heading}%
186   \@setpar{\@@par}%
```

Properties are stored in macros specific to the current heading Sub-Container, therefore we evaluate the level's Properties, not those of the `Heading` Container. However, since we made use of the inheritance mechanism earlier, each Sub-Container's Property list also contains the general `Heading` Property list.

```

187 \def\cchLevel{#1}%
188 \ccEvalType[#2]{Properties}%

```

Processing the author name list (from coco-meta.sty).

```

189 \ccm@role@eval{Author}%
190 \ccComposeCollection{Author}{author-contact-block-format}{AuthorContactBlock}%
191 \ccComposeCollection{Affil}{affil-block-item-format}{AffilBlock}%

```

Processing the `Quote` Group Container, if any.

```

192 \ccComposeCollection{Quote}{quote-block-format}{QuoteBlock}%

```

Hyperref related stuff.

```

193 \def\Hy@toclevel{#1}%

```

Call the mechanism to calculate the heading's counter.

```

194 \cch@auto@number{#1}{#2}%

```

Here, the actual construction of the heading begins.

```

195 \ccUseProperty{heading-par}%
196 \cch@use@hook{before}{#2}%
197 \ccUseProperty{before-heading}%

```

Add vertical space before the heading

```

198 \cch@add@before@skip

```

The counters we calculated earlier and the space needed to render them are evaluated

```

199 \cc@format@number{}{#1}%

```

The value of `after-skip` is essential to determine whether the heading is to be displayed as block or inline element. In case, some heading definition omits setting a proper value, we build a fallback.

```

200 \ccIfProp{after-skip}{\expandafter\global\expandafter\@tempskipa\expandafter=\ccUseProperty{
after-skip}\relax}{\global\@tempskipa=1sp\relax}%

```

Now, we store the counters for the actually used heading levels.

```

201 \expandafter\ifx\csname cch@notag@#2\endcsname\@empty\else
202 \ifnum#1<\cch@min@level\relax\global\cch@min@level=#1\relax\fi
203 \ifnum#1>\cch@max@level\relax\global\cch@max@level=#1\relax\fi
204 \fi

```

```

205 \cch@use@hook{print/before}{#2}%
206 \def\@svsec{%

```

The `heading block` is the composition of all of the heading's Components that are to be printed where the `heading` environment is in the source.

```

207 \ccUseProperty{before-heading-block}%

```

Labels to be used with LaTeX's cross reference mechanism are defined

```

208 \ccCreateLabel{#2}% label facility
209 \leftskip\ccUseProperty{margin-left}%
210 \rightskip\ccUseProperty{margin-right}%
211 \bgroup
212 \ccUseProperty{heading-block}%

```

Generate entries for ToC, bookmarks and page headers. This has to be here because in rare cases, abstracts could cause the whole heading to spread over more than one page and that results in the ToC entry pointing to the last page.

**Style programmers need to make sure that no page breaks are allowed within the heading-block!**

```

213 \ccIfPropVal{no-toc}{true}{\cch@make@toc}% ToC entries
214 \ccIfPropVal{no-BM}{true}{\cch@make@bookmarks}% Bookmarks
215 \ccUseProperty{toc-hook}%
216 \ccIfProp{extended}{\ccUseProperty{extended-heading}}{}%
217 \egroup%
218 \cch@make@run% Running headers
219 \cch@use@hook{run/after}{#2}%
220 \ccUseProperty{after-heading-block}%
221 \cch@use@hook{after}{#2}%
222 }%

```

Finally, we decide whether the printable material we stored in `\@svsec` is to be rendered as a block or inline. This is adopted from L<sup>A</sup>T<sub>E</sub>X's `\@startsection`. The distinction is made by the sign of `after-skip`: a positive value yields a block heading, a negative value yields an inline heading.

```

223 \ifdim\@tempskipa <\z@\relax
224 \cch@make@inline%
225 \else
226 \cch@make@block%
227 \fi

```

This macro is called at the end of the heading environment. In order to deal with possible vertical spaces after the heading, we wait until the group of the heading environment is closed before we actually print the fully composed heading. The definition of `\next` happens in either `\cch@make@inline` or `\cch@make@block`.

```

228 \aftergroup\next%
229 }%
230 }

```

`\cch@use@hook` recursively includes a hook #1 from the heading #2's parent before expanding its own version.

```

231 \def\cch@use@hook#1#2{%
232 \expandafter\ifx\csname cc@parent@#2\endcsname\relax\else
233 \letcs\cch@parent{cc@parent@#2}%
234 \cch@use@hook{#1}{\csname cc@parent@#2\endcsname}%
235 \fi
236 \UseHook{cc/headings/#2/#1}%
237 \ignorespaces}

```

`\cch@add@before@skip` is a routine that determines the skip that is inserted before a heading.

```

238 \def\cch@add@before@skip{%
239 \setlength\@tempskipa{\ccUseProperty{before-skip}}%
240 \ifdim\@tempskipa<\z@\relax
241 \def\do@skip{\minusvspace{-\@tempskipa}}%
242 \else
243 \def\do@skip{\addvspace{\@tempskipa}}%

```

```

244 \fi%
245 \if@nobreak
246 \everypar{}%
247 \do@skip
248 \else
249 \addpenalty\@secpenalty
250 \do@skip
251 \fi}

```

## 1.1 Initializers for New Heading Levels

`\cch@init@hooks` initializes the Hooks for heading level #1.

```

252 \def\cch@init@hooks#1{%

```

`cc/headings/[level]/toc/before` is expanded before the ToC entry is printed

```

253 \NewHook{cc/headings/#1/toc/before}%

```

`cc/headings/[level]/toc/after` is expanded after the ToC entry is printed

```

254 \NewHook{cc/headings/#1/toc/after}%

```

`cc/headings/[level]/before` is expanded before the before-heading property called

```

255 \NewHook{cc/headings/#1/before}%

```

`cc/headings/[level]/after` is expanded after the `after-heading-block` property was called.

```

256 \NewHook{cc/headings/#1/after}%

```

`cc/headings/[level]/print/before` is expanded just before `\@svsec` is locally defined.

```

257 \NewHook{cc/headings/#1/print/before}%

```

`cc/headings/[level]/run/after` is expanded after the local `RunTitle` has been generated

```

258 \NewHook{cc/headings/#1/run/after}%

```

`env/heading/[level]/begin` is the hook that is called at the begin of each defined heading level with the name `[name]`. It is called at the beginning of every `Heading` environment whose mandatory argument matches `[name]` immediately before the Instance's Attributes are evaluated.

```

259 \NewHook{env/heading/#1/begin}%
260 }

```

`\cch@init@cnt` initialises a counter with the name #1 for automatic numbering if it doesn't exist, yet.

```

261 \def\cch@init@cnt#1{\ifcsname c@#1\endcsname\else\@definecounter{#1}\fi}

```

## 1.2 Initializers for Instances of Heading Levels

`\cch@auto@number` advances the heading counter if the `numbering` Property is set to auto and the current heading is not overridden by the `Number` Component. #1 is the numeric level of the heading, #2 is the name of the heading's counter.

```

262 \def\cch@auto@number#1#2{%
263   \ccIfPropVal{numbering}{auto}
264   {\expandafter\ifx\csname c@#2\endcsname\relax\cch@init@cnt{#2}\fi
265   \ccIfAttrIsSet{Heading}{nonumber}
266   {}
267   {\ccIfComp{Number}
268   {}
269   {\ifnum #1>\c@secnumdepth\relax\else
270     \stepcounter{#2}%
271     \edef\@tempa{\csname the#2\endcsname}%
272     \ccComponentEA{Number}{\@tempa}%
273     \fi}}
274   }{}}

```

## 2 Externalisation of Heading Components

Components of headings may be used far away from the heading itself. Since, by design, Components are defined strictly local within their containers, those external usages demand special treatment.

### 2.1 Common Stuff

`\cch@set@author@name@list` sets the `#1AuthorNameList` Component.

```

275 \def\cch@set@author@name@list#1{%

```

first, we look if the Override was given in the `Heading` Container. If so, we do nothing.

```

276   \ccUnlessComp{#1AuthorNameList}{%

```

If not, we look whether or not the general `AuthorNameList` override was given in the `Heading` Container.

```

277   \ifx\cc@used@AuthorNameList@override\@empty

```

If yes, then we copy its value to `#1AuthorNameList`.

```

278     \ccComponent{#1AuthorNameList}{\cc@Heading@AuthorNameList}%
279   \else

```

Or else, we re-build the `#1AuthorNameList` from the raw `Author` Subcontainers by using the `author-list-print-format` Property.

```

280     \ifnum\ccAuthorCnt>\z@
281     \ccdefFromCountedComp\cch@tempa{Author}{author-list-print-format}%
282     \ifx\cch@tempa\relax\else
283     \ccComponent{#1AuthorNameList}{\cch@tempa}%
284     \fi
285   \fi
286 \fi
287 }{}}%

```

## 2.2 Table of Contents Entry

`\cch@make@toc` initializes the creation of a `Heading` instance's entry in the table of contents.

Each entry is in itself treated as a Container. As such, it consists of Components that are written into the .toc file.

```

288 \def\cch@make@toc{%
289   \cc@check@empty{Heading}{Title}{Toc}%
290   \cc@check@empty{Heading}{Number}{Toc}%
291   \cc@check@empty{Heading}{Subtitle}{Toc}%
292   \cch@set@author@name@list{Toc}%
293   \ccIfAttrIsSet{Heading}{notoc}{%
294     {\protected@edef\cch@toc@entry{%
295       \ccIfComp{TocTitle}{\string\ccComponent{TocTitle}{\string\ignorespaces\space\expandonce{\cc@Heading@TocTitle}}}{}%
296       \ccIfComp{TocNumber}{\string\ccComponent{TocNumber}{\string\ignorespaces\space\expandonce{\cc@Heading@TocNumber}}}{}%
297       \ccIfComp{TocAuthorNameList}{\string\ccComponent{TocAuthorNameList}{\string\ignorespaces\space\expandonce{\cc@Heading@TocAuthorNameList}}}{}%
298       \ccIfComp{TocSubtitle}{\string\ccComponent{TocSubtitle}{\string\ignorespaces\space\expandonce{\cc@Heading@TocSubtitle}}}{}%
299     }%
300     \ccIfProp{toc-level}{
301       {\edef\cch@toc@sec@name{\ccUseProperty{toc-level}}}%
302       {\let\cch@toc@sec@name\ccCurSecName}%
303       \protected@write\@auxout
304       {\ccGobble}%
305       {\string\@writefile{toc}{\protect\ccContentsline{\cch@toc@sec@name}{\cch@toc@entry}{\thepage}{\@currentHref}\protected@file@percent}}\relax
306       \ccCreateContentListEntries{Heading}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage}{\@currentHref}%
307       \ccCreateContentListEntries{\cch@toc@sec@name}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage}{\@currentHref}%
308     }}

```

`\cc@toc@extract@data` is called within the `\l@<level>` macro to extract the Components for each entry in the .toc file. #1 is the numerical heading level, #2 is the name of the heading level, #3 is the content of the toc entry (which holds the Components), #4 is the page number.

```

309 \def\cc@toc@extract@data#1#2#3#4{%
310   \ccSetContainer{Heading}%
311   \ccEvalType[#2]{Properties}%
312   \ccDeclareComponent{TocPage}{}%
313   \ccComponent{TocPage}{\ccUseProperty{toc-page-face}#4}%
314   \ccDeclareComponent{TocTitle}{}%
315   \ccDeclareComponent{TocSubtitle}{}%
316   \ccDeclareComponent{TocNumber}{}%
317   \ccDeclareComponent{TocAuthorNameList}{}%
318   \cc@expand@l@contents{#3}{Heading}{Toc}{Title}%
319   \cc@format@number{toc-}{Toc}{#1}%
320 }

```

`\cc@toc@print@entry` is also called within the `\l@<level>` macro and eventually prints the entry by expanding a `Heading`'s toc-specific Properties.

```

321 \def\cc@toc@print@entry#1{%
322   \bgroup
323   \cch@use@hook{toc/before}{#1}%
324   \ccUseProperty{toc-before-entry}%
325   \ccUseProperty{toc-format}%

```



```

326 \cch@use@hook{toc/after}{#1}%
327 \ccUseProperty{toc-after-entry}%
328 \egroup}

```

## 2.3 Facility to create the running title macros

`\cch@make@run` prepares the Components used to compose the running titles. It checks if the user provides page header specific overrides in the `Heading` instance. If not, it uses the non-specific Components instead, as long as they are not empty.

After all the header-specific Components are set, the heading level specific property `running-heading` is evaluated and passed to the corresponding `\<level>mark` macros iff they exist.

```

329 \def\cch@make@run{%
330 \cc@check@empty{Heading}{Title}{Run}%
331 \cc@check@empty{Heading}{Number}{Run}%
332 \cc@check@empty{Heading}{Subtitle}{Run}%
333 \cch@set@author@name@list{Run}%
334 \ccUseProperty{running-extra}%
335 \ccIfProp{running-level}
336   {\letcs\cch@mark@name{\ccUseProperty{running-level}mark}}
337   {\letcs\cch@mark@name{\ccCurSecName mark}}%
338   \letcs\cch@parent{cc@parent@ccCurSecName}%
339   \ifx\cch@mark@name\@undefined
340     \ifx\cch@parent\relax\else
341       \letcs\cch@mark@name{\cch@parent mark}%
342     \fi
343   \fi
344   \ifx\cch@mark@name\@undefined\else
345     \begingroup
346     \ccGobble
347     \protected@edef\@tempa{\csname cc@Heading@running-heading\endcsname}%
348     \expandafter\cch@mark@name\expandafter{\@tempa}%
349     \endgroup
350   \fi
351 }

```

## 2.4 Facility to create PDF bookmarks

`\cch@make@bookmarks` generates an entry that is directly written as Bookmark into the PDF file. This is done using the `bookmark` package.

```

352 \def\cch@make@bookmarks{%
353 \cc@check@empty{Toc}{Heading}{Title}{BM}%
354 \cc@check@empty{Toc}{Heading}{Number}{BM}%
355 \cc@check@empty{Toc}{Heading}{AuthorNameList}{BM}%
356 \cc@check@empty{Toc}{Heading}{Subtitle}{BM}%
357 \ccIfAttrIsSet{Heading}{noBM}
358   {}
359   {\ccIfProp{bookmark-level}{\edef\Hy@toclevel{\ccUseProperty{bookmark-level}}}{}%
360     \begingroup
361     \ccGobble
362     \protected@edef\@tempa{\csname cc@Heading@bookmark\endcsname}%
363     \bookmark[level=\Hy@toclevel,dest=\@currentHref]{\expandonce{\@tempa}}%
364     \endgroup
365   }}

```

## 3 Rendering the Headings

### 3.1 Inline Headings

`\cch@make@inline` Inline headings are stored in a temporary box and expanded after the next (non-heading) paragraph is opened.

```

366 \newbox\cch@inline@sec@box
367 \def\cch@make@inline{%
368   \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
369   \ccIfProp{interline-para}
370     {\global\setbox\cch@inline@sec@box\hbox{\ifvoid\cch@inline@sec@box\else\unhbox\
371       cch@inline@sec@box\ccUseProperty{interline-para-sep}\fi\svsec}}%
372     {\global\setbox\cch@inline@sec@box\hbox{\@svsec}}
373   \@nobreakfalse
374   \global\@noskipsectrue
375   \gdef\next{%
376     \global\everypar{%
377       \if@noskipsec
378         \global\@noskipsecfalse
379         {\setbox\z@\lastbox}%
380         \clubpenalty\@M
381         \begingroup
382           \unhbox\cch@inline@sec@box
383         \endgroup
384         \unskip
385         \hskip -\@tempskipa
386       \else
387         \clubpenalty \@clubpenalty
388         \global\setbox\cch@inline@sec@box\box\voidb@x
389         \everypar{}%
390       \fi}%
391   \ignorespaces}}

```

### 3.2 Block Headings

`\cch@make@block` is used to print block headings.

```

391 \def\cch@make@block{%
392   \@svsec
393   \ccUseProperty{after-heading-par}%
394   \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
395   \gdef\next{%
396     \ifdim\parskip>\z@\relax\advance\@tempskipa-\parskip\relax\fi
397     \vskip \@tempskipa
398     \@afterheading
399     \ignorespaces}}

```

## 4 The Heading environment

## 4.1 Environment Macros

`\cch@heading` is the macro called at the begin of the `Heading` environment. Optional #1 stores the headings local parameters, #2 is the level of the heading.

```
400 \def\cch@heading{\cc@opt@empty\@cch@heading}%
401 \def\@cch@heading[#1]#2{%
```

Adding start tags for the contents that “belong” to a document section. They are tagged with individual names, but all are mapped to the `<Sect>` tag.

**Warning**, the following code is incredibly ugly. In principle, we close the semi-group opened by `begin`, add the tagging, and then re-build the rest of the code from L<sup>A</sup>T<sub>E</sub>X’s standard definitions of `begin`.

This is necessary, because otherwise we would need to either manually add the starting sectioning tag outside the `\ccPrefix Heading` environment, or, if we want to keep `ltpdfa`’s `autoclose` mechanism, the sectioning tag is auto-ended at `\end{Heading}`. Using the `env/Heading/before` hook won’t work either, because at the time of its expansion, the level of the heading isn’t known, yet. So, we need to take the ugly road, for now.

```
402 \ifhmode\unskip\fi%
403 \expandafter\ifx\csname cch@notag@#2\endcsname\@empty\else
404   \ccWhenAlly{%
405     \global\let\cch@currentvir\@currentvir
406     \endgroup
407     \ifnum\cch@lowest@level=99\relax\else
408       \ccaVstructStart{#2}%
409     \fi
410     \UseHook{env/\ccPrefix Heading/before}%
411     \@ignorefalse
412     \begingroup
413     \endpfalse
414     \let\@currentvir\cch@currentvir
415     \edef\@currentvline{\on@line}%
416     \@execute@begin@hook{\ccPrefix Heading}%
417   }%
418 \fi
```

Some L<sup>A</sup>T<sub>E</sub>X kernel macros are saved, the namespace is set and counted groups from previous headings are reset.

```
419 \cch@reserve
```

`\ccCurSecName` stores the name of the current heading level.

```
420 \edef\ccCurSecName{#2}%
```

Then, we call the heading level specific `env/heading/[level]/begin` hook.

```
421 \UseHook{env/heading/#2/begin}%
```

```
422 \ccEvalAttributes[Heading]{#1}%
```

The cascaded Properties of the heading level are expanded. This is excluded into its own macro to simplify re-definition if necessary.

```
423 \ccEvalType[#2]{Components}%
424 \ignorespaces
425 }
```

`\cch@end@heading` is stuff that happens at the end of the `Heading` environment.

```

426 \def\cch@end@heading{%
427   \expandafter\ifx\curname ccUseHeading\ccCurSecName\endcurname\relax
428   \PackageError{coco-headings.sty}{Heading level \ccCurSecName\space unknown!}{A Heading with
      level \ccCurSecName\space is unknown. Use the \string\ccDeclareHeading\space macro to
      declare heading levels.}%
429   \else
430     \curname ccUseHeading\ccCurSecName\endcurname%
431   \fi
432   \cch@reset
433 }

```

## 4.2 Content Handlers

`\cch@reserve` re-directs some of L<sup>A</sup>T<sub>E</sub>X's kernel macros and makes sure that some other macros have their default values:

```

434 \def\cch@reserve{%
435   \ccSetContainer{Heading}%
436   \let\cch@ltx@dbl@backslash\\%
437   \letcs\{\ccPrefix Break}%
438   \let\cc@ltx@label\label%
439   \def\ccAuthorCnt{\z@}%
440   \def\ccAffilCnt{\z@}%
441   \cc@reset@components{\cc@cur@cont}%
442   \let\cch@current@class\relax
443   \ignorespaces}

```

`\cch@reset` restores L<sup>A</sup>T<sub>E</sub>X's default definitions (however, this should be unnecessary since `Heading` is an environment and therefore constitutes a closed group).

```

444 \def\cch@reset{%
445   \let\cc@cur@cont\relax
446   \let\\cch@ltx@dbl@backslash
447   \let\label\cc@ltx@label
448   \let\ccCurSecName\relax
449 }

```

`\cch@provide@quotes` covers multiple quotation blocks associated with a heading.

```

450 \def\cch@provide@quotes{%

```

`QuoteBlockCL` is the Collection Component for one or more ➡ `Quote` Component Groups.

```

451   \ccDeclareComponent{QuoteBlock}{-}{-}%

```

`QuoteGC` is a Component Group for quotes that belong to a heading.

```

452   \ccDeclareComponentGroup{Quote}{-}%

```

`QuoteTextCC` is the quotation text

```

453   \ccDeclareCountedComponent{QuoteText}%

```

`QuoteSourceCC` is the source of the quotation.

```
454 \ccDeclareCountedComponent{QuoteSource}%
455 }%
456 }
```

`\cch@provide@authors` sets up the additional Components for the ➡ `Author` Role specific to headings.

```
457 \def\cch@provide@authors{%
458 \ccAddToRole{Author}{%
```

`AuthorContactCC` holds the contact information of an author.

```
459 \ccDeclareCountedComponent{AuthorContact}%
460 }
```

`AuthorContactBlockCL` is the Collection Component for the Counted Component ➡ `AuthorContact`.

```
461 \ccDeclareRoleBlock{Author}{ContactBlock}{author-contact-block-format}%
```

```
462 \ccDeclareGroupHandler{Author}{%
463 \ccIfComp{AuthorContact}{%{\ccComponent{AuthorContact}{\ccUseProperty{author-contact-format
464 }}}}%
465 }
```

`AuthorNameListCL` is the Collection Component for the Author names.

```
465 \cc@provide@overrides{AuthorNameList}%
466 }
```

`\cch@provide@comp` is a wrapper that creates the user-level macros for the Component itself and its overrides. #1 is the Component name.

```
467 \def\cch@provide@comp#1{%
468 \ccDeclareComponent{#1}{-}%
469 \cc@provide@overrides{#1}%
470 }
```

`\cc@provide@overrides` declares the Component macros for a Heading Component's overrides. #1 is the Component name. The overrides allow a four-way distinction between *i* the data printed in-situ (`#1`), *ii* data sent to toc (`Toc#1`), *iii* data sent to the page styles (`Run#1`), and *iv* the data sent to the PDF bookmarks (`BM#1`).

```
471 \def\cc@provide@overrides#1{%
472 \ccDeclareComponent{Toc#1}{-}% toc overrides
473 \ccDeclareComponent{Run#1}{-}% running overrides
474 \ccDeclareComponent{BM#1}{-}% bookmark overrides
475 }
```

## 5 Defaults

```
476 \ccAddToProperties{Heading}{%
```

`interline-para` <any> is a switch that is automatically set whenever an inline heading is not-yet sent to the vertical list and another inline heading is processed.

```
477 \ccSetProperty{interline-para}{}%
```

`interline-para-sep` <any> is the material that is printed between to inline headings.

```
478 \ccSetProperty{interline-para-sep}{\space}
```

`heading-par` <any> is the material added to the very beginning of a heading.

```
479 \ccSetProperty{heading-par}{%
480   \ccIfProp{interline-para}{\if@noskipsec \leavevmode \fi}{}%
481   \par
482   \global\@afterindenttrue
483 }%
```

`after-heading-par` <any> is expanded at the very end of non-inline headings.

```
484 \ccSetProperty{after-heading-par}{\par \nobreak}%
```

`before-heading` <any> is expanded immediately before any vertical skips of a heading are inserted, but after the `\begin-hook`.

```
485 \ccSetProperty{before-heading}{}%
```

`title-face` <any> is the style of the heading's main title.

```
486 \ccSetProperty{title-face}{\bfseries}%
```

`subtitle-face` <any> is the style of the heading's subtitle.

```
487 \ccSetProperty{subtitle-face}{\normalfont}%
```

`author-face` <any> is the face of the heading's printed Author Component.

```
488 \ccSetProperty{author-face}{\normalfont}%
```

`quote-face` <any> is the style of a quotation.

```
489 \ccSetProperty{quote-face}{\raggedleft}%
```

`quote-source-face` <any> is the style of a quotation's source line.

```
490 \ccSetProperty{quote-source-face}{}%
```

`quote-block-format` <any> is the format of a single quotation. By default, it uses the `\QuoteText` and `\QuoteSource` Components.

```
491 \ccSetProperty{quote-block-format}{%
492   \bgroup
493   \ccUseProperty{quote-face}%
494   \ccUseComp{QuoteText}\par
495   \ccIfComp{QuoteSource}{\ccUseProperty{quote-source-face}--\space\ccUseComp{QuoteSource}}\
    par}{}%
496 \egroup}
```

**heading-block** <any> is the format of the main heading. It uses the ➤ Subtitle, ➤ AuthorNameList, ➤ QuoteBlock and ➤ AffilBlock Components.

```

497 \ccSetProperty{heading-block}
498   {\ccUseProperty{main-title-format}%
499    \ccIfComp{Subtitle}{\{\ccUseProperty{subtitle-face}\ccUseComp{Subtitle}\}\par}{}%
500    \ccIfComp{AuthorNameList}{\{\ccUseProperty{author-face}\ccUseComp{AuthorNameList}\}\par}{}%
501    \ccIfComp{QuoteBlock}{\ccUseComp{QuoteBlock}\}%
502    \ccIfComp{AffilBlock}{\{\ccUseProperty{affil-block-face}\ccUseComp{AffilBlock}\}\par}{}%
503   }%

```

**main-title-format** <any> is the format of the heading's main title. It should also enclose the heading's ➤ Number and ➤ Title Components with Tags that are mapped to <H/> or <Hn/> with  $1 < n < 6$ .

```

504 \ccSetProperty{main-title-format}{%
505   \ccUseProperty{title-face}%
506   \cchHeadTagStart
507   \ccIfComp{Number}%
508   {\ccUseProperty{hang-number}}%
509   {\leftskipOpt}%
510   \ccUseComp{Title}%
511   \cchHeadTagEnd
512   \par
513 }

```

**extended-heading** <any> is the format of extended headings which incorporates the ➤ Abstract and ➤ Keywords Labeled Components. Requires the ⚙ extended Property to be non-empty.

```

514 \ccSetProperty{extended-heading}{%
515   \ccIfComp{Abstract}
516   {\par\vskip\baselineskip
517    {\bfseries\ccIfComp{AbstractLabel}{\ccUseComp{AbstractLabel}}{Abstract}}\par
518    {\itshape\small\ccUseComp{Abstract}}\par}
519   {}%
520   \ccIfComp{Keywords}
521   {\par\vskip\baselineskip
522    {\bfseries\ccIfComp{KeywordsLabel}{\ccUseComp{KeywordsLabel}}{Keywords}}\par
523    {\itshape\small\ccUseComp{Keywords}}\par}
524   {}%
525 }%

```

**before-skip** <skip> the vertical space before heading. Positive values are set with L<sup>A</sup>T<sub>E</sub>X's \addvspace, while negative values are set with \minusvspace.

**TODO**  
values < 0pt use  
\minusvspace, else  
\addvspace. L<sup>A</sup>T<sub>E</sub>X's  
default behaviour of  
\@afterindent is  
relocated to the  
after-indent property.

```

526 \ccSetProperty{before-skip}{\z@skip}%

```

**after-heading-block** <any> is expanded at the very end of the printed heading.

```

527 \ccSetProperty{after-heading-block}{}%

```

**before-heading-block** <any> is expanded at the very beginning of @svsec.


```
528 \ccSetProperty{before-heading-block}{\parindent\z@ \parskip\z@}%
```

**toc-hook** <any> is called after ToC and Bookmark entries are written and allows for material to be added to the toc file.

```
529 \ccSetProperty{toc-hook}{}% Called, after ToC and BM entries have been written to the .aux file
```

**after-indent** <any> if non-empty, the first paragraph after the heading will be indented.

```
530 \ccSetProperty{after-indent}{}%
```

**margin-left** [auto|<dimen>|<empty>] is the left margin of the heading. Its value can either be a fixed dimension, the string auto, or empty. If the Property is set to auto or an empty string, the margin is calculated from the  **indent** (see below). Otherwise the fix value is used.


```
531 \ccSetProperty{margin-left}{}%
```


**margin-right** <skip> is the right margin of the heading block.

```
532 \ccSetProperty{margin-right}{\@flushglue}%
```

**after-skip** <skip> is the vertical space after the heading block. If the value is greater than or equal to 0pt, the heading is formatted in block, while it is formatted as inline heading if the value is negative.

```
533 \ccSetProperty{after-skip}{1sp}%
```

**indent** [auto|auto-global|<dimen>] is the offset of the first line of the heading relative to  **margin-left**.

If the value is auto, the indent of the heading is the width of the widest  **Number** Component of *all headings with the same level*.

If the value is auto-global, the indent is the width of the widest Number component across *all heading levels*. Both auto and auto-global require at least two L<sup>A</sup>T<sub>E</sub>X runs. See Sect. 3.3 in Module Module 3 for more details.

```
534 \ccSetProperty{indent}{auto}%
```

**number-width** <dimen> is the (actula) width of the Number component.





```
535 \ccSetProperty{number-width}{}%
```

**number-sep** <any> Is the separator between the Number and the Title components

```
536 \ccSetProperty{number-sep}{\space}%
```

**number-align** [left|center|right] is the horizontal alignment of the Number component inside its surrounding **\hbox**.

```
537 \ccSetProperty{number-align}{left}%
```

**number-format** <any> is the format of a heading's counter. It prints the  **Number** component and the  **number-sep** Property, and stylizes them both with the  **title-face** and  **number-face** Properties.

```
538 \ccSetProperty{number-format}{}%
539 \bgroup
540 \ccUseProperty{title-face}%
541 \ccUseProperty{number-face}%
542 \ccUseComp{Number}%
543 \ccUseProperty{number-sep}%
544 \egroup
```



**numbering** [auto|<any>] if non-auto, headings are not numbered automatically if no Number component is given. This property can be overridden in a local instance with the **nonumber** Attribute.

```
545 \ccSetProperty{numbering}{auto}%
```

**running-level** <name> is an override that allows the heading's running title to appear as another level's running title. Usually, the **RunTitle** Component is passed to **\<level>mark** for the page header, but if this Property is non-empty, the heading will be passed to **\<running-level>mark**, instead.

```
546 \ccSetProperty{running-level}{}% override level for running title, name
```

**running-heading** <any> is the format of the material passed to the **\<level>mark** or **\<running-level>mark** command. It uses the **RunTitle** and **RunAuthorNameList** Components.

```
547 \ccSetProperty{running-heading}{%
548   \ccIfComp{RunAuthorNameList}{\ccUseComp{RunAuthorNameList}:\space}{}%
549   \ccUseComp{RunTitle}%
550 }%
551 %% ToC
```

**no-toc** [true|false] whether or not the heading does *not* create an entry in the table of contents (true means no toc entry, false means toc entry).

```
552 \ccSetProperty{no-toc}{false}% toc entries are generally disabled iff true
```

**no-BM** [true|false] whether or not the heading does *not* create a bookmark (true means no bookmark, false means bookmark).

```
553 \ccSetProperty{no-BM}{false}% bookmark entries are generally disabled, iff true
```

**toc-margin-top** <skip> vertical space before the ToC entry.

```
554 \ccSetProperty{toc-margin-top}{\z@}% left indent of the whole entry
```

**toc-margin-bottom** <skip> vertical space after the ToC entry.

```
555 \ccSetProperty{toc-margin-bottom}{\z@}% bottom margin of the whole entry
```

**toc-margin-left** [auto|<dimen>] left margin of the toc entry. See **margin-left** for the meaning of auto.

```
556 \ccSetProperty{toc-margin-left}{auto}% left indent of the whole entry
```

**toc-margin-right** <dimen> right margin of the ToC entry.

```
557 \ccSetProperty{toc-margin-right}{\@pnumwidth}% right margin of the whole entry
```

**toc-title-face** <any> style of the title in the ToC entry.

```
558 \ccSetProperty{toc-title-face}{}% appearance of title
```

**toc-indent** [auto|auto-global|<dimen>] offset of the ToC entry's first line relative to margin-left. See **indent**.

```
559 \ccSetProperty{toc-indent}{auto}%
```

**toc-number-width** <dimen> the actual width of the TocNumber Component.

```
560 \ccSetProperty{toc-number-width}{}% current width of the TocNumber
```

`toc-number-align` [left|center|right] the alignment of the `TocNumber` within the surrounding `\hbox`.

```
561 \ccSetProperty{toc-number-align}{left}% alignment of TocNumber within the hbox when hanging
```

`toc-number-face` <any> style of the `TocNumber` component.

```
562 \ccPropertyLet{toc-number-face}{toc-title-face}% appearance of the TocNumber
```

`toc-number-sep` <any> separator between the `TocNumber` and `TocTitle` Components

```
563 \ccSetProperty{toc-number-sep}{\enskip}% thing between TocNumber and TocTitle
```

`toc-number-format` <any> is the format of the `TocNumber` Component, using the `toc-number-face` and `toc-number-sep` Properties.

```
564 \ccSetProperty{toc-number-format}{% Format of the TocNumber
565   \bgroup
566   \ccUseProperty{toc-number-face}%
567   \ccUseComp{TocNumber}%
568   \ccUseProperty{toc-number-sep}%
569   \egroup}
```

`toc-page-sep` <any> separator between the `TocTitle` and the page counter.

```
570 \ccSetProperty{toc-page-sep}{\dotfill}% between TocTitle and the page counter
```

`toc-page-face` <any> style of the page counter

```
571 \ccSetProperty{toc-page-face}{}% appearance of the page value
```

`toc-page-format` <any> format of the page counter using the `toc-page-sep` and `toc-page-face` Properties.

```
572 \ccSetProperty{toc-page-format}{% format of the page value
573   \ccUseProperty{toc-page-sep}%
574   \bgroup
575   \ccUseProperty{toc-page-face}%
576   \ccUseComp{TocPage}%
577   \egroup}%
```

`toc-level` <name> name of another heading level as which the ToC entry should be rendered.

```
578 \ccSetProperty{toc-level}{}%
```

`toc-before-entry` <any> is expanded before any ToC entry is rendered. Should setup margins, alignment, line-breaking rules, etc.

```
579 \ccSetProperty{toc-before-entry}{%
580   \addvspace{\ccUseProperty{toc-margin-top}}%
581   \parindent \z@
582   \let\\\@centercr
583   \hyphenpenalty=\@M
584   \rightskip \ccUseProperty{toc-margin-right} \@plus 1fil\relax
585   \parfillskip -\rightskip
586   \leftskip\ccUseProperty{toc-margin-left}%
587   }%
```

`toc-after-entry` <any> is expanded at the very end of a ToC entry. By default, it sets the skip after the entry to `toc-margin-bottom`.

```
588 \ccSetProperty{toc-after-entry}{\par\addvspace{\ccUseProperty{toc-margin-bottom}}}%
```

**toc-format** <any> format of the ToC entry itself. It uses the **toc-title-face**, **toc-hang-number** and **toc-page-format** Properties to print the **TocNumber**, **TocAuthorNameList**, **TocTitle**, and **TocPage** Components. Tagging should incorporate the **<TOCI/>**, **<P/>**, and **<Reference/>** tags for the entire entry, as well as **<Lbl/>** for the **TocNumber**, and **<Span/>** for the rest of the entry.

```

589 \ccSetProperty{toc-format}{%
590   \ccUseProperty{toc-title-face}%
591   \ccaVstructStart{TOCI}%
592   \ccIfComp{TocNumber}
593     {\ccaStructStart{P}\ccaStructStart{Reference}\ccaStructStart{Lbl}\ccUseProperty{toc-hang-
        number}\ccaStructEnd{Lbl}}
594     {\leftskipOpt\leavevmode}%
595   \ccaVstructStart{Span}%
596   \ccTocLink{%
597     \ccWhenComp{TocAuthorNameList}{\ccUseComp{TocAuthorNameList}:\space}%
598     \ccUseComp{TocTitle}%
599     \ccUseProperty{toc-page-format}%
600   }%
601   \ccaVstructEnd{Span}%
602   \ccWhenComp{TocNumber}{\ccaStructEnd{Reference}\ccaStructEnd{P}}%
603   \ccaVstructEnd{TOCI}%
604 }%
```

**bookmark-level** <num> number(!) of the heading level as which the Bookmark entry should be rendered.

```

605 \ccSetProperty{bookmark-level}{}%
```

**bookmark** <any> is the format of the bookmark, which by default is built only from the **BMNumber** and **BMTitle** Components.

```

606 \ccSetProperty{bookmark}{%
607   \ccIfComp{BMNumber}{\ccUseComp{BMNumber}\space}{}%
608   \ccUseComp{BMTitle}%
609 }%
```

**orcid-link** <any> how an **ORCID** link is rendered.

```

610 \ccSetProperty{orcid-link}{%
611   \ccIfComp{ORCID}{\ccCompLink{ORCID}{\includegraphics[height=1em]{logos/ORCID.pdf}}}{}%
612 }%
```

**author-contact-format** <any> how a single Author Component's contact information should be rendered. By default, it uses the Author's **FullName**, the value of the **AffilRef** component as superscript, and the **orcid-link** Property.

```

613 %% a single Author's contact information block
614 \ccSetProperty{author-contact-format}{%
615   \ccUseComp{FullName}\ccWhenComp{RefAffil}{\textsuperscript{\ccUseComp{AffilRef}}}%
616   \ccUseProperty{orcid-link}%
617 }%
```

**author-list-format** <any> how a single entry in the **AuthorNameList** Collection Component should be rendered.

```

618 \ccPropertyLet{author-list-format}{author-list-print-format}%
```

**author-contact-block-format** <any> is the Collection Property for the **AuthorContactBlock** Collection Component and sets how each single entry in the Collection should be formatted. By default, it uses the **AuthorContact** Counted Component and appends the **counted-name-sep** to all instance of that Component but the last.

```

619 \ccSetProperty{author-contact-block-format}{% Format of the whole contact information block
620 \ccUseComp{AuthorContact}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi
621 }}

```

`\cchHeadTagStart` is used inside Property declarations to insert the start tag of the current section, unless the heading level was declared with the starred version of `\ccDeclareHeading`.

```

622 \def\cchHeadTagStart{%
623 \expandafter\ifx\csname cch@notag@\ccCurSecName\endcsname\@empty\else
624 \ccaVstructStart{\ccCurSecName head}%
625 \fi}

```

`\cchHeadTagEnd` is used inside Property declarations to insert the end tag of the current section, unless the heading level was declared with the starred version of `\ccDeclareHeading`.

```

626 \def\cchHeadTagEnd{%
627 \expandafter\ifx\csname cch@notag@\ccCurSecName\endcsname\@empty\else
628 \ccaVstructEnd{\ccCurSecName head}%
629 \fi}

```

## 6 Miscellaneous

### 6.1 Alternative paragraph separation

`\ccNewPar` is a user-level macro to have a vertical skip between two local paragraphs and no indent in the second one. The amount of vertical space between the paragraphs can be adjusted with the optional argument. If #1 is omitted, `\ccnewparskip` is inserted, which defaults to `1\baselineskip` if the dimension isn't set to something other than `Opt` in the preamble. This macro is intended to be used at the end of the first of the paragraphs.

```

630 \newdimen\ccnewparskip \AtBeginDocument{\ifdim\ccnewparskip=\z@\relax \ccnewparskip=1\
631 \baselineskip\relax\fi}
632 \def\ccNewPar{\@ifnextchar[{\cc@newpar[\the\ccnewparskip]}}%]
633 \def\cc@newpar[#1]{%
634 \ifhmode\par\fi
635 \vskip#1\relax
636 \afterheading
637 }
638 \cslet{\ccPrefix NewPar}\ccNewPar

```

**WARNING!**  
The following section is deprecated and will be changed or deleted in future releases.

`\TitleBreak`

```

638 \letcs\TitleBreak{\ccPrefix Break}

```

`</headings>`

## Module 7

# coco-notes.dtx

```
<*endnotes>
```

This file contains the code for foot- and endnote handling. It provides a switch between endnotes and footnotes as well as options to handle the resetting of footnote/endnote counters.

```
23 %%
24 %% module for CoCoTeX that handles footnote/endnote switching.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-notes}
32 [2024/07/16 0.5.0 le-tex coco notes module]
```

## 1 Internal Switches and Package Options

### 1.1 Package Switches

`\ifccn@use@en` is an internal switch for endnotes (`\ccn@use@entrue`) or footnotes (`\ccn@use@enfalse`, default).

```
33 \newif\ifccn@use@en \ccn@use@enfalse
```

`\ifccn@en@links` is an internal switch for back-referencing (`\ccn@en@linkstrue`) or plain endnotes (`\ccn@en@linksfalse`).

```
34 \newif\ifccn@en@links \ccn@en@linksfalse
```

### 1.2 Package Options

The `endnotes` option causes all footnotes to be rendered as endnotes.

```
35 \DeclareOption{endnotes}{\global\ccn@use@entrue}
```

The option `ennotoc` prevents headings in the Notes section from creating entries in the Table of Contents.

```
36 \DeclareOption{ennotoc}{\global\let\ccn@en@no@toc\relax}
```

The option `resetnotesperchapter` resets foot- and endnote counters at the start of each `chapter` level heading. If omitted (default) foot- or endnotes are numbered throughout the whole document

```
37 \DeclareOption{resetnotesperchapter}{\global\let\ccn@reset@notes@per@chapter\relax}
```

The option `endnoteswithchapters` implies `endnotes` and allows the output of all collected endnotes at the end of each chapter. It also sets the note's heading to the level `section` (otherwise it is `chapter`).

```
38 \DeclareOption{endnoteswithchapters}{\global\@ccn@use@entrue\global\let\ccn@en@with@chapters\relax}
```

The option `endnotelinks` causes endnotes to back-reference to their Reference in the main text body.

```
39 \DeclareOption{endnotelinks}{\global\@ccn@en@linkstrue}
40 \ProcessOptions
```

## 1.3 Hard Requirements

The `footnote` package is mandatory since it provides the `\savenotes` and `\spewnotes` macros needed in other CoCoTeX modules.

```
41 \RequirePackage{footnote}
```

## 2 Endnote Handling

`\if@enotesopen` is a switch from the `endnotes` package. but since the package is loaded only with the `endnotes` options set, we need to define the conditional, anyhow.

```
42 \newif\if@enotesopen
```

`\ccn@parindent` is the par indent used in the endnotes section. It defaults to the value of `parindent` at the very end of the L<sup>A</sup>T<sub>E</sub>X Preamble.

```
43 \AtBeginDocument{\edef\ccn@parindent{\the\parindent}}
```

If endnotes are activated via a Package option, we include the `endnotes` package.

```
44 \if@ccn@use@en
45 \RequirePackage{endnotes}
```

`\ccn@use@TeX@heading` is a switch that defines itself when the CoCoTeX Headings module is loaded.

```
46 \@ifpackageloaded{coco-headings}{\let\ccn@use@TeX@heading\relax}{}%
```

`\@endnotemark` is re-defined when endnotes should back-reference. In this case, we insert a L<sup>A</sup>T<sub>E</sub>X `\label` for later referencing.

**TODO**  
This macro should be patched, not re-defined!

```
47 \if@ccn@en@links
48 \global\newcount\endnoteLinkCnt \global\endnoteLinkCnt\z@
49 \def\@endnotemark{%
50 \leavevmode
```

```

51 \ifhmode\edef\@x@sf{\the\spacefactor}\nobreak\fi
52 \phantomsection%
53 \label{endnote-\the\endnoteLinkCnt}%
54 \hyperref[endnotetext-\the\endnoteLinkCnt]{\makeenmark}%
55 \ifhmode\spacefactor\@x@sf\fi%
56 \relax%
57 }
58 \fi

```

`\footnote` is re-defined to be an alias of the `\endnote` macro.

```

59 \let\footnote=\endnote

```

`\enotesize` holds the font size of the endnotes section.

```

60 \def\enotesize{\normalsize}%

```

`\enoteformat` is the format of an endnote. We create the label right at the start of the endnote text to prevent erroneous pointing to the next page.

```

61 \def\enoteformat{%
62 \if@ccn@en@links%
63 \phantomsection%
64 \label{endnotetext-\currentEndnote}%
65 \fi
66 \noindent
67 \leavevmode
68 \hskip-2em\hb@xt@2em{%
69 \if@ccn@en@links
70 \hyperref[endnote-\currentEndnote]{\@theenmark}\hss%
71 \else
72 \@theenmark\hss%
73 \fi%
74 }%
75 \expandafter\parindent\ccn@parindent\relax\expandafter%
76 }%

```

`\enoteheading` is a macro that is expanded at the beginning of the endnotes section. Originally, it was intended to hold the endnote section's heading, we mis-use it to set the leftskip. Apparently, the intention is to re-define the macro style-wise if needed...

```

77 \gdef\enoteheading{%
78 \leftskip2em
79 }%

```

`\printnotes` is the macro that eventually prints the endnote section in its stead.

```

80 \def\printnotes{%
81 \ifx\ccn@en@with@chapters\relax
82 \ifnum\c@endnote>\z@
83 \expandafter\global\expandafter\let\csname enotes@in@\the\realchap\endcsname\@empty
84 \fi
85 \fi
86 \if@enotesopen
87 \global\c@endnote\z@%
88 \bgroup
89 \parskip\z@
90 \theendnotes

```

```

91     \egroup
92     \fi}
93 \else

```

`\c@endnote` is defined to ensure upward-compatibility.

```

94 \newcount\c@endnote \c@endnote\z@
95 \let\printnotes\relax
96 \fi

```

## 3 Processing Package Options

### 3.1 Endnotes With Chapters

If endnotes are printed chapter-wise, we need to hook into the `chapter` heading level using `\before-hook-chapter`. There, we check if the last chapter did actually contain endnotes. If yes, we pass the chapter's `Title` and `RunTitle` components into the endnote temporary `.ent` file as a `section` heading.

```

97 \newcount\realchap \realchap\z@
98 \ifx\ccn@en@with@chapters\relax
99   \AtBeginDocument{%
100     \AddToHook{env/heading/chapter/begin}{%
101       \ifnum\c@endnote>\z@\relax
102         \expandafter\global\expandafter\let\csname enotes@in@the\realchap\endcsname\@empty
103       \fi
104       \global\advance\realchap\@ne
105       \global\c@endnote\z@
106       \def\ccn@par@title{\ccIfComp{TocTitle}{\ccUseComp{TocTitle}}{\ccUseComp{Title}}}%
107       \def\ccn@par@runtitle{\ccIfComp{RunTitle}{\ccUseComp{RunTitle}}{\ccUseComp{Title}}}%
108       \addtoendnotes{%
109         \noexpand\expandafter\noexpand\ifx\noexpand\csname enotes@in@the\realchap\noexpand\
110           endcsname\noexpand\@empty
111           \bgroup
112             \noexpand\leftskip\noexpand\z@
113             \noexpand\begin{heading}\ifx\ccn@en@no@toc\relax[notoc]\fi{section}%
114             \noexpand\ccComponent{Title}{\ccn@par@title}%
115             \noexpand\ccComponent{RunTitle}{\ccn@par@runtitle}%
116             \noexpand\end{heading}%
117           \egroup
118         \noexpand\fi}%
119     }%
120 \fi

```

### 3.2 Chapter-wise Resetting

```

121 \ifx\ccn@reset@notes@per@chapter\relax
122   \AtBeginDocument{%
123     \AddToHook{env/heading/chapter/begin}{%
124       \global\c@footnote\z@
125       \global\c@endnote\z@
126     }%
127   }%
128 \fi

```



### 3.3 Back-Referencing Endnotes

Linking endnotes requires overwriting the `\@endnotetext` macro to save a global counter to the \*.ent file.

```

129 \global\newif\if@haveenotes
130 \long\def\@endnotetext#1{%
131   \global\@haveenotesttrue
132   \if@notesopen \else \@openenotes \fi
133   \immediate\write\@enotes{%
134     \ifccn@en@links
135       \string\def\string\currentEndnote{\the\endnoteLinkCnt}%
136     \fi%
137     \@doanenote{\@theenmark}%
138   }%
139   \begingroup
140     \def\next{#1}%
141     \newlinechar='40
142     \immediate\write\@enotes{\meaning\next}%
143   \endgroup
144   \immediate\write\@enotes{\@endanenote}%
145   \ifccn@en@links
146     \global\advance\endnoteLinkCnt\@ne%
147   \fi%
148 }
```

## 4 Adjusting Regular Footnotes

### 4.1 Allowing Multiple Paragraphs in Footnotes

First, we make a small adjustment to the `\fn@fntext` macro from the `footnote` package by making it `\long` and therefore allowing `\par` inside its argument.

```

149 \long\def\fn@fntext#1{%
150   \ifx\ifmeasuring@\@undefined%
151     \expandafter\@secondoftwo\else\expandafter\@iden%
152   \fi%
153   {\ifmeasuring@\expandafter\@gobble\else\expandafter\@iden\fi}%
154   {%
155     \global\setbox\fn@notes\vbox{%
156       \unvbox\fn@notes%
157       \fn@startnote%
158       \@makefntext{%
159         \rule\z@\footnotesep%
160         \ignorespaces%
161         #1%
162         \@finalstrut\strutbox%
163       }%
164       \fn@endnote%
165     }%
166   }%
167 }
```

## 4.2 Allowing Non-Numeric Footnote Counters

Re-definition of `footnote` package's footnote mark retriever to allow non-numeric values in the optional argument of `\footnote`.

```

168 \def\fn@getmark@i#1[#2]{%
169   \sbox\z@{\@tempcnta0#2\relax}%
170   \ifdim\wd\z@>0\p@\relax
171     \def\thempfn{#2}%
172     \fn@getmark@iii%
173   \else
174     \csname c@\@mpfn\endcsname#2%
175     \fn@getmark@ii%
176   \fi
177 }
178 \def\fn@getmark@iii#1{%
179   \unrestored@protected@xdef\@thefnmark{\thempfn}%
180   \endgroup%
181   #1%
182 }

```

And the same for plain  $\text{\LaTeX}$ :

```

183 \def\@xfootnote[#1]{%
184   \begingroup
185     \sbox\z@{\@tempcnta0#1\relax}%
186     \ifdim\wd\z@>0\p@\relax
187       \unrestored@protected@xdef\@thefnmark{#1}%
188     \else
189       \csname c@\@mpfn\endcsname #1\relax
190       \unrestored@protected@xdef\@thefnmark{\thempfn}%
191     \fi
192   \endgroup
193   \@footnotemark\@footnotetext%
194 }

```

## 5 Tagging Footnotes

Adding artifact tagging to the footnoterule:

```

195 \pretocmd\footnoterule{\ccaVstructStart[document]{footnoterule}}{}{}
196 \apptocmd\footnoterule{\ccaVstructEnd{footnoterule}}{}{}

```

patching `\@footnotemark` to introduce the `<FootnoteMark/>` tag which will be mapped to the `<Reference/>` tag, later.

```

197 \pretocmd\@footnotemark{%
198   \protected@xdef\@lt@fn@parent{\ccaGetCurStruct{idx}}%
199   \ccaStructStart{FootnoteMark}%
200 }{}{}
201 \apptocmd\@footnotemark{%
202   \ccaStructEnd{FootnoteMark}%
203 }{}{}

```

patching `\@makefnmark` for the `<Lb1/>` tag both in the text body and in the footnote insert.

```

204 \pretocmd\@makefnmark{%

```

```

205 \ccaStructStart{Lbl}%\addAltText{\@thefnmark}
206 }{}{}
207 \apptocmd\@makefnmark{%
208 \ccaStructEnd{Lbl}%\addAltText{\@thefnmark}
209 }{}{}

```

patching `\@makefnmark` to introduce the `<FootnoteText/>` tag, which will be mapped to `<Note/>`, below.

```

210 \pretocmd\@makefnmark{%
211 \ccaStructStart{FootnoteText}%
212 \ifx\@lt@fn@parent\empty\relax\else\ccaAddToStruct{\@lt@fn@parent}\fi%
213 }{}{}
214 \apptocmd\@makefnmark{%
215 \ccaAddID{auto}\ccaStructEnd{FootnoteText}%
216 }{}{}

```

Finally, we add the `<FootnoteMark/>` and `<FootnoteText/>` PDF tags to the rolemap.

```

217 \ccaAddRolemap{FootnoteMark}{Reference}
218 \ccaAddRolemap{FootnoteText}{Note}

```

```

</endnotes>

```



## Module 8

# coco-script.dtx

```
<*script>
```

This package is used to handle non-latin based script systems like Japanese, Chinese, Armenian and the like.

```
23 %% module for CoCoTeX that handles script switching.
24 %%
25 %% Maintainer: p.schulz@le-tex.de
26 %%
27 %% lualatex - texlive > 2019
28 %%
29 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
30 \ProvidesPackage{coco-script}
31 [2024/07/16 0.5.0 CoCoTeX script module]
```

The argument of the `usescript` option is a list of script systems that are used in the document. It is used to determine the additional fonts that are to be loaded via the babel package.

```
32 \RequirePackage{coco-kernel}
33 \RequirePackage{xkeyval}
34 \let\usescript\relax
35 \define@key{coco-script.sty}{usescript}{\def\usescript{#1}}
36 \ProcessOptionsX
37 \RequirePackage[quiet]{fontspec}
38 \RequirePackage[bidi=basic,silent]{babel}
39 \def\parse@script#1,#2,\relax{%
40   \ccs@callback{#1}%
41   \edef\@argii{#2}%
42   \let\next\relax
43   \ifx\@argii@empty\else
44     \def\next{\parse@script#2,\relax}%
45   \fi\next}
46 \ifx\usescript\relax\else
47   \def\ccs@callback#1{\expandafter\global\expandafter\let\csname use@script@#1\endcsname\@empty}
48   \expandafter\parse@script\usescript,,\relax
49 \fi
50 \ccPackageInfo{Script}{Info}{Fonts loaded: \meaning\usescript}
```

## 1 Fallback fonts

### 1.1 Default Fallback Font: Noto

The default fall backfont is the NotoSans Font Family

```
51 \newfontfamily\fallbackfont{NotoSerif-Regular.ttf}%
```

```

52 [BoldFont = NotoSerif-Bold.ttf,%
53 ItalicFont = NotoSerif-Italic.ttf,%
54 BoldItalicFont = NotoSerif-BoldItalic.ttf,%
55 Path = ./fonts/Noto/Serif/,%
56 WordSpace = 1.25]
57 \newfontfamily\sffallbackfont{NotoSans-Regular.ttf}%
58 [BoldFont = NotoSans-Bold.ttf,%
59 ItalicFont = NotoSans-Italic.ttf,%
60 BoldItalicFont = NotoSans-BoldItalic.ttf,%
61 Path = ./fonts/Noto/Sans/,%
62 WordSpace = 1.25]
63 \DeclareTextFontCommand\textfallback{\fallbackfont}
64 \DeclareTextFontCommand\textsfallback{\sffallbackfont}

```

## 1.2 Emojis

A font and a text command for using plain, black emojis.

```

65 \newfontfamily\emojifont{NotoEmoji-Regular.ttf}%
66 [BoldFont = NotoEmoji-Bold.ttf,%
67 Path = ./fonts/Noto/Emoji/]
68 \DeclareTextFontCommand\textemoji{\emojifont}

```

## 1.3 Support for medieval scripts and special characters

**Warning:** Junicode provides supports only for the `rm` font family!

```

69 \babelfont{mdv}{%
70 Path=fonts/Junicode/,%
71 ItalicFont = Junicode-Italic.ttf,%
72 BoldFont = Junicode-Bold.ttf,%
73 BoldItalicFont = Junicode-BoldItalic.ttf,%
74 }{Junicode.ttf}
75 \def\mdvfont#1{{\mdvfamily#1}}

```

# 2 Generic Fonts Declaration Mechanism

- #1 Options passed to `\babelprovide`
- #2 language
- #3 argument(s) passed to `\babelfont{rm}`
- #4 argument(s) passed to `\babelfont{sf}`

```

76 \def\ccDeclareBabelFont{\cc@opt@empty\ccs@declare@babel@font}%
77 \def\ccs@declare@babel@font[#1]#2#3#4{%
78   \expandafter\ifx\csname use@script@#2\endcsname\@empty
79     \babelprovide[#1]{#2}%
80     \message{^^J [coco-script Loaded Script: #2]^^J}%
81     %%
82     \expandafter\gdef\csname ccs@babel@rm@font@#2\endcsname{#3}%
83     \expandafter\gdef\csname ccs@babel@sf@font@#2\endcsname{#4}%
84     \if!#2!\else
85       \def\ccs@tempa{\babelfont[#2]{rm}}%
86       \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@rm@font@#2\endcsname

```

```

87 \fi
88 \if!#3!\else
89 \def\ccs@tempa{\babelfont[#2]{sf}}%
90 \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@sf@font@#2\endcsname
91 \fi
92 \fi
93 }

```

Top level macro to declare a font alias.

#1 font family alias  
#2 font family fallback

```

94 \def\ccBabelAlias#1#2{%
95 \ifx\usescript\relax\else
96 \def\ccs@callback##1{%
97 \expandafter\ifx\csname ccs@no@fallback@##1\endcsname\relax
98 \expandafter\ifx\csname ccs@babel@#2@font@##1\endcsname\relax
99 \PackageError
100 {coco-script.sty}
101 {\expandafter\string\csname #2family\endcsname\space for Language `##1' was not
102 declared!}
103 {You attempted to declare an alias towards a font family that has not been declared
104 for the language `##1', yet.}%
105 \else
106 \def\ccs@tempa{\babelfont[##1]{#1}}%
107 \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@#2@font@##1\endcsname
108 \fi
109 \else
110 \PackageInfo{coco-script.sty}{^^J\space\space\space\space No fallback for `##1';^^J\space
111 \space\space\space\space Skipping font family `#1'->`#2'}%
112 \fi}%
113 \expandafter\parse@script\usescript,,\relax
114 \fi}

```

## 3 Predefined script systems

### 3.1 Support for Armenian script

```

112 \ifx\use@script@armenian\@empty
113 \message{^^J [coco-script Loaded Script: Armenian]^^J}
114 \def\NotoArmenianPath{./fonts/Noto/Armenian/}
115 \newfontfamily\fallbackfont@armenian{NotoSansArmenian-Regular.ttf}%
116 [BoldFont = NotoSansArmenian-Bold.ttf,%
117 Path = \NotoArmenianPath,%
118 WordSpace = 1.25]
119 \DeclareTextFontCommand\armenian{\fallbackfont@armenian}
120 \let\ccs@no@fallback@armenian\@empty%
121 \fi

```

### 3.2 Support for Chinese script

```

122 \ccDeclareBabelFont{chinese}{[%
123 Path=./fonts/Noto/Chinese/,

```

```

124     BoldFont = NotoSerifSC-Bold.otf,%
125     WordSpace = 1.25]{NotoSerifSC-Regular.otf}}
126   {[%
127     Path=./fonts/Noto/Chinese/,
128     BoldFont = NotoSansSC-Bold.otf,%
129     WordSpace = 1.25]{NotoSansSC-Regular.otf}%
130   }

```

### 3.3 Support for Japanese script

```

131 \ccDeclareBabelFont{japanese}{[%
132     Path=./fonts/Noto/Japanese/,
133     BoldFont = NotoSerifJP-Bold.otf,%
134     WordSpace = 1.25]{NotoSerifJP-Regular.otf}
135 }{[%
136     Path=./fonts/Noto/Japanese/,
137     BoldFont = NotoSansJP-Bold.otf,%
138     WordSpace = 1.25]{NotoSansJP-Regular.otf}
139 }

```

### 3.4 Support for Korean script

```

140 \ccDeclareBabelFont{korean}{[%
141     BoldFont = NotoSerifKR-Bold.otf,%
142     ItalicFont = NotoSerifKR-Regular.otf,%
143     BoldItalicFont = NotoSerifKR-Medium.otf,%
144     Path=./fonts/Noto/Korean/,%
145     Script=CJK%
146   ]{NotoSerifKR-Regular.otf}}
147 {[%
148     BoldFont = NotoSansKR-Bold.otf,%
149     ItalicFont = NotoSansKR-Regular.otf,%
150     BoldItalicFont = NotoSansKR-Medium.otf,%
151     Path=./fonts/Noto/Korean/,%
152     Script=CJK%
153   ]{NotoSansKR-Regular.otf}%
154 }

```

### 3.5 Support for Hebrew script

```

155 \ccDeclareBabelFont{hebrew}{[%
156     Scale=MatchUppercase,%
157     Path=./fonts/Noto/Hebrew/,%
158     Ligatures=TeX,%
159     BoldFont = NotoSerifHebrew-Bold.ttf]{NotoSerifHebrew-Regular.ttf}%
160 }{[%
161     Scale=MatchUppercase,%
162     Path=./fonts/Noto/Hebrew/,%
163     Ligatures=TeX,%
164     BoldFont = NotoSansHebrew-Bold.ttf]{NotoSansHebrew-Regular.ttf}%
165 }

```



### 3.6 Support for Arabic script

```

166 \ccDeclareBabelFont{arabic}{[%
167     BoldFont = NotoNaskhArabic-Bold.ttf,%
168     Path = ./fonts/Noto/Arabic/%
169     ]{NotoNaskhArabic-Regular.ttf}}
170 {[%
171     BoldFont = NotoSansArabic-Bold.ttf,%
172     Path = ./fonts/Noto/Arabic/%
173     ]{NotoSansArabic-Regular.ttf}%
174 }

```

### 3.7 Support for Greek script

```

175 \ccDeclareBabelFont{greek}{[%
176     BoldFont = NotoSerif-Bold.ttf,%
177     ItalicFont = NotoSerif-Italic.ttf,%
178     BoldItalicFont = NotoSerif-BoldItalic.ttf,%
179     Path = ./fonts/Noto/Serif/,%
180     WordSpace = 1.25
181     ]{NotoSerif-Regular.ttf}}
182 {[%BoldFont = NotoSans-Bold.ttf,%
183     ItalicFont = NotoSans-Italic.ttf,%
184     BoldItalicFont = NotoSans-BoldItalic.ttf,%
185     Path = ./fonts/Noto/Sans/,%
186     WordSpace = 1.25%
187     ]{NotoSans-Regular.ttf}%
188 }

```

### 3.8 Support for Ethiopian/Amharic script

```

189 \ccDeclareBabelFont{ethiop}{[%
190     BoldFont = NotoSerifEthiopic-Bold.ttf,%
191     ItalicFont = NotoSerifEthiopic-Regular.ttf,%
192     BoldItalicFont = NotoSerifEthiopic-Bold.ttf,%
193     Path = ./fonts/Noto/Ethiop/,%
194     WordSpace = 1.25
195     ]{NotoSerifEthiopic-Regular.ttf}}
196 {[%BoldFont = NotoSansEthiopic-Bold.ttf,%
197     ItalicFont = NotoSansEthiopic-Regular.ttf,%
198     BoldItalicFont = NotoSansEthiopic-Bold.ttf,%
199     Path = ./fonts/Noto/Ethiop/,%
200     WordSpace = 1.25%
201     ]{NotoSansEthiopic-Regular.ttf}%
202 }
203 \ccDeclareBabelFont{amharic}{[%
204     BoldFont = NotoSerifEthiopic-Bold.ttf,%
205     ItalicFont = NotoSerifEthiopic-Regular.ttf,%
206     BoldItalicFont = NotoSerifEthiopic-Bold.ttf,%
207     Path = ./fonts/Noto/Ethiop/,%
208     WordSpace = 1.25
209     ]{NotoSerifEthiopic-Regular.ttf}}
210 {[%BoldFont = NotoSansEthiopic-Bold.ttf,%
211     ItalicFont = NotoSansEthiopic-Regular.ttf,%
212     BoldItalicFont = NotoSansEthiopic-Bold.ttf,%
213     Path = ./fonts/Noto/Ethiop/,%

```

```

214     WordSpace = 1.25%
215   ]{NotoSansEthiopic-Regular.ttf}%
216 }

```

### 3.9 Support for Syrian script

Since Babel does not support the Syrian script natively, we create a `babel-syriac.ini` file and include it, if it is needed. If we don't, the kerning and ligatures of Syriac text will be off.

Please note that due to the restrictions of the `listings`-Package, some Unicode characters cannot be displayed correctly in the documentation of the following code. Therefore, Syriac letters appear as “x” in the following source code listing.

```

217 \expandafter\ifx\csname use@script@syriac\endcsname\@empty%
218 \RequirePackage{filecontents}
219 \begin{filecontents*}{babel-syriac.ini}
220 [identification]
221 charset = utf8
222 version = 0.1
223 date = 2019-08-25
224 name.local = ??????????
225 name.english = Classical Syriac
226 name.babel = classicalsyrac
227 tag.bcp47 = syc
228 tag.opentype = SYR
229 script.name = Syriac
230 script.tag.bcp47 = Syrc
231 script.tag.opentype = syrc
232 level = 1
233 encodings =
234 derivate = no
235 [captions]
236 [date.gregorian]
237 [date.islamic]
238 [time.gregorian]
239 [typography]
240 [characters]
241 [numbers]
242 [counters]
243 \end{filecontents*}
244 \fi

```

Now, we can create the fallback font and import the newly created ini file:

```

245 \ccDeclareBabelFont[import=syriac]{syriac}{[%
246   BoldFont = NotoSansSyriac-Black.ttf,%
247   ItalicFont = NotoSansSyriac-Regular.ttf,%
248   BoldItalicFont = NotoSansSyriac-Black.ttf,%
249   Path = ./fonts/Noto/Syriac/,%
250   WordSpace = 1.25
251   ]{NotoSansSyriac-Regular.ttf}}
252 {[BoldFont = NotoSansSyriac-Black.ttf,%
253   ItalicFont = NotoSansSyriac-Regular.ttf,%
254   BoldItalicFont = NotoSansSyriac-Black.ttf,%
255   Path = ./fonts/Noto/Syriac/,%
256   WordSpace = 1.25%
257   ]{NotoSansSyriac-Regular.ttf}%
258 }

```

```
</script>
```



## Module 9

# coco-title.dtx

```
<*title>
```

This file provides macros and facilities for title pages.

```

23 %%
24 %% module for CoCoTeX for maketitle.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-title}
32   [2024/07/16 0.5.0 CoCoTeX title module]
33 \RequirePackage{coco-meta}

```

## 1 Top-Level Interface

`titlepage` is the main Container for the whole document's meta data.

```

34 \ccDeclareContainer{titlepage}{%
35   \ccInherit {Components,Properties}{CommonMeta}%
36   \ifarticle\ccInherit{Components}{article-meta}\fi
37   \ccDeclareType{Components}{%
38     \cct@simple@comps
39     \cct@fundings@comp
40     \cct@role@handlers{author}{Author}%
41     \cct@declare@role{editor}{Editor}%
42     \cct@declare@role{series-editor}{SeriesEditor}%
43   }%
44   \ccDeclareType{Properties}{}%
45   \ccDeclareEnv[Meta]{\cctmeta}{\endcctmeta}%
46 }

```

`\cct@declare@role` declares the roles for editors and series editors and initializes the biography meta block for both.

```

47 \def\cct@declare@role#1#2{%
48   \ccDeclareRole[#1]{#2}%
49   \cct@role@handlers{#1}{#2}%
50 }

```

`\cct@role@handlers` adds title page specific Components and Handlers to the Author, Editor and Series-Editor Roles.

```

51 \def\cct@role@handlers#1#2{%
52   \ccAddToRole{#2}{%
53     \ccDeclareCountedComponent{Bio}%
54     \ccDeclareCountedComponent{Biography}}%
55   \ccDeclareGroupHandler{#2}{%
56     \ccIfComp{Biography}{}\ccIfComp{Bio}{\ccComponent{Biography}{\ccUseProperty{#1-biography-
57       format}}}\}}%
58   \ccDeclareRoleBlock[apply]{#2}{BioBlock}{#1-bio-block-format}%
59 }

```

`\ccDeclareTitlepage` is the default titlepage declarator with the next token being added the titlepage's Property list.

```

60 \def\ccDeclareTitlepage{\ccAddToType{Properties}{titlepage}}

```

`\cctmeta` is the code executed at the beginning of the `\ccPrefix Meta` Container

```

61 \def\cctmeta{\ccOpt@empty\cct@meta}
62 \def\cct@meta[#1]{%
63   \UseHook{env/meta/begin}%
64   \ccEvalAttributes[titlepage]{#1}%
65   \ccEvalType{Components}%
66   \let\and\relax
67 }

```

`\ccAddTitleRole` is a user-level macro to add both a new Role with the name `{#2}` and a controlling Property `{#1}` to the `titlepage` container.

```

68 \def\ccAddTitleRole#1#2{%
69   \ccAddToType{Components}{titlepage}{\cct@declare@role{#1}{#2}}%
70   \ccAddTitleEval{\cct@eds@eval{#2}}%
71 }

```

`\ccAddTitleEval` is a User-level macro to add additional Material titlepage evaluators (the next token).

```

72 \def\ccAddTitleEval{\csgappto{\cct@add@eval}}

```

`\cct@add@eval` is a hook for additional titlepage evaluators

```

73 \def\cct@add@eval{}

```

`\endcctmeta` is the code executed at the end of the `Meta` Container

```

74 \def\endcctmeta{%
75   \ccSetContainer{titlepage}%
76   \ccEvalType{Properties}%
77   \cct@maketitle
78   \ccm@role@eval{Author}%
79   \ccApplyCollection{Affil}{affil-block-item-format}{AffilBlock}%
80   \cct@eds@eval{Editor}%
81   \cct@eds@eval{SeriesEditor}%
82   \ccm@generic@eval
83   \cct@fundings@eval
84   \cct@add@eval
85   \cc@if@preamble\cct@set@pdfmeta\relax

```

Now, we expand the `\cct/document/meta`.

```
86 \UseHook{cct/document/meta}%
87 \let\cct@cur@cont\@empty
88 }
```

## 2 Processing of PDF Meta Data

The next few macros handle the content that is written directly into the pdf as meta data.

`\cct@set@pdfmeta` is the wrapper for the whole meta data handling.

```
89 \def\cct@set@pdfmeta{%
```

`\cct@write@pdf@meta@string` handles meta data that are stored as plain *strings* in the XMP file.

`{#1}` is the hyperref name for the meta datum  
`{#2}` is the ltpdfa name for the meta datum  
`{#3}` is the fallback value.

The fallback value is only chosen if either no XMP file exists, or if the XMP file does not contain the required data field. In this case, the string is sanitized through hyperref's `\pdfstringdef`, which may result in weird encoding errors, so be aware.

```
90 \def\cct@write@pdf@meta@string##1##2##3{%
91   \let\cct@cur@data\@empty
92   \ccIfAlly
93     {\pdfstringdef\cct@cur@data{\directlua{tex.print(cocotex.ally.meta.##2)}}}%
94     \ifx\cct@cur@data\@empty
95       \pdfstringdef\cct@cur@data{##3}%
96     \fi
97     \ifx\cct@cur@data\@empty
98       \ccDebugMsg[pdfmeta]{##2 is empty (3: ##3); doing nothing}%
99     \else
100       \ccDebugMsg[pdfmeta]{Writing \string\setDocinfo[utf-16]{##2}{\expandafter\strip@prefix\
101         meaning\cct@cur@data}}%
102       \edef\x{\noexpand\ccaSetDocinfo[utf-16]{##2}}\expandafter\x\expandafter{\cct@cur@data}%
103       \fi
104     {\cct@write@hr@infodict{##1}{##3}}}%
105 }
```

`\cct@write@pdf@meta@list` handles meta data that are represented as *lists* in the XMP file.

`{#1}` is the hyperref name for the meta datum  
`{#2}` is the ltpdfa name for the meta datum  
`{#3}` is the fallback value.  
`{#4}` is the handler for the list. This is a macro that takes one argument. Each item of the list will be passed to that macro as argument.  
`{#5}` is the separator between the items

```
104 \def\cct@write@pdf@meta@list##1##2##3##4##5{%
105   \let\cct@cur@data\@empty
106   \ccIfAlly
107     {\protected@edef\cct@cur@data{\directlua{tex.print(cocotex.ally.meta.##2)}}}%
108     \ifx\cct@cur@data\@empty
109       \protected@edef\cct@cur@data{##3}%
110     \ifx\cct@cur@data\@empty\else
```

```

111 \edef\x{\noexpand\cct@split@pdf@meta{\noexpand##4}{##5}}\x%
112 \fi
113 \else
114 \cct@split@pdf@meta{##4}{\and}%
115 \fi}
116 {\cct@write@hr@infodict{##1}{##3}}}%

```

`\cct@split@pdf@meta` is a helper function to split the list-form meta datum stored `\cct@cur@data` in into its items.

`{#1}` is a CS token that takes one argument. Each item of the list is recursively passed to that CS token

`{#2}` is the separator between the items

```

117 \def\cct@split@pdf@meta##1##2{%
118 \def\cct@split@pdf@meta####1##2####2\@nil####3{%
119 \if\relax\detokenize{####1}\relax\else
120 ####3{####1}%
121 \if!####2!\else
122 \edef\@argi{##2}%
123 \edef\@argii{####2}%
124 \ifx\@argi\@argii\else
125 \cct@split@pdf@meta####2\@nil{####3}%
126 \fi
127 \fi
128 \fi
129 }%
130 \expandafter\cct@split@pdf@meta\cct@cur@data##2\@nil{##1}%
131 }%

```

`\cct@write@hr@infodict` writes the PDF info dictionary entry via `hyperref` with the key `{#1}` and the value `{#2}`.

```

132 \def\cct@write@hr@infodict##1##2{%
133 \protected@edef\x{\noexpand\hypersetup{##1={\expandonce{##2}}}}\x
134 }%

```

After we decided how we want to process the PDF meta data, we now start to collect the necessary data points:

```

135 \cct@title@insert@xmp
136 \cct@title@process@bkbc
137 \cct@title@process@bkt
138 \cct@title@process@bkk
139 \cct@title@process@bka
140 }

```

## 2.1 Processing of the Document's Title

`\cct@title@process@bkt` processes the document's main title

```

141 \def\cct@title@process@bkt{%
142 \cslet{\ccPrefix Break}\space
143 \protected@xdef\@title{\ccUseProperty{doc-book-title}}%
144 \cct@write@pdf@meta@string{pdftitle}{Title}{\@title}%
145 \ccpgdefFromProperty{RunBookTitle}{run-book-title}%
146 }

```



## 2.2 Processing of the Document's Author

`\cct@title@process@bka` processes the document's main author or, if that doesn't exist, the main editor, or throws a warning if neither exist.

```

147 \def\cct@title@process@bka{%
148   \@tempswatrue
149   \begingroup
150   \ccGobble
151   \renewcommand\foreignlanguage[2]{\##2}%
152   \ccIfComp{AuthorPDFInfo}
153     {\ccpgdefFromComp{RunBookName}{AuthorPDFInfo}}
154     {\ccIfComp{EditorPDFInfo}
155       {\ccpgdefFromComp{RunBookName}{EditorPDFInfo}}
156       {\ccIfComp{AuthorNameList}
157         {\ccpgdefFromComp{RunBookName}{AuthorNameList}}
158         {\ccIfComp{EditorNameList}
159           {\ccpgdefFromComp{RunBookName}{EditorNameList}}
160           {\ifnum\ccAuthorCnt>\z@
161             \@setpar{\@par}%
162             \ccpxdefFromCountedComp{RunBookName}{Author}{author-list-pdfinfo-format}%
163             \else
164             \ifnum\ccEditorCnt>\z@
165               \ccpxdefFromCountedComp{RunBookName}{Editor}{editor-list-pdfinfo-format}%
166               \else
167               \ccPackageWarning{transcript-title}{Meta Data}{No author or editor given!}%
168               \@tempswafalse
169             \fi
170           \fi
171         }}}}%
172   \if@tempswa
173     \pdfstringdef\@author{\csname\ccPrefix RunBookName\endcsname}%
174     \cct@write@pdf@meta@list{pdfauthor}{Author}{\csname\ccPrefix RunBookName\endcsname}{\
175       ccaAddAuthor}{\and}%
176   \fi
177 \endgroup
}

```

## 2.3 Processing of the PDF's Creator, Producer, and Keywords Meta Data

`\cct@title@process@bkc` processes the metadata for the pdf creator and producer.

```

178 \def\cct@title@process@bkc{%
179   \cct@write@pdf@meta@string{pdfcreator}{Creator}{\ccWhenComp{PDFCreator}{\ccUseComp{PDFCreator
180     }}}}%
181   \cct@write@pdf@meta@string{pdfproducer}{Producer}{\ccWhenComp{PDFProducer}{\ccUseComp{
182     PDFProducer}}}%
183 }

```

`\cct@title@process@bkk` processed the metadata for the keywords.

```

182 \def\cct@title@process@bkk{%
183   \cct@write@pdf@meta@list{pdfkeywords}{Keywords}{\ccWhenComp{Keywords}{\ccUseComp{Keywords}}}{\
184     ccaAddKeyword}{\ccUseProperty{keywords-sep}}}%
185 }

```

## 2.4 Including the XMP Meta Data

`\cct@title@insert@xmp` inserts the contents of the XMP meta data file into the pdf, if it exists. There are two versions, depending on whether coco-accessibility is active or not.

```
185 \def\cct@title@insert@xmp{\ccIfAlly{\cct@title@insert@xmp@ltpdfa}{\cct@title@insert@xmp@direct}}
```

`\cct@title@insert@xmp@direct` is the default version which writes the xmp meta data directly into the PDF.

```
186 \def\cct@title@insert@xmp@direct{%
187   \edef\include@xmp{\noexpand\@include@xmp{\ccUseComp{XmpFile}.xmp}}%
188   \def\@include@xmp##1{\IfFileExists{##1}{\@include@xmp{##1}}{}}%
189   \def\@include@xmp##1{%
190     \begingroup
191     \immediate\pdfobj stream attr {/Type /Metadata /Subtype /XML}
192     file{##1}
193     \pdfcatalog{/Metadata \the\pdflastobj\space 0 R}
194     \endgroup}%
195   \include@xmp
196 }
```

`\cct@title@insert@xmp@ltpdfa` is the version that uses ltpdfa's mechanism to write XMP meta data into the PDF.

First we check if the specified xmp file exists. If it exists, the `DocumentInfo` is extracted from the XMP file. Otherwise, we set the `DocumentInfo` from the contents of the `titlepage` Container and let `ltpdfa` generate the `xmp` file.

```
197 \def\cct@title@insert@xmp@ltpdfa{%
198   \edef\cca@xmp@file@name{\ccUseComponentFrom{titlepage}{XmpFile}.xmp}%
199   \IfFileExists{\cca@xmp@file@name}
200     {\ccaAddToConfig{metadata}{xmpfile=\cca@xmp@file@name}%
201      \directlua{ally.meta:extract()}}
202     {\ccPackageWarning{A11y}{File}{%
203 \cca@xmp@file@name\space not found.^^J
204 Note that the ltpdfa package will create one^^J
205 from the Components given in the Meta Container.}}}

```

## 3 Intermediate Level Interfaces

`cct/maketitle/before` is expanded right before the titlepage is printed.

```
206 \NewHook{cct/maketitle/before}
```

`cct/maketitle/after` is expanded at the end of the titlepage.

```
207 \NewHook{cct/maketitle/after}
```

`cct/document/meta` is expanded at the very end of the Meta Container.

```
208 \NewHook{cct/document/meta}
```

`env/meta/begin` is used to add code to be executed at the very beginning of the Meta Container's main environment, before the Attributes are evaluated.

```
209 \NewHook{env/meta/begin}
```

`\cct@article@titlepage` is the prototype for article title pages.

```
210 \def\cct@article@titlepage{%
211   \ccUseProperty{article-title}%
212 }
```

`\cct@journal@titlepage` is the prototype for journal title pages.

```
213 \def\cct@journal@titlepage{%
214   \ccUseProperty{before-titlepage}%
215   \ccUseProperty{coverpage}%Cover ist kein Bild, wird von uns gebaut
216   \ccUseProperty{before-titlepage-roman}%
217   \ccUseProperty{titlepage-roman}%
218   \ccUseProperty{after-titlepage}%
219 }
```

`\cct@book@titlepage` is the prototype for book (monographs and collections) title pages.

```
220 \def\cct@book@titlepage{%
221   \ccUseProperty{before-titlepage}%
222   \ccWhenComp{Cover}{\ccUseProperty{coverpage}}%
223   \ccUseProperty{before-titlepage-roman}%
224   \ccUseProperty{titlepage-roman}%
225   \ccUseProperty{after-titlepage}%
226 }
```

`\cct@maketitle` assigns one of the above definitions to the `\ccPrefix Maketitle` macro.

```
227 \def\cct@maketitle{%
228   \expandafter\gdef\csname\ccPrefix Maketitle\endcsname{%
229     \let\cc@cnt@grp\@empty
```

Here, we expand the `\cct/maketitle/before`.

```
230   \UseHook{cct/maketitle/before}%
231   \bgroup
232   \ccSetContainer{titlepage}%
233   \ccEvalType{Properties}%
234   \ifarticle
235     \cct@article@titlepage
236   \else
237     \ifjournal
238       \cct@journal@titlepage
239     \else
240       \cct@book@titlepage
241     \fi
242   \fi
243 \egroup
244   \UseHook{cct/maketitle/after}%
245 }%
246 }
```

### 3.1 Funds, Grants, and Supporters

This is a Subcontainer within `\ccPrefix Meta` which allows to set up multiple funding, grant, or supporter callouts.

`\cct@fundings@comp` wrapper to set up the Subcontainer

```

247 \def\cct@fundings@comp{%
248   \ccDeclareComponent{FundingBlock}{\expandafter\global}{}%
249   \ccDeclareComponentGroup{Funding}{%
250     \ccDeclareCountedComponent{FundName}%
251     \ccDeclareCountedComponent{FundLogo}%
252     \ccDeclareCountedComponent{FundID}%
253   }{}%
254 }
```

`\cct@fundings@eval` Evaluator for the funding

```

255 \def\cct@fundings@eval{%
256   \def\cc@cur@cont{titlepage}%
257   \ccComposeCollection{Funding}{fund-format}{FundingBlock}%
258 }
```

`\cct@eds@eval` evaluator for the editors

```

259 \def\cct@eds@eval#1{%
260   \ccm@role@eval{#1}%
261   \cct@create@editor@string{#1}}
```

`\cct@create@editor@string` evaluates the editor string and adds a suffix.

```

262 \def\cct@create@editor@string#1{%
263   \expandafter\ifx\csname cc@\cc@cur@cont @#1NameList\endcsname\relax\else
264     \csgappto{cc@\cc@cur@cont @#1NameList}{\letcs\ccTotalCount{cc#1Cnt}\ccUseProperty{editor-
       suffix}}}%
265   \fi
266 }
```

## 3.2 Simple Component Declarations

`\cct@simple@comps` wrapper for the Titlepage's simple Components.

```

267 \def\cct@simple@comps{%
```

### General Information

`Cover` holds the path(!) to the cover image (without `\includegraphics!`)

```

268   \ccDeclareGlobalComponent{Cover}%
```

`Dedication` is a dedication.

```

269   \ccDeclareGlobalComponent{Dedication}%
```

`Acknowledgements` self explanatory.

```

270   \ccDeclareGlobalComponent{Acknowledgements}%
```

**Statement** additional publication statement

```
271 \ccDeclareGlobalComponent{Statement}%
```

**Editorial** generic statement by the editors of a periodical or collection.

```
272 \ccDeclareGlobalComponent{Editorial}%
```

## Titles and Names

**Title** is the document's (printed) main title.

```
273 \ccDeclareGlobalComponent{Title}%
```

**ShortTitleOR** is a shortened version of the document title. If set, this is the title that is written into the PDF meta data (unless DocTitle is also set) and stored as running title (unless RunTitle is also set). If the Component is not used, the Title component is used, instead.

```
274 \ccDeclareGlobalComponent{ShortTitle}%
```

**DocTitleOR** is an override for the title that is written into the PDF meta data (unless an XMP file is used). This Component's value should only contain alphanumeric characters, ideally from the ASCII code block, but must not contain any L<sup>A</sup>T<sub>E</sub>X markup.

```
275 \ccDeclareGlobalComponent{DocTitle}%
```

**RunTitleOR** is an override for the title that is used as running title for page headers. It should contain only robust L<sup>A</sup>T<sub>E</sub>X markup.

```
276 \ccDeclareGlobalComponent{RunTitle}%
```

**AltTitle** is an alternative title for bastard title pages, etc. It is not used by CoCoT<sub>E</sub>X, but some publisher styles may need it.

```
277 \ccDeclareGlobalComponent{AltTitle}%
```

**Subtitle** is the document's subtitle.

```
278 \ccDeclareGlobalComponent{Subtitle}%
```

**TitleNote** additional printed information tied to the document's title.

```
279 \ccDeclareGlobalComponent{TitleNote}%
```

**RunNamesOR** is an override for the document author's names, which is intended to be used in running page headers. If not set, the (calculated) ➤ **AuthorNameList** or ➤ **EditorNameList** is used, instead.

```
280 \ccDeclareGlobalComponent{RunNames}%
```

**AltNames** is an alternative data field for additional names.

```
281 \ccDeclareGlobalComponent{AltNames}%
```

## Series

**Series** is the series title.

```
282 \ccDeclareGlobalComponent{Series}%
```

**SubSeries** is the subtitle for the series.

```
283 \ccDeclareGlobalComponent{SubSeries}%
```

**SeriesNote** are additional notes concerning the series.

```
284 \ccDeclareGlobalComponent{SeriesNote}%
```

**Volume** is the volume of the document within a series.

```
285 \ccDeclareGlobalComponent{Volume}%
```

**Number** is the number of the document within the series.

```
286 \ccDeclareGlobalComponent{Number}%
```

**EditorNameListCC** is the Collection Component for the Editor's names.

```
287 \ccDeclareGlobalComponent{EditorNameList}%
```

**SeriesEditorNameListCC** is the Collection Component for the Series-Editor's names.

```
288 \ccDeclareGlobalComponent{SeriesEditorNameList}%
```

## Publisher Information

**Publisher** is the publisher name

```
289 \ccDeclareGlobalComponent{Publisher}%
```

**PubDivision** is the publisher division.

```
290 \ccDeclareGlobalComponent{PubDivision}%
```

**PubDivInfo** holds additional information about the publisher division.

```
291 \ccDeclareGlobalComponent{PubDivInfo}%
```

**PubPlace** holds the place of publication or the publisher's address.

```
292 \ccDeclareGlobalComponent{PubPlace}%
```

**PubLogo** holds the publisher logo. Depending on the publisher style, this may be just a path, or a complete `\includegraphics` expression.

```
293 \ccDeclareGlobalComponent{PubLogo}%
```

**PubNote** additional generic notes about the publisher.

```
294 \ccDeclareGlobalComponent{PubNote}%
```

**PubWeb** holds the url or email contact address of a publisher.

```
295 \ccDeclareGlobalComponent{PubWeb}%
```

### Publication Meta

**XmpFile** is the basename of the XMP meta data file without the .xmp file ending (which is added automatically). The default value is `\jobname`.

```
296 \ccDeclareGlobalComponent[\jobname]{XmpFile}%
```

**PDFCreator** is the tool with which the original document was created (for *xerif*, this is usually M\$ Word).

```
297 \ccDeclareGlobalComponent{PDFCreator}%
```

**PDFProducer** is the tool with which the PDF was created. Defaults to “le-tex xerif with CoCoTeX v(CoCoTeX version)”

```
298 \ccDeclareGlobalComponent[le-tex xerif with CoCoTeX v.0.5.0]{PDFProducer}%
```

**Year** the year of the publication

```
299 \ccDeclareGlobalComponent{Year}%
```

**Edition** the edition of the publication.

```
300 \ccDeclareGlobalComponent{Edition}%
```

**EditionNote** additional notes about the particular edition.

```
301 \ccDeclareGlobalComponent{EditionNote}%
```

**ISBNPreText** text added before the ISBN block

```
302 \ccDeclareGlobalComponent{ISBNPreText}%
```

**ISBN** the publication’s international standard book number.

```
303 \ccDeclareGlobalComponent{ISBN}%
```

**ISSN** the publication’s international standard serial number for periodicals.

```
304 \ccDeclareGlobalComponent{ISSN}%
```

**EISSN** additional ISSN for electronic publications

```
305 \ccDeclareGlobalComponent{EISSN}%
```

**EpubPreText** additional text between ISBN and eISBN

```
306 \ccDeclareGlobalComponent{EpubPreText}%
```

**EISBN** ISBN for electronic publications.

```
307 \ccDeclareGlobalComponent{EISBN}%
```

**EpubISBN** ISBN for EPUBs.

```
308 \ccDeclareGlobalComponent{EpubISBN}%
```

**ElibPDF** additional serial number for electronic libraries.

```
309 \ccDeclareGlobalComponent{ElibPDF}%
```

**BiblISSN** additional ISSN for special libraries.

```
310 \ccDeclareGlobalComponent{BiblISSN}%
```

**BibleISSN** additional electronic ISSN for special libraries.

```
311 \ccDeclareGlobalComponent{BibleISSN}%
```

## Funding

**FundingPreText** additional text before the funding list.

```
312 \ccDeclareGlobalComponent{FundingPreText}%
```

**FundingPostText** additional text after the funding list.

```
313 \ccDeclareGlobalComponent{FundingPostText}%
```

## Imprint Meta Data

**Biblio** bibliographic information block.

```
314 \ccDeclareGlobalComponent{Biblio}%
```

**BiblioTitle** the title of the bibliographic information block

```
315 \ccDeclareGlobalComponent{BiblioTitle}%
```

**Print** the name and address of the printing company.

```
316 \ccDeclareGlobalComponent{Print}%
```

**PrintNote** additional information about the printing process.

```
317 \ccDeclareGlobalComponent{PrintNote}%
```



**Lectorate** name and address of the lectorate

```
318 \ccDeclareGlobalComponent{Lectorate}%
```

**Translator** name of the document's translator

```
319 \ccDeclareGlobalComponent{Translator}%
```

**CoverConcept** concept creator of the front page cover

```
320 \ccDeclareGlobalComponent{CoverConcept}%
```

**CoverDesign** designer of the front page cover.

```
321 \ccDeclareGlobalComponent{CoverDesign}%
```

**Component** creator of the front page cover image

```
322 \ccDeclareGlobalComponent{CoverImage}%
```

**Typesetter** name and address of the typesetter

```
323 \ccDeclareGlobalComponent{Typesetter}%
```

**QA** name of the person(s) responsible for quality assurance.

```
324 \ccDeclareGlobalComponent{QA}%
```

**UsedFont** information about the fonts used throughout the document

```
325 \ccDeclareGlobalComponent{UsedFont}%
```

**Conversion** name of the person(s) responsible for data conversion

```
326 \ccDeclareGlobalComponent{Conversion}%
```

**EnvDisclaimer** environmental disclaimer, used paper, etc.

```
327 \ccDeclareGlobalComponent{EnvDisclaimer}%
```

**Advertise** advertisements.

```
328 \ccDeclareGlobalComponent{Advertise}%
```

## Licencing

**LicenceText** License Description

```
329 \ccDeclareGlobalComponent{LicenceText}%
```

**LicenseLogo** the path(!) to the licence logo. `\includegraphics` is added automatically.

```
330 \ccDeclareGlobalComponent{LicenseLogo}%
```

**LicenceLink** URL to the license.

```
331 \ccDeclareGlobalComponent{LicenceLink}%
```

**LicenceName** the plain name of the license.

```
332 \ccDeclareGlobalComponent{LicenceName}%
```

**CopyrightDisclaimer** self explanatory...

```
333 \ccDeclareGlobalComponent{CopyrightDisclaimer}%
```

### Journal-specific Meta Data

**JournalName** Full name of the journal.

```
334 \ccDeclareGlobalComponent{JournalName}%
```

**JournalAbbrev** short name of the journal.

```
335 \ccDeclareGlobalComponent{JournalAbbrev}%
```

**Issue** of the journal.

```
336 \ccDeclareGlobalComponent{Issue}%
```

**PubCycle** Publication cycle

```
337 \ccDeclareGlobalComponent{PubCycle}%
```

**Prices** of the journal issues or subscription models

```
338 \ccDeclareGlobalComponent{Prices}%
```

**MemberList** in case of publishing organizations, this Component may hold a list of members.

```
339 \ccDeclareGlobalComponent{MemberList}%
```

**Startpage** is the start page of the Journal

```
340 \ccDeclareGlobalComponent{Startpage}%
```

### Generic additional information

**AddNoteI** additional information for the first title page.

```
341 \ccDeclareGlobalComponent{AddNoteI}%
```

**AddNoteII** additional information for the second title page.

```
342 \ccDeclareGlobalComponent{AddNoteII}%
```

`AddNoteIII` additional information for the third title page.

```
343 \ccDeclareGlobalComponent{AddNoteIII}%
```

`AddNoteIV` additional information for the fourth title page.

```
344 \ccDeclareGlobalComponent{AddNoteIV}%
```

```
345 }
```

## 4 Default Settings

```
346 \ccAddToProperties{titlepage}{%
347   \ccSetProperty{article-title}{}%
348   % Title page hooks
349   % Before \ccPrefix Maketitle and outside the group
350   \ccSetProperty{before-titlepage}{%
351     \pagestyle{empty}%
352     \parindent\z@
353     \parskip\z@
354   }%
355   \ccSetProperty{after-titlepage}{\pagestyle{headings}}%
356   % Pages of title
357   %% Cover page
358   \ccSetProperty{coverpage}{%
359     \bgroup
360     \def\thepage{\@alph\c@page}%
361     \smash{\rlap{%
362       \raise\dimexpr\headheight+\headsep+\topmargin+\topskip-\paperheight\relax
363       \vtop{%
364         \hskip-\oddsidemargin
365         \includegraphics[width=\paperwidth,height=\paperheight]{\ccUseComp{Cover}}%
366       }}}%
367     \ccUseProperty{after-coverpage}%
368     \egroup
369   }%
370   \ccSetProperty{after-coverpage}{\cleardoublepage}%
371   \ccSetProperty{titlepage-roman}{%
372     \ccUsePropertyEnv{titlepage-i}%
373     \clearpage
374     \ccUsePropertyEnv{titlepage-ii}%
375     \clearpage
376     \ccUsePropertyEnv{titlepage-iii}%
377     \clearpage
378     \ccUsePropertyEnv{titlepage-iv}%
379     \clearpage
380   }%
381   %% Generic meta blocks
382   \ccSetProperty{generic-meta-heading-face}{\large}% format of the heading of a generic meta block
383   \ccSetProperty{generic-meta-format}{% Format of a single generic meta-block
384     \ccIfComp{Heading}{\ccUseProperty{generic-meta-heading-face}\ccUseComp{Heading}\par}\vskip\baselineskip}%
385     \ccUseComp{Content}%
386     \par%
387   }%
388   %% Funding
```

```

389 \ccSetProperty{funding-columns}{2}
390 \ccSetProperty{funding-format}{}%
```

Fallback for the width in case someone sets up a fixed value for a fund's width.

```

391 \ccSetProperty{fund-width}{.5\textwidth}
392 \ccSetProperty{fund-vertical-sep}{\baselineskip}%
393 \ccSetProperty{fund-sep}{%
394   \expandafter\@tempcnta\CalcModulo{\ccCurCount}{\ccUseProperty{funding-columns}}%
395   \ifnum\@tempcnta=\z@
396     \par
397     \ifnum\ccCurCount<\ccTotalCount\relax
398       \vskip\ccUseProperty{fund-vertical-sep}%
399     \fi
400   \else
401     \hfill
402   \fi}
403 \ccSetProperty{fund-format}{% Format of a single fund/grant/sponsor
404   \strut\vtop{%
405     \hsize\ccUseProperty{fund-width}%
406     \ccIfComp{FundName}{\ccUseComp{FundName}}\{[1ex]\}%
407     \includegraphics[width=\ccUseProperty{fund-width}]{\ccUseComp{FundLogo}}}%
408   \ccUseProperty{fund-sep}%
409 }%
410 \ccSetProperty{funding-sep}{4mm}%
411 \ccSetProperty{funding-block}{%
412   \bgroup
```

We set `fund-width` here so that the value is calculated only once and only the result is stored in the `fund-width` Property.



```

413   \ccSetPropertyX{fund-width}{\dimexpr(\textwidth/\ccUseProperty{funding-columns})-(\
414     ccUseProperty{funding-sep}/\ccUseProperty{funding-columns})\relax}
415   \ccUseProperty{funding-format}%
416   \ccGetComp{FundingPreText}%
417   \ccGetComp{FundingBlock}%
418   \ccGetComp{FundingPostText}%
419   \par
420   \egroup
421 }
422 %% before the roman part of the title pages but after cover page
423 \ccSetProperty{before-titlepage-roman}{%
424   \setcounter{page}{1}%
425   \def\thepage{\roman{page}}%
426 }%
427 \ccSetProperty{titlepage-i}{%
428   \ifmonograph
429     \ccUseComp{AuthorNameList}%
430   \else
431     \ccUseProperty{EditorNameList}%
432   \fi%
433   \vskip\baselineskip
434   \bgroup
435   \ccUseProperty{title-face}\ccUseComp{Title}%
436   \egroup
437 }%
438 \ccSetProperty{titlepage-ii}{%
439   \ccGetComp{Editorial}%
440   \ccGetComp{SeriesNote}%
441   \ccGetComp{GenericMetaBlock}%
442   \vfill
```

```

442 \ccUseProperty{bio-output}%
443 }%
444 \ccSetProperty{titlepage-iii}{%
445 \ifmonograph
446 \ccUseComp{AuthorNameList}%
447 \else
448 \ccUseProperty{EditorNameList}%
449 \fi%
450 \par
451 \ccUseProperty{title-format}
452 \ccGetComp{Edition}%
453 \ccGetComp{EditionNote}%
454 \vfill
455 \clearpage
456 }%
457 \ccSetProperty{titlepage-iv}{%
458 \ccGetComp{Dedication}% Dedication
459 \ccGetComp{Acknowledgements}% Dedication
460 \ccUseProperty{imprint-format}%
461 \ccUseProperty{funding-block}%
462 \vfill
463 \bgroup
464 \ccUseProperty{imprint-face}%
465 \ccIfComp{Biblio}{\bfseries\ccGetComp{BiblioTitle}}\ccGetComp{Biblio}}}%
466 \ccUseProperty{imprint-sep}%
467 \ccUseProperty{imprint}%
468 \egroup
469 \clearpage
470 }%
471 %% predefined face and format Properties
472 \ccSetProperty{title-face}{\Huge\sffamily\bfseries}%

```

The document's main title is tagged with the  `<Title/>` tag, which in PDF-Versions less than 2.0 should be mapped to  `<H1/>`.

```

473 \ccSetProperty{title-format}{%
474 \bgroup
475 \ccVstructStart{Title}% PDF 2.0
476 \ccUseProperty{title-face}%
477 \ccUseComp{Title}\par
478 \ccVstructEnd{Title}% PDF 2.0
479 \egroup
480 \ccWhenComp{Subtitle}{\ccUseProperty{subtitle-format}}%
481 \ccWhenComp{TitleNote}{\ccUseProperty{title-note-format}}%
482 \ccGetComp{Statement}%
483 \vskip\baselineskip
484 }%
485 \ccSetProperty{title-note-face}{\large\sffamily}%
486 \ccSetProperty{title-note-format}{%
487 \bgroup
488 \ccUseProperty{title-note-face}%
489 \ccUseComp{TitleNote}%
490 \egroup
491 \par
492 }%
493 \ccSetProperty{subtitle-face}{\Large\sffamily\bfseries}%
494 \ccSetProperty{subtitle-format}{%
495 \bgroup
496 \ccUseProperty{subtitle-face}%
497 \ccUseComp{Subtitle}%
498 \egroup

```

```

499 \par
500 }%
501 %% Imprint
502 \ccSetProperty{imprint-face}{\footnotesize}%
503 \ccSetProperty{imprint-sep}{\ifhmode\par\fi\addvspace{\baselineskip}}%
504 \ccSetProperty{imprint}{%
505   \ccUseProperty{publisher}%
506   \ccGetComp{Qualification}%
507   \ccGetComp{Conversion}%
508   \ccGetComp{CoverDesign}%
509   \ccGetComp{CoverImage}%
510   \ccGetComp{Lectorate}%
511   \ccGetComp{QA}%
512   \ccGetComp{Translator}%
513   \ccGetComp{Appraiser}%
514   \ccGetComp{Discussion}%
515   \ccGetComp{Typesetter}%
516   \ccGetComp{Print}%
517   \ccGetComp{UsedFont}%
518   \ccGetComp{DOI}%
519   \ccGetComp{Keywords}%
520   \ccUseProperty{imprint-sep}%
521   \ccGetComp{ISBNPreText}%
522   \ccGetComp{ISBN}%
523   \ccGetComp{EpubPreText}%
524   \ccGetComp{EISBN}%
525   \ccGetComp{EpubISBN}%
526   \ccUseProperty{imprint-sep}%
527   \ccGetComp{EnvDisclaimer}%
528 }%
529 \ccSetProperty{journal-meta}{%
530   \ccUseLabeledComp{Submitted}%
531   \ccUseLabeledComp{Received}%
532   \ccUseLabeledComp{Revised}%
533   \ccUseLabeledComp{Accepted}%
534   \ccUseLabeledComp{Published}%
535   \ccUseLabeledComp{Copyright}%
536   \ccUseLabeledComp{COIStatement}%
537   \ccUseLabeledComp{Keywords}%
538 }%
539 \ccSetProperty{licence}{%
540   \ccIfComp{LicenceLogo}{\includegraphics{\ccUseComp{LicenceLogo}}\par}{}%
541   \ccGetComp{LicenceText}%
542 }%
543 \ccSetProperty{copyright}{%
544   \ccIfComp{Copyright}
545     {\ccUseComp{Copyright}\par}
546     {\textcopyright\space\ccUseComp{Year}\space\ccUseComp{Publisher},\space\ccUseComp{PubPlace}
547       }\par}%
548 }%
549 \ccSetProperty{publisher}{%
550   \ccGetComp{PubDivInfo}%
551   \ccUseProperty{copyright}%
552   \ccGetComp{PubNote}%
553   \ccGetComp{PubWeb}%
554 }%
555 %% Name Formats
556 \ccSetProperty{counted-meta-sep}{\ifnum\ccCurCount<\ccTotalCount\relax\vskip\baselineskip\fi}%
557   separator between multiple instances of the same meta datum
558 \ccSetProperty{counted-name-sep}{% Separator between multiple names; titlepage-specific override of
559   the same Property in coco-meta!

```

```

557 \ifnum\ccTotalCount>1\relax
558 \ifnum\ccCurCount<\ccTotalCount\relax
559 \ifnum\ccCurCount<\numexpr\ccTotalCount-1\relax
560 \ccUseProperty{name-sep}%
561 \else
562 \ccUseProperty{name-and}%
563 \fi
564 \fi
565 \fi
566 }%
567 % Aliases for different Roles, see coco-meta.sty for the actual Property values:
568 %% editors:
569 \ccPropertyLet{editor-cite-name-format}{role-cite-name-format}%
570 \ccPropertyLet{editor-short-cite-name-format}{role-short-cite-name-format}%
571 \ccPropertyLet{editor-full-name-format}{role-full-name-format}%
572 \ccPropertyLet{editor-pdfinfo-name-format}{role-pdfinfo-name-format}%
573 \ccPropertyLet{editor-correspondence-as-format}{role-correspondence-string-format}%
574 %
575 \ccPropertyLet{editor-list-print-format}{role-block-print-format}%
576 \ccPropertyLet{editor-list-cite-format}{role-block-cite-format}%
577 \ccPropertyLet{editor-list-short-cite-format}{role-block-short-cite-format}%
578 \ccPropertyLet{editor-list-pdfinfo-format}{role-block-pdfinfo-format}%
579 \ccPropertyLet{editor-list-correspondence-format}{role-block-correspondence-format}%
580 %% series-editors:
581 \ccPropertyLet{series-editor-cite-name-format}{role-cite-name-format}%
582 \ccPropertyLet{series-editor-short-cite-name-format}{role-short-cite-name-format}%
583 \ccPropertyLet{series-editor-full-name-format}{role-full-name-format}%
584 \ccPropertyLet{series-editor-pdfinfo-name-format}{role-pdfinfo-name-format}%
585 \ccPropertyLet{series-editor-correspondence-as-format}{role-correspondence-as-format}%
586 %
587 \ccPropertyLet{series-editor-list-print-format}{role-block-print-format}%
588 \ccPropertyLet{series-editor-list-cite-format}{role-block-cite-format}%
589 \ccPropertyLet{series-editor-list-short-cite-format}{role-block-short-cite-format}%
590 \ccPropertyLet{series-editor-list-pdfinfo-format}{role-block-pdfinfo-format}%
591 \ccPropertyLet{series-editor-list-correspondence-format}{role-block-correspondence-format}%
592 %% name Separators
593 \ccSetProperty{editor-suffix-ssl}{(Ed.)}%
594 \ccSetProperty{editor-suffix-pl}{(Eds.)}%
595 \ccSetProperty{editor-suffix}{%
596 \space
597 \ifnum\ccTotalCount=\@ne\relax
598 \ccUseProperty{editor-suffix-ssl}%
599 \else
600 \ccUseProperty{editor-suffix-pl}%
601 \fi
602 }%
603 % Biography
604 % those Properties control how (Role specific) Biography Blocks are formatted, i.e. the list of all
605 % Biographies of a specific Role:
606 \ccSetProperty{role-bio-block-face}{}% face for the entire, role-specific, Biography Block
607 \ccSetProperty{role-bio-block-format}{\ccUseProperty{role-bio-block-face}\ccUseComp{Biography
608 }}\par}% Format of the whole, Role specific, Biography Block
609 \ccPropertyLet{author-bio-block-format}{role-bio-block-format}% Override for single author meta
610 info
611 \ccPropertyLet{editor-bio-block-format}{role-bio-block-format}% Override for single editor meta
612 info
613 \ccPropertyLet{series-editor-bio-block-format}{role-bio-block-format}% Override for single
614 series editor meta info
615 % those Properties control how a (Role specific) Biography is formatted:
616 \ccSetProperty{role-biography-format}{\bfseries\ccUseComp{FullName}:}\space\ccUseComp{Bio}\
617 par}% Format of a single entry in the Role specific Biography

```

```

612 \ccPropertyLet{author-biography-format}{role-biography-format}% Override for single author meta
    info
613 \ccPropertyLet{editor-biography-format}{role-biography-format}% Override for single editor meta
    info
614 \ccPropertyLet{series-editor-biography-format}{role-biography-format}% Override for single
    series editor meta info
615 \ccSetProperty{bio-output-format}{%
616   \ccGetComp{AuthorBioBlock}%
617   \ccGetComp{EditorBioBlock}%
618   \ccGetComp{SeriesEditorBioBlock}%
619 }%
620 % Running headers
621 \ccSetProperty{run-book-title}{%
622   \ccIfComp{RunTitle}
623   {\ccUseComp{RunTitle}}
624   {\ccIfComp{ShortTitle}
625    {\ccUseComp{ShortTitle}}
626    {\ccIfComp{Title}{\ccUseComp{Title}}{No title given!}}}%
627 }%
628 \ccSetProperty{run-book-name}{%
629   \ccIfComp{RunNames}
630   {\ccUseComp{RunNames}}
631   {\ifmonograph
632    \ccIfComp{AuthorNameList}
633    {\ccUseComp{AuthorNameList}}
634    {no author defined!}%
635   \else
636    \ccIfComp{EditorNameList}
637    {\ccUseComp{EditorNameList}}
638    {no editor defined!}%
639   \fi}%
640 }%
641 \ccSetProperty{doc-book-title}{%
642   \ccIfComp{DocTitle}
643   {\ccUseComp{DocTitle}}
644   {\ccIfComp{ShortTitle}
645    {\ccUseComp{ShortTitle}}
646    {\ccUseComp{Title}}}%
647 }%
648 }

```

## 5 Accessibility Features

### 5.1 Output Intent and ICC Profiles

```

649 \ifx\cc@color@enc\relax\else
650 \ccWhenAlly{%

```

First, we declare some Components that represent the three necessary parameters for the output intent:

```

651 \ccAddToType{Components}{titlepage}{%

```

➡ `IccProfileFile` holds the path (relative to the main tex file) and name of the .icc file.

```

652 \ccDeclareGlobalComponent{IccProfileFile}

```



➡ **IccComponents** holds the number of components in the color profile

```
653 \ccDeclareGlobalComponent{IccComponents}
```

➡ **IccIdentifier** holds the identifier of the color profile

```
654 \ccDeclareGlobalComponent{IccIdentifier}}
```

The Components are composed via a new Property **output-intent** which we add to **coco-title**'s Properties list (**\cc@color@enc** is set via the **coco-common** module):

```
655 \ifdefstring\cc@color@enc{cmyk}
656   {\def\cca@default@icc@comp{4}}
657   {\def\cca@default@icc@comp{3}}
658 \ifdefstring\cc@color@enc{cmyk}
659   {\def\cca@default@icc@iden{Coated FOGRA39}}
660   {\def\cca@default@icc@iden{sRGB IEC61966-2.1}}
661 \ccAddToType{Properties}{titlepage}{%
```

**output-intent** <see below> sends the output intent information to the ltpdfa package. It must contain of three data fields:

**profile** with the name of the to-be-embedded **.icc** file,  
**componetns** with an integer telling the pdfwriter how many values are coded by each color (e.g., **4** for cmyk, **3** for rgb)  
**identifier** with the identifying name of the profile (e.g., **Coated FOGRA39** for the included cmyk profile, etc.)

```
662 \ccSetProperty{output-intent}{%
663   profile=\ccIfComp{IccProfileFile}{\ccUseComp{IccProfileFile}}{suppl/\cc@color@enc.icc};%
664   components=\ccIfComp{IccComponents}{\ccUseComp{IccComponents}}{\cca@default@icc@comp};%
665   identifier=\ccIfComp{IccIdentifier}{\ccUseComp{IccIdentifier}}{\cca@default@icc@iden}%
666 }}
```

The Component Handler which links the new Components to that Property is added to titlepage's **cct/document/meta**:

```
667 \AddToHook{cct/document/meta}{\edef\x{\noexpand\ccaAddToConfig{intent}{\ccUseProperty{output-
  intent}}}\x}
```

## 5.2 Encoding of the PDF-A Conformance

As before, the parameters for the PDF conformity level are encoded via specific Components in the titlepage Container:

```
668 \ccAddToType{Components}{titlepage}{%
```

➡ **PDFaID** defines the PDF/A ID (Default: 2, meaning: PDF/A-2)

```
669 \ccDeclareGlobalComponent [2] {PDFaID}%
```

➡ **PDFaLevel** defines the PDF/A Level (Default: A, meaning PDF/A-2A)

```
670 \ccDeclareGlobalComponent [A] {PDFaLevel}%
```

➡ **PDFUAID** defines the PDF standard (Default: 1, meaning: PDF/UA-1). Use **\ccPrefix PDFUAID{}** (i.e. set it to nothing) to make the document conform to the PDF/A standard, but **not** to the PDF/UA standard.


```
671 \ccDeclareGlobalComponent [1] {PDFUAID}}%
```

The checking if the values are valid, and the separation of the various parts of the standard is done via a lua script in the `cct/document/meta`. The `conformance` DocumentInfo nodes are only written, if *neither* `PDFaID`, nor `PDFaLevel` is empty.

```

672 \AddToHook{cct/document/meta}{%
673   \ccIfCompEmpty{PDFaID}{-}{\ccIfCompEmpty{PDFaLevel}{-}{%
674     \edef\x{\noexpand\ccaSetDocinfo{conformance}{%
675       pdfaid=\ccUseComp{PDFaID};%
676       level=\ccUseComp{PDFaLevel}%
677       \ccIfCompEmpty{PDFuaID}{-}{pdfuid=\ccUseComp{PDFuaID}}}%
678     \x}}}
```

### 5.3 Titlepage Specific Role Maps

According to the “Tagged PDF Best Practice Guide” page by the PDF Association, the main title of the document should be mapped to  `<P/>` until the more appropriate  `<Title/>` tag becomes widely accepted with the PDF 2.0 Standard.

```

679 \ccaAddRolemap{Title}{H1}
680 \ccaAddRolemap{Titlepage}{Div}
```

```

681 }%ccWhenAlly
682 \fi
```

```
</title>
```

## Module 10

# coco-floats.dtx

---

Output driver for `coco-floats.sty`.

```
<floats>
```

This module provides handlers for floating objects like tables and figures common to all CoCoTeX projects

Note that we take the term “Float” quite liberally: “Floats” basically mean “*things that may have a caption and which are somewhat outside the main text body*”, whether they actually float (i. e., moved into the `\@toplist` or `\@botlist` by L<sup>A</sup>T<sub>E</sub>X), or not.

```
23 %%
24 %% module for CoCoTeX that extends floating objects.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-floats}
32   [2024/07/16 0.5.0 CoCoTeX floats module]
33 \DeclareOptionX{nofigs}{\global\let\ccf@no@figs\relax}
34 \ProcessOptionsX
```

## 1 Package Setup

### 1.1 Hard requirements

For the list-of mechanism, we need the CoCoTeX common module, which also loads the CoCoTeX kernel module.

```
35 \RequirePackage{coco-common}
```

For landscape images, we load the `rotating` package.

```
36 \RequirePackage{rotating}
```

Since file names form word often contain spaces and periods, we also include the `grffile` package.

```
37 \RequirePackage{grffile}
```

In order to save footnotes in captions, we require the `footnote` package.

```
38 \RequirePackage{footnote}
```

The `adjustbox` package is needed to restrict the maximum dimensions of image files.

```
39 \RequirePackage[Export]{adjustbox}
```

Finally, we need the `stfloats` package to allow bottom placed images on pages that start L<sup>A</sup>T<sub>E</sub>X's twocolumn mode.

```
40 \usepackage{stfloats}
41 \setcounter{dblbotnumber}{5}
```

## 1.2 Adjustments at the Beginning of the Document

```
42 \AtBeginDocument{%
```

The first adjustment implements the `nofigs` option by deactivating the `\includegraphics` macro.

```
43 \ifx\ccf@no@figs\relax
44 \renewcommand\includegraphics[2] [] {}%
45 \fi
```

`\ccf@ltx@includegraphics` stores the final definition of the `\includegraphics` macro for later use.

```
46 \global\let\ccf@ltx@includegraphics\includegraphics
```

Adjustments to the `htmltabs` package, if it is used:

```
47 \@ifpackageloaded{htmltabs}
48 {\global\let\cc@uses@htmltabs\relax
49 \def\ht@adjust@linewidth{%
50 \advance\ht@h@offset\leftskip
51 \advance\ht@h@offset\@totalleftmargin
52 \advance\linewidth-\rightskip
53 }%
54 }{}}%
```

In order to catch the actual dimensions of the float box, we need to hook into L<sup>A</sup>T<sub>E</sub>X's `\@endfloatbox` macro. This macro is low-level enough so it covers regular, double-column, and rotated floats. Those values will later be written into the `.aux` file for each float. The values, together with the float's overall width, are stored in a macro called `cc-float-\the\ccf@int@cnt-dimens`.

```
55 \gappto\@endfloatbox{%
56 \global\ccf@total@height=\ht\@currbox\relax%
57 \global\ccf@total@depth=\dp\@currbox\relax%
58 }%
59 }%
```

## 1.3 Document Class-Option Overrides

Since CoCoTeX is mainly developed for automatic typesetting and float positioning, we set rather high tolerances for macros from L<sup>A</sup>T<sub>E</sub>X's standard `.clo` files:

```
60 \def\topfraction{0.9}
61 \def\textfraction{0.1}
62 \def\bottomfraction{0.8}
63 \def\totalnumber{8}
64 \def\topnumber{8}
65 \def\bottomnumber{8}
66 \def\floatpagefraction{0.8}
67 \@fptop\z@
68 \@fpbot\@flushglue
```

## 1.4 Internal Registers

`\ccf@floatbox` is for measuring the dimensions of the whole float

```
69 \newbox \ccf@floatbox
```

`\ccf@sub@box` is for measuring a single sub-float.

```
70 \newbox \ccf@sub@box
```

`\ccf@int@cnt` is an internal global counter that numbers all top-level floats sequentially.

```
71 \newcount\ccf@int@cnt \ccf@int@cnt\z@
```

`\ccSubFloatCnt` counts the sub-floats within a parent float Container instance.

```
72 \newcount\ccSubFloatCnt \ccSubFloatCnt=\z@\relax
```

`\ccf@int@sub@flt@cnt` is a temporary counter that holds the total number of subfloats inside a parent float Container instance.

```
73 \newcount\ccf@int@sub@flt@cnt \ccf@int@sub@flt@cnt\z@
```

Various dimension registers that store dimensions and spaces of floats and sub-floats:

`\ccf@sub@maxheight` stores and self-updates the height of the largest sub-float inside a float

```
74 \newdimen\ccf@sub@maxheight \ccf@sub@maxheight=\z@\relax
```

`\ccf@sub@sep` is the space between sub-floats

```
75 \newdimen\ccf@sub@sep \ccf@sub@sep=\fboxsep\relax
```

`\ccf@total@width` stores the cumulated overall width of the entire float

```
76 \newdimen\ccf@total@width \ccf@total@width=\textwidth\relax
```

`\ccf@calc@width` is an internal dimension used to calculate the ratio between mutiple sub-floats that should be scaled to the same height

```
77 \newdimen\ccf@total@height \ccf@total@height=\textwidth\relax
```

`\ccf@total@height` is the overall height of a float

```
78 \newdimen\ccf@total@depth \ccf@total@depth=\textwidth\relax
```

`\ccf@total@depth` is the overall depth of a float

```
79 \newdimen\ccf@calc@width \ccf@calc@width=\ccf@total@width\relax
```

`\ccf@sep@top` holds the actual vertical skip inserted at the top of a float. If the float is floating, this equals to `intext-skip`, or `float-skip`, otherwise.

```
80 \newskip\ccf@sep@top \ccf@sep@top=\z@\relax
```

`\ccf@sep@bottom` is the same for the bottom vertical skip.

```
81 \newskip\ccf@sep@bottom \ccf@sep@bottom=\z@\relax
```

Internal dimensions for the horizontal margins:

`\ccf@margin@r` holds the right side margin

```
82 \newdimen\ccf@margin@r \ccf@margin@r=\z@\relax
```

`\ccf@margin@l` holds the left side margin

```
83 \newdimen\ccf@margin@l \ccf@margin@l=\z@\relax
```

`\ccf@margin@i` holds the inner margin

```
84 \newdimen\ccf@margin@i \ccf@margin@i=\z@\relax
```

`\ccf@margin@o` holds the outer margin

```
85 \newdimen\ccf@margin@o \ccf@margin@o=\z@\relax
```

`\ifccf@break@capt` is a locally adjustable switch that indicates whether captions are allowed to break across pages (true) or not (false).

```
86 \newif\ifccf@break@capt \ccf@break@captfalse
```

`\ifccf@sameheight` determines if subfloats should be scaled such that they are all the same height.

```
87 \newif\ifccf@sameheight \ccf@sameheighttrue
```

## 2 Internal macros

### 2.1 Generic resetter

Some macros are re-evaluated for each new top-level float.

`\ccf@reset@defaults` resets the those macros. It is called at the very beginning of each new float.

```
88 \def\ccf@reset@defaults{%
89   \global\ccSubFloatCnt=\z@
90   \global\ccf@total@width=\z@
91   \global\let\ccf@has@capt@top\@undefined
92   \global\let\ccf@has@capt@bottom\@undefined
93   \global\let\ccf@has@subcapt@top\@undefined
94   \global\let\ccf@has@subcapt@bottom\@undefined
95   \global\let\ccf@sub@contentsline@store\@empty
96   \global\ccf@sub@maxheight=\z@\relax
97   \@tempcnta=\z@\relax
98   \cc@reset@components{\cc@cur@cont}%
99   \let\ccf@prefix\@empty
100   \let\ht@cur@element\ccfCapType
101   \global\let\ccf@current@class\relax
102 }
```

## 2.2 Wrapper for L<sup>A</sup>T<sub>E</sub>X's Native float Environments

`\ccf@set@env` determines the low-level L<sup>A</sup>T<sub>E</sub>X float environment depending on orientation and document options. If no `float-pos` is given (implicitly or determined), the object is not treated as a float at all.

```

103 \def\ccf@set@env{%
104   \ifx\ccf@floatpos\empty
105     \let\ccf@begin@env\bgroup
106     \let\ccf@end@env\egroup
107   \else
108     \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
109     {\edef\ccf@env@name{sideways\ccfCapType}%
110      \edef\ccf@begin@env{\noexpand\begin{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}%
111      \edef\ccf@end@env{\noexpand\end{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}}
112     {\edef\ccf@env@name{\ifx\ccf@do@dbl\relax db\fi float}%
113      \edef\ccf@begin@env{\expandafter\noexpand\csname @x\ccf@env@name\endcsname {\ccfCapType}[\ccf@floatpos]}%
114      \edef\ccf@end@env{\expandafter\noexpand\csname end@\ccf@env@name\endcsname}}%
115   \fi}

```

`\ccf@get@seps` determines the top and bottom skips dependent on float position and orientation

```

116 \def\ccf@get@seps{%
117   \ifx\ccf@floatpos\empty
118     \expandafter\ccf@sep@top\dimexpr\ccUseProperty{intext-skip-top}\relax%
119   \else
120     \expandafter\ccf@sep@top\dimexpr\ccUseProperty{float-skip-top}\relax%
121   \fi
122   \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}{%
123     \ifx\ccf@floatpos\empty
124       \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{intext-skip-bottom}\relax%
125     \else
126       \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{float-skip-bottom}\relax%
127     \fi}}

```

`\ccf@set*@sep` Hooks to apply top and bottom skips, respectively.

```

128 \def\ccf@set@top@sep{\addvspace{\ccf@sep@top}}
129 \def\ccf@set@bot@sep{\addvspace{\ccf@sep@bottom}}

```

## 3 The Generic float Container

Components in Containers that are derived from the abstract `float` are in fact all Counted Components, where top-level instances use 0 as their internal counter and sub-floats are counted incrementally. Thus, we can *simplify* the **internal** names to `<Componentname>--<Counter>`, which is done via a custom wrapper for the `\cc@def@counted@comp` Component declarator.

`\ccfMakeComp` is a shortcut for float Component declarations.

`{#1}` is the generic name of the Component.

```

130 \def\ccfMakeComp#1{%
131   \cc@def@counted@comp{#1-\the\ccSubFloatCnt}{#1}{}%
132 }

```

`\ccfMakeCompL` is a shortcut to declare Float Components together with their *list-of* overrides.

`{#1}` is the generic name of the Component.

```
133 \def\ccfMakeCompL#1{%
134   \ccfMakeComp{#1}%
135   \ccfMakeComp{Listof#1}}
```

`float` is the main parent Container for all floats.

```
136 \ccDeclareContainer{float}{%
```

### 3.1 Common Float Components

```
137   \ccDeclareType{Components}{%
```

First, we set the naming scheme of the internal Component macros which is then valid for all Component declarations by locally re-defining `\cc@counted@comp@scheme`.

```
138   \def\cc@counted@comp@scheme#1{#1-\the\ccSubFloatCnt}%
```

`Content` is the main content holder of a float.

```
139   \ccfMakeComp{Content}%
```

`Caption` is the main caption of a float.

`ListofCaptionOR` is the corresponding list-of-entry

```
140   \ccfMakeCompL{Caption}%
```

`Legend` is a legend to a float.

`ListofLegendOR` is the corresponding list-of-entry

```
141   \ccfMakeCompL{Legend}%
```

`Source` is the source of a float.

`ListofSourceOR` is the corresponding list-of-entry

```
142   \ccfMakeCompL{Source}%
```

`Number` is the counter of the float (including the label)

`ListofNumberOR` is the corresponding list-of-entry

```
143   \ccfMakeCompL{Number}%
```

`RefLabel` is the float's ID used for cross-references (replaces L<sup>A</sup>T<sub>E</sub>X's `\label` command)

```
144   \ccfMakeComp{RefLabel}%
```



`ListofEntryCL` is the Collection Component for the entire `Listof` entry.

```
145 \ccfMakeComp{ListofEntry}%
146 }%
```

## 3.2 Common Float Properties

```
147 \ccDeclareType{Properties}{%
```

### Placement and Spacing

`intext-skip-top` <skip> vertical space between the text body and following non-floating floats

```
148 \ccSetProperty{intext-skip-top}{\intextsep}%
```

`intext-skip-bottom` <skip> vertical space between non-floating floats and the following text body

```
149 \ccSetProperty{intext-skip-bottom}{\intextsep}%
```

`float-skip-top` <skip> vertical space between text body and following floating floats

```
150 \ccSetProperty{float-skip-top}{\z@}%
```

`float-skip-bottom` <skip> vertical space between floating floats and following text body

```
151 \ccSetProperty{float-skip-bottom}{\z@}%
```

`sub-float-sep` <skip> horizontal space between sub-floats

```
152 \ccSetProperty{sub-float-sep}{\ccf@sub@sep}%
```

`margin-inner` <skip> inner margins of floats in twopage mode, i. e., left margin on odd pages and right margin on even pages, respectively.

```
153 \ccSetProperty{margin-inner}{\z@}%
```

`margin-outer` <skip> outer margin of floats in twopage mode, i. e., right margin on odd pages and left margin on even pages, respectively.

```
154 \ccSetProperty{margin-outer}{\z@}%
```

`margin-left` <skip> horizontal space between the left page area boundary and the float.

```
155 \ccSetProperty{margin-left}{\z@}%
```

`margin-right` <skip> horizontal space between the right page area boundary and the float.

```
156 \ccSetProperty{margin-right}{\z@}%
```

`before-float` <any> is the code that is executed before a float's content is evaluated.

```
157 \ccSetProperty{before-float}{\parindent\z@}%
```

### Properties for Float-Type Handlers

`subfloat-content` <any> is the material that is put into the `\ccf@sub@box` for further processing.

```
158 \ccSetProperty{subfloat-content}{\ccUseComp{Content}}%
```

**float-render** <any> the output routine for top-level float type specific contents

```
159 \ccSetProperty{float-render}{\ccUseComp{Content}}%
```

**subfloat-render** <any> the output routine for second-level float type specific contents.

```
160 \ccSetProperty{subfloat-render}{\ccUseComp{Content}}%
```

### Properties for Captions

**caption-face** <any> style applied to both top and bottom placed captions

```
161 \ccSetProperty{caption-face}{}%
```

**caption-face-top** <any> style applied to top placed captions only

```
162 \ccSetProperty{caption-face-top}{}%
```

**caption-face-bottom** <any> style applied to bottom placed captions only

```
163 \ccSetProperty{caption-face-bottom}{}%
```

**source-face** <any> style applied to the printed **Source** Component.

```
164 \ccSetProperty{source-face}{}%
```

**legend-face** <any> style applied to the printed **Legend** Component.

```
165 \ccSetProperty{legend-face}{}%
```

**caption-sep-top** <skip> vertical space between top caption and content, i. e., the skip *after* the top placed caption.

```
166 \ccSetProperty{caption-sep-top}{\z@}%
```

**caption-sep-bottom** <skip> vertical space between bottom caption and content, i.e., the skip *before* the bottom placed caption.

```
167 \ccSetProperty{caption-sep-bottom}{\z@}%
```

**caption-top** <any> the content of the top placed caption

```
168 \ccSetProperty{caption-top}{%
169   \ccIfComp{Number}{\{\ccUseProperty{number-face}\ccUseComp{Number}\ccUseProperty{number-sep
170     }\}}{}%
171   \ccUseComp{Caption}%
172 }%
```

**caption-bottom** <any> the content of the bottom placed caption

```
172 \ccSetProperty{caption-bottom}{%
173   \ccIfComp{Legend}{\{\ccUseProperty{legend-face}\ccUseComp{Legend}\}}{}%
174   \ccIfComp{Source}{%
175     \ccIfComp{Legend}{\par\nopagebreak}{}%
176     {\ccUseProperty{source-face}%
177       \ccUseComp{Source}\}}{}%
178 }
```

**subcaption-face** <any> the style of captions of second level floats

```
178 \ccPropertyLet{subcaption-face}{caption-face}%
```

**subcaption-face-top** <any> the style of top placed captions of second level floats

```
179 \ccSetProperty{subcaption-face-top}{\ccUseProperty{caption-face-top}}%
```

**subcaption-face-bottom** <any> the style of bottom placed captions of second level floats

```
180 \ccSetProperty{subcaption-face-bottom}{\ccUseProperty{caption-face-bottom}}%
```

**subcaption-add-sep-top** <skip> additional vertical space between top caption and top sub-caption

```
181 \ccSetProperty{subcaption-add-sep-top}{\z@}%
```

**subcaption-add-sep-bottom** <skip> additional vertical space between bottom sub-caption and bottom caption

```
182 \ccSetProperty{subcaption-add-sep-bottom}{\z@}%
```

**subcaption-sep-top** <skip> vertical space between top placed sub-captions and content, i. e., the space *after* top placed sub-captions.

```
183 \ccSetProperty{subcaption-sep-top}{\ccUseProperty{caption-sep-top}}%
```

**subcaption-sep-bottom** <skip> vertical space between content and top placed sub-captions, i. e., the space *before* bottom placed sub-captions.

```
184 \ccSetProperty{subcaption-sep-bottom}{\ccUseProperty{caption-sep-bottom}}%
```

**subcaption-top** <any> the content of top placed sub-captions

```
185 \ccSetProperty{subcaption-top}{\ccUseProperty{caption-top}}%
```

**subcaption-bottom** <any> the content of bottom placed sub-captions

```
186 \ccSetProperty{subcaption-bottom}{\ccUseProperty{caption-bottom}}%
```

**subcaption-valign-top** [top|bottom|middle] vertical alignment of neighboring top-placed sub-captions

```
187 \ccSetProperty{subcaption-valign-top}{top}%
```

**subcaption-valign-bottom** [top|bottom|middle] vertical alignment of neighboring bottom-placed sub-captions

```
188 \ccSetProperty{subcaption-valign-bottom}{top}%
```

## Properties for Counters

**auto-number-prefix** <any> Prefix for auto-generated Number components

```
189 \ccSetProperty{auto-number-prefix}{\csname\ccfCapType name\endcsname}%
```

**auto-number-prefix-sep** <any> Separator between the auto-generated number prefix and the auto-generated Number component.

```
190 \ccSetProperty{auto-number-prefix-sep}{~}%
```

**numbering** [auto|<any>] if auto, float counters in instances without the Number component are generated automatically. Any other value suppresses auto-numbering.

```
191 \ccSetProperty{numbering}{auto}%
```

**numbering** [auto|<any>] if auto, subfloat counters in instances without the Number component are generated automatically. Any other value suppresses auto-numbering.

**Note:** this Property has only effect when subfloats are second-level. In first-level sub-floats, the numbering Property is used.

192 `\ccSetProperty{sub-numbering}{}%`

**number-sep** <any> separator between the printed float number and the caption

193 `\ccSetProperty{number-sep}{\enskip}%`

**number-face** <any> style of number, additional to caption-format

194 `\ccSetProperty{number-face}{\bfseries}%`

**sub-number-sep** <any> separator between number and caption in sub-floats

195 `\ccSetProperty{sub-number-sep}{\,}%`

**sub-number-style** [arabic|Alph|alph|roman|Roman] numbering style for automatically generated subfloat counters

196 `\ccSetProperty{sub-number-style}{alph}%`

**sub-number-face** <any> style of the number of a subfloat

197 `\ccSetProperty{sub-number-face}{}%`

**sub-number-before** <any> stuff that is put immediately before the automatically generated subfloat counter

198 `\ccSetProperty{sub-number-before}{(}%`

**sub-number-after** <any> stuff that is put immediately after the automatically generated subfloat counter

199 `\ccSetProperty{sub-number-after}{})%`

**sub-number-format** <any> the format of the number

200 `\ccSetProperty{sub-number-format}{%`  
 201 `\ccUseProperty{float-number}%`  
 202 `\ccUseProperty{sub-number-sep}%`  
 203 `\ccUseProperty{sub-number}}%`

**label-pos** [top|bottom] position of the cross reference anchor, referring to top or bottom placed captions.

204 `\ccSetProperty{label-pos}{top}%`

**sublabel-pos** [top|bottom] position of the cross reference anchor for sub-floats, referring to top or bottom placed sub-captions.

205 `\ccSetProperty{sublabel-pos}{top}%`

### Properties for List-Of Entries

**list-of-page-sep** <any> separator between the listof-entry and the page

206 `\ccSetProperty{list-of-page-sep}{\dotfill}%`

`list-of-number-face` <any> style of the listof-entry

```
207 \ccPropertyLet{list-of-number-face}{list-of-caption-face}%
```

`list-of-number-sep` <any> separator between the number and the listof entry.

```
208 \ccSetProperty{list-of-number-sep}{\enskip}%
```

`list-of-number-align` [left|center|right] horizontal alignment of the listof number within its local hbox.

```
209 \ccSetProperty{list-of-number-align}{left}%
```

`list-of-number-format` <any> format of the number in listof entries.

```
210 \ccSetProperty{list-of-number-format}{%
211   \bgroup
212   \ccUseProperty{list-of-number-face}%
213   \ccUseComp{ListofNumber}%
214   \ccUseProperty{list-of-number-sep}%
215   \egroup}%
```

`list-of-parfillskip` <skip> parfillskip of an entry in the listof

```
216 \ccSetProperty{list-of-parfillskip}{-\rightskip}%
```

`list-of-margin-right` <skip> right margin of the listof entry

```
217 \ccSetProperty{list-of-margin-right}{\@pnumwidth \@plus 1fil}%
```

`list-of-margin-left` [auto|<skip>] right margin of the listof entry

```
218 \ccSetProperty{list-of-margin-left}{auto}%
```

`list-of-indent` [auto|<dimen>] horizontal offset of the first line of an listof-entry, relative to margin-left.

```
219 \ccSetProperty{list-of-indent}{auto}%
```

`list-of-block` <any> format of the entire listof entry.

```
220 \ccSetProperty{list-of-block}{%
221   \ccUseProperty{list-of-caption-face}%
222   \ccIfComp{ListofNumber}
223     {\ccUseComp{list-of-hang-number}}
224     {\leftskip0pt}%
225   \ccUseComp{ListofCaption}%
226   \ccUseProperty{list-of-page-sep}\ccUseComp{ListofPage}%
227 }%
```

`list-of-before-entry` <any> material inserted at the beginning of each listof entry

```
228 \ccSetProperty{list-of-before-entry}{%
229   \ccGobble
230   \leftskip\ccUseProperty{list-of-margin-left}\relax%
231   \rightskip \ccUseProperty{list-of-margin-right}\relax%
232   \parfillskip \ccUseProperty{list-of-parfillskip}\relax
233   \parindent\z@
234   \@afterindenttrue
235   \interlinepenalty\@M
236   \leavevmode
237   \null\nobreak
238 }% list-of-float appearance
```

`list-of-after-entry` <any> material inserted at the end of a listof entry.

```
239 \ccSetProperty{list-of-after-entry}{\par}%
240 }% /Properties
241 \ccDeclareType{Attributes}{%
```

`class` <string> is the class of the Float.

```
242 \ccDeclareAttributeHandler{class}{\ccf@attr@class{\ccAttrVal}}%
```

`break-caption` <bool> whether or not the caption is allowed to break across pages

```
243 \ccDeclareAttributeHandler*{break-caption}[\ccf@break@captfalse]{\ccf@break@capttrue}%
```

`float-pos` [(h|t|p|b)\*|h!] the float position of the float. h! means that the float is not actually floating and is equivalent to omitting the Attribute.

```
244 \ccDeclareAttributeHandler{float-pos}[\let\ccf@floatpos\empty]{\ccf@attr@pos{\ccAttrVal}}%
```

`orientation` landscape is whether the float is rotated by 90° (landscape) or not (if omitted, default)

```
245 \ccDeclareAttributeHandler{orientation}{\ccf@attr@orient{\ccAttrVal}}
```

`debug` <flag> if set, additional debugging is written into the shell and log file.

```
246 \ccDeclareAttributeHandler{debug}[\let\ccf@debug\relax]{\let\ccf@debug\ccf@attr@debug}
```

`no-same-height` <flag> if set, the same-height calculations are de-activated for this float.

```
247 \ccDeclareAttributeHandler{no-same-height}{\@ccf@sameheightfalse}%
```

```
248 }
249 }% /Container
```

### 3.3 The Generic float Environment

This section defines the macros for the float's Container-specific LaTeX environment.

`\ccf@float` is a mid-level Macro that provides the common floating L<sup>A</sup>T<sub>E</sub>X environment. #1 is the float environment's kv-attribute list.

#1 float position (optional)

```
250 \def\ccf@float{\cc@opt@empty\@ccf@float}
251 \def\@ccf@float[#1]{%
252   \par
253   \begingroup
254   \@cc@is@finalfalse
255   \global\advance\ccf@int@cnt\@ne
256   \ccEvalType{FloatEnvInfo}%
257   \ccf@reset@defaults
258   \ccToggleCountedConditionals
259   \ccEvalType{Properties}%
260   \ccIfPropVal{subfloat-same-height}{true}{\global\@ccf@sameheighttrue}{\global\@ccf@sameheightfalse}
```

```

261 \ccEvalAttributes{#1}%
262 \ccf@eval@class
263 \ccf@set@hsize
264 \ccf@get@seps
265 \ccEvalType{Components}%
266 \ccUseProperty{before-float}%
267 \ccf@set@env
268 \ifx\ccf@floatpos\empty\else\savenotes\fi
269 \@cc@is@finaltrue
270 \ignorespaces}

```

`\endccf@float` is the end of the common float environment.

```

271 \def\endccf@float{%
272 \ccf@begin@env
273 \@cc@is@finalfalse
274 \ccf@set@top@sep
275 \ccf@int@sub@flt@cnt=\ccSubFloatCnt\relax
276 \ccSubFloatCnt=\z@\relax
277 \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
278 {\ccf@create@counter
279 \ccf@compose@listof}%
280 \ccSubFloatCnt=\ccf@int@sub@flt@cnt\relax
281 \ccf@test@caption{0}{\top}%
282 \ccf@test@caption{0}{\bottom}%
283 \bgroup
284 \@cc@is@finaltrue

```

Floats as a whole are tagged as `<Aside/>` when PDF/UA standard is 2.0, or else as `<Div/>`.

```

285 \ccIfCompFromVal{titlepage}{PDFUAID}{2}
286 {\ccaStructStart{Aside}}
287 {\ccaStructStart{Div}}}%
288 \edef\ccf@parstruct@id{\ccaGetCurStruct{idx}}%
289 \hsize\ccf@total@width
290 \ccf@process
291 \ccIfCompFromVal{titlepage}{PDFUAID}{2}
292 {\ccaStructEnd{Aside}}
293 {\ccaStructEnd{Div}}}%
294 \par
295 \egroup
296 \ccSavePage
297 \ccf@set@bot@sep
298 \ccf@end@env
299 \ccf@debug%
300 \ifx\ccf@floatpos\empty\else\spewnotes\fi
301 \endgroup
302 \ccf@store@dimens
303 \global\let\ccf@current@class\relax
304 }

```

`\ccf@store@dimens` writes the float's final dimensions into the aux file.

```

305 \def\ccf@store@dimens{%
306 \immediate\write@auxout
307 {\string\expandafter\string\gdef\string\csname\space cc-float-\the\ccf@int@cnt-dimens\string
308 \endcsname{%
309 {\the\ccf@total@width}%
310 {\the\ccf@total@height}%
311 {\the\ccf@total@depth}%

```

```

311   }}%
312 }

```

### 3.4 The SubFloat Environment

#### The SubFloat Sub-Container

Second-level floats (or SubFloats) are sub-containers of the float container.

`\ccSubFloat` is the user-level environment for sub-floats

```

313 \def\ccSubFloat{%
314   \ifx\ccf@is@subfloat\relax
315     \PackageError{coco-floats.sty}{Nested ccSubFloats detected!}{You cannot (yet) nest a `
      ccSubFloat' environment into another `ccSubFloat' environment!}%
316   \else
317     \global\let\ccf@is@subfloat\relax
318     \global\advance\ccSubFloatCnt\@ne
319   \fi
320   \global\cslet\ccf@made@label@for@\the\ccSubFloatCnt\relax
321   \ignorespaces}

```

`\endccSubFloat` is the end of the sub-float environment

```

322 \def\endccSubFloat{%
323   \ifhmode\unskip\fi
324   \setbox\ccf@sub@box\hbox{\ccGobble
325     \let\includegraphics\ccf@measuresubgraphics
326     \ccUseProperty{subfloat-content}}%
327   }%
328   \expandafter\xdef\csname ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcsname{\the\wd\
      ccf@sub@box}%
329   \expandafter\xdef\csname ccf@\cc@cur@cont @height-\the\ccSubFloatCnt\endcsname{\the\ht\
      ccf@sub@box}%
330   \expandafter\xdef\csname ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcsname{\the\dp\
      ccf@sub@box}%
331   \@tempdima=\dimexpr\the\ht\ccf@sub@box+\the\dp\ccf@sub@box\relax
332   \@tempdimb=\dimexpr\the\wd\ccf@sub@box\relax
333   \ifdim\@tempdima>\ccf@sub@maxheight\relax
334     \global\ccf@sub@maxheight=\@tempdima\relax
335   \fi
336   \global\setbox\ccf@sub@box\box\voidb@x
337   \global\let\ccf@is@subfloat\@undefined
338   \aftergroup\ignorespaces
339 }

```

#### Printing the Subfloats

`\ccfRenderSubFloats` iterates through the single sub-floats and renders them in a nice row. #1 is the subfloat counter, #2 is the Component name that contains the actual contents of the sub-float, for `\ccPrefix Figure` it is Fig, for `\ccPrefix Table` it is Content.

```

340 \long\def\ccfRenderSubFloats#1#2{%
341   \leavevmode
342   \savenotes
343   \ifnum#1>\@ne\hfill\fi
344   \vtop\bgroup
345     \expandafter\hsize\csname cc@\cc@cur@cont @res@width-#1\endcsname\relax

```



```

346 \let\includegraphics\ccf@includesubgraphics
347 \leavevmode
348 \ccf@render@sub{#1}{#2}%
349 \egroup
350 \spewnotes
351 }

```

`\ccf@render@sub` renders a single sub-float. For the arguments, see `\ccfRenderSubFloats`, above.

```

352 \long\def\ccf@render@sub#1#2{%
353 \ccSubFloatCnt=#1\relax
354 \ccf@make@subcaption{top}%
355 \bgroup\strut\ccUseComp{#2}\strut\par\egroup%
356 \ccf@make@subcaption{bottom}}

```

### 3.5 Attribute Handlers

The following macros handle the Attributes of Float Container instances.

`\ccf@attr@class` handles the style class of the float.

`{#1}` is the value of the “class” Attribute.

```

357 \def\ccf@attr@class#1{%
358 \gdef\ccf@current@class{#1}%%
359 }

```

`\ccf@eval@class` expands the style class specific Properties.

```

360 \def\ccf@eval@class{%
361 \ccUseStyleClass{default}{\ccfCapType}%
362 \ifx\ccf@current@class\relax\else
363 \ccUseStyleClass{\ccf@current@class}{\ccfCapType}%
364 \fi}

```

`\ccf@attr@pos` is the handler for determining the float position. Some float Properties and Attributes restrict and override the explicit float positions, e.g., fully rotated floats must be positioned in `p` mode (i.e., as float page).

`{#1}` is the value of the float-pos Attribute. It may be any combination of `h`, `t`, `p`, `b`; or `h!`, which means that the float is non-floating (which is equivalent to an omitted `float-pos` Attribute)

```

365 \def\ccf@attr@pos#1{%
366 \edef\ccf@floatpos{#1}%
367 \def\@tempa{h!}\ifx\ccf@floatpos\@tempa\let\ccf@floatpos\empty\fi
368 \def\@tempa{h}\ifx\ccf@floatpos\@tempa\def\ccf@floatpos{htbp!}\fi
369 \ifx\ccf@do@dbl\relax
370 \ifx\ccf@floatpos\empty\def\ccf@floatpos{htpb!}\fi% 11514
371 \linewidth\dimexpr2\columnwidth+\columnsep\relax
372 \hsize\linewidth\relax
373 \fi
374 }

```

`\ccf@attr@orient` is the handler for the orientation Attribute.

`{#1}` is the value of the orientation Attribute. Currently, the only value that does things is `landscape`.

```

375 \def\ccf@attr@orient#1{%

```

```

376 \ccIfAttrIsStr{#1}{orientation}{landscape}
377   {\linewidth\textheight
378    \hsize\linewidth
379    \def\ccf@floatpos{p}}{}}

```

`\ccf@attr@debug` prints some debug information to `stdout` for a single float that has the Attribute `debug` set.

```

380 \def\ccf@attr@debug{%
381   \message{^^J[CoCo Float Debug]^^J
382     Textheight:\space\the\textheight^^J
383     Type:\space\space\space\space\space\space\space\cc@cur@cont^^J
384   \ifx\ccfCapType\cc@str@figure
385     Path:\space\space\space\space\space\space\space\ccf@fig@path^^J
386   \fi
387     Class:\space\space\space\space\space\space\space\ccf@current@class^^J
388     Floatpos:\space\space\space\ccf@floatpos^^J
389     Environ:\space\space\space\space\expandafter\noexpand\ccf@begin@env...\expandafter\noexpand
390       \ccf@end@env^^J
391     Subfloat:\space\space\space \the\ccSubFloatCnt^^J
392   \ifnum\ccSubFloatCnt=\z@
393     Width:\space\space\space\space\space\space\the\ccf@total@width^^J
394     Height:\space\space\space\space\space\space\the\ccf@total@height^^J
395     Depth:\space\space\space\space\space\space\the\ccf@total@depth^^J
396   \else
397     Width \the\ccSubFloatCnt:\space\space\space\space\space\space\space\expandafter\meaning\cename
398       ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcename^^J
399     Height \the\ccSubFloatCnt:\space\space\space\space\space\space \expandafter\meaning\cename ccf@\
400       cc@cur@cont @height-\the\ccSubFloatCnt\endcename^^J
401     Depth \the\ccSubFloatCnt:\space\space\space\space\space\space\space\expandafter\meaning\cename
402       ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcename^^J
403   \fi}}

```

### 3.6 Handling of List-of Entries

`\ccf@generate@listof@handlers` generates handlers for listof-entries.

- #1 is the file ending
- #2 is the caption type
- #3 is the Container name

```

400 \def\ccf@generate@listof@handlers#1#2#3{%

```

`cc@listof@extract@data` The first macro that is dynamically defined, is the Component collector.

- ##1 is a numeric level that represents the order of the listof-entries
- ##2 is the caption type
- ##3 is the content of the `l@<level>` macro
- ##4 is the page number associated with that entry.

```

401 \expandafter\gdef\cename cc@#1@extract@data\endcename##1##2##3##4{%
402   \ccSetContainer{#3}%
403   \ccEvalType[#3]{Properties}%
404   \ccDeclareComponent{ListofCaption}{-}{-}%
405   \ccDeclareComponent{ListofLegend}{-}{-}%
406   \ccDeclareComponent{ListofSource}{-}{-}%
407   \ccDeclareComponent{ListofNumber}{-}{-}%
408   \ccDeclareComponent{ListofPage}{-}{-}%

```

```

409 \ccComponent{ListofPage}{\ccUseProperty{list-of-page-face}##4}%
410 \cc@expand@l@contents{##3}{##3}{Listof}{Caption}%
411 \cc@format@number{list-of-}{Listof}{##1}%
412 }%

```

`\cc@listof@print@entry` The second dynamically defined macro is the entry renderer. It applies the Listof properties and selects the components to be printed. ##1 is the caption type of the float.

```

413 \expandafter\gdef\csname cc@#1@print@entry\endcsname##1{%
414   \bgroup
415   \ccUseHook{list-of-before-hook-##1}%
416   \ccUseProperty{list-of-before-entry}%
417   \ccUseProperty{list-of-block}%
418   \ccUseHook{list-of-after-hook-##1}%
419   \ccUseProperty{list-of-after-entry}%
420   \egroup}%
421 }

```

`\ccf@addcontentsline` fork of L<sup>A</sup>T<sub>E</sub>X's `\addtocontents` macro.

```

422 \def\ccf@addcontentsline{%
423   \ccWhenComp{ListofEntry}{%
424     \protected@write\@auxout
425       {\ccGobble}%
426       {\string\@writefile{\ccf@cap@list@type}
427         {\protect\ccContentsline
428           {\ifnum\ccSubFloatCnt>\z@\ccIfAttr{\ccfCapType}{subfloat}{sub}{\fi\ccfCapType}
429           {\ccUseComp{ListofEntry}}
430           {\thepage}
431           {\@currentHref}\protected@file@percent}}\relax}}

```

`\ccf@check@empty` is a wrapper for CoCoTeX kernel's `\cc@check@empty`

```

432 \def\ccf@check@empty#1{\cc@check@empty{\cc@cur@cont}{#1-\the\ccSubFloatCnt}{Listof}}

```

`\ccf@compose@listof` is the Component Group Handler for Listof Components.

```

433 \def\ccf@compose@listof{%
434   \ccf@check@empty{Number}%
435   \ccf@check@empty{Caption}%
436   \ccf@check@empty{Legend}%
437   \ccf@check@empty{Source}%
438   \let\ccf@listof@entry\relax
439   \ccWhenComp{ListofCaption}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofCaption}{\ccUseComp{ListofCaption}}}}%
440   \ccWhenComp{ListofNumber}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofNumber}{\ccUseComp{ListofNumber}}}}%
441   \ccWhenComp{ListofLegend}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofLegend}{\ccUseComp{ListofLegend}}}}%
442   \ccWhenComp{ListofSource}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofSource}{\ccUseComp{ListofSource}}}}%
443   \ifx\ccf@listof@entry\relax\else
444     \bgroup
445     \ccGobble
446     \protected@edef\ccf@listof@entry{\ccf@listof@entry}%
447     \ccComponentEA{ListofEntry}{\ccf@listof@entry}%
448     \egroup
449   \fi
450 }%

```

`\ccf@write@listof` The last macro to be defined here is the list-of writer. This macro is responsible to write the entry into TeX's auxiliary file system.

```

451 \def\ccf@write@listof{%
452   \ccUnlessAttr{\ccfCapType}{nolist}
453   {\ifnum\ccSubFloatCnt=\z@\relax
454     \ccIfAttr{\ccfCapType}{subfloat}
455     {\ccSubFloatCnt=\z@\relax
456       \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
457       {\ccf@addcontentsline}}%
458     {\ccf@addcontentsline}%
459   \else
460     \ccIfAttr{\ccfCapType}{subfloat}{\ccf@addcontentsline}%
461   \fi}%
462 }
```

### 3.7 Label and Referencing mechanisms

#### Generation of Number Components

`\ccf@create@counter` checks for the various parameters that control whether or not a Number component is auto-generated for each sub-float.

```

463 \def\ccf@create@counter{%
464   \ccIfAttrIsSet{\ccfCapType}{nonumber}{\}
465   {\ccUnlessComp{Number}
466     {\ccIfPropVal{numbering}{auto}
467       {\ccIfAttr{\ccfCapType}{subfloat}
468         {\ifnum\ccSubFloatCnt=\z@\relax
469           \ccf@set@top@counter%
470         \else
471           \ccIfPropVal{sub-numbering}{auto}
472           {\ccf@set@subcounter}{\}%
473         \fi}
474     {\ccf@set@top@counter}}{\}}}
```

`\ccf@set@top@counter` generates first level float counter.

```

475 \def\ccf@set@top@counter{%
476   \ccWhenComp{Caption}{%
477     \global\expandafter\advance\csname c@\ccfCapType\endcsname\@ne\relax
478     \ccdefFromProperty\ccf@name@prefix{auto-number-prefix}%
479     \ccdefFromProperty\ccf@name@sep{auto-number-prefix-sep}%
480     \protected@edef\@tempa{\ccf@name@prefix\ccf@name@sep\expandafter\the\csname c@\ccfCapType\endcsname}%
481     \ccComponentEA{Number}{\@tempa}%
482   }%
483 }
```

`\ccf@set@subcounter` generates second level counters for numbered sub-floats. #1 is the sub-float counter.

```

484 \def\ccf@set@subcounter{%
```

`float-number` <any> the counter of a first-level float

```

485   \ccSetPropertyVal{float-number}{\csname cc@\cc@cur@cont @Number-0\endcsname}%
```

`sub-number` <any> the counter of a second-level float

```

486 \ccSetPropertyVal{sub-number}{%
487   \beginingroup
488   \expandonce{\ccUseProperty{sub-number-face}}}%
489   \relax\ccUseProperty{sub-number-before}}%
490   \csname @\ccUseProperty{sub-number-style}\endcsname{\the\ccSubFloatCnt}%
491   \ccUseProperty{sub-number-after}}%
492   \endgroup}%
493 \ccComponent{Number}{\ccUseProperty{sub-number-format}}}%
494 }
```

### Generation of L<sup>A</sup>T<sub>E</sub>X Labels

`\ccfCreateLabel` creates labels

```

495 \def\ccfCreateLabel{%
496   \ccIfComp{Number}
497   {\def\cc@fallback@anchor{%
498     \ccGobble
499     \ccdefFromComp\@currentlabel{Number}%
500     \ccdefFromComp\@currentlabelname{ListofCaption}}}%
501   \def\cc@labelname@comp{Caption}}
502   {\def\cc@fallback@anchor{\phantomsection}}}%
503   \expandafter\ccCreateLabel\expandafter{\ccfCapType}}
```

## 3.8 Processing the Float

### Sizes, Spacing and Margins

`\ccf@set@hsize` calculates the available maximum width for the float contents and captions according to the values of the `margin-right` and the `margin-left` properties.

```

504 \def\ccf@set@hsize{%
505   \expandafter\ccf@sub@sep\ccUseProperty{sub-float-sep}\relax%
506   \global\ccf@total@width=\hsize\relax
507   \expandafter\ccf@margin@l\ccUseProperty{margin-left}\relax
508   \expandafter\ccf@margin@r\ccUseProperty{margin-right}\relax
509   \expandafter\ccf@margin@i\ccUseProperty{margin-inner}\relax
510   \expandafter\ccf@margin@o\ccUseProperty{margin-outer}\relax
511   \ccf@set@margins
512   \global\advance\ccf@total@width-\ccf@margin@r\relax
513 }
```

`\ccf@set@margins` realises inner and outer margins via the left and right margins.

```

514 \def\ccf@set@margins{%
515   \ccTestPage
516   \if@cc@odd
517     \advance\ccf@margin@l\ccf@margin@i
518     \advance\ccf@margin@r\ccf@margin@o
519   \else
520     \advance\ccf@margin@l\ccf@margin@o
521     \advance\ccf@margin@r\ccf@margin@i
522   \fi
523 }
```

## Processing the Contents of the Float Environment

`\ccf@process` calculates the dimensions of the content of a float environment (including captions and spacing) and eventually prints the contents using the `float-render` and `subfloat-render` Properties.

```

524 \def\ccf@process{%
525   \ifx\ccf@has@capt@top\@empty\leavevmode\fi
526   \ccf@make@outer@caption{top}%
527   \ifnum\the\ccSubFloatCnt=\z@\relax
528     \bgroup\advance\hsize-\ccf@margin@l
529     \@cc@is@finaltrue
530     \ccUseProperty{float-render}%
531     \egroup
532   \else
533     \ccf@test@subcapt
534     \@cc@is@finalfalse
535     \ccf@calc@sameheight
536     \def\ccf@prefix{sub}%
537     \ifx\ccf@has@subcapt@top\@empty\ccf@calc@row@ht{top}\fi
538     \ifx\ccf@has@subcapt@bottom\@empty\ccf@calc@row@ht{bottom}\fi
539     \@cc@is@finaltrue
540     \ccUseProperty{subfloat-render}%
541     \let\ccf@prefix\@empty
542   \fi
543   \ccf@make@outer@caption{bottom}%
544 }

```

`\ccf@calc@row@ht` calculates the heights of all captions in the same row.

`{#1}` determines if the `top` or `bottom` row is calculated.

```

545 \def\ccf@calc@row@ht#1{%
546   \@tempcnta\z@
547   \@tempdima\z@
548   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
549     \setbox\z@\vbox{%
550       \ccSubFloatCnt\@tempcnta\relax
551       \expandafter\hsize\expandafter\dimexpr\csname cc@\cc@cur@cont @res@width-\the\@tempcnta\endcsname\relax
552       \ccGobble
553       \ccUseProperty{\ccf@prefix caption-face}%
554       \ccUseProperty{\ccf@prefix caption-face-#1}%
555       \leavevmode
556       \strut\ccUseProperty{caption-#1}\strut%
557     }%
558     \expandafter\ifdim\dimexpr\ht\z@+\dp\z@\relax>\@tempdima \@tempdima\dimexpr\ht\z@+\dp\z@\relax\fi
559   }%
560   \expandafter\edef\csname ccf@capt@row@height@#1\endcsname{\the\@tempdima}%
561 }

```

`\ccf@calc@sameheight` calculates the ratio between each sub-float's height and the height of the largest sub-float

```

562 \def\ccf@calc@sameheight{%
563   \if@ccf@sameheight
564     \@tempdima=\z@\relax
565     \@tempcnta=\z@\relax
566     \ccf@calc@width=\ccf@total@width\relax
567     \advance\ccf@calc@width-\ccf@margin@l\relax
568     \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%

```

```

569 \edef\@tempa{\CalcRatio{\ccf@sub@maxheight}{\csname ccf@cc@cur@cont @height-\the\@tempcnta
\endcsname}}}%
570 \ifnum\the\@tempcnta>\@ne\relax
571 \advance\ccf@calc@width-\ccf@sub@sep\relax%
572 \fi
573 \expandafter\@tempdimc\csname ccf@cc@cur@cont @width-\the\@tempcnta\endcsname\relax
574 \@tempdimb=\@tempa\@tempdimc\relax
575 \csedef{cc@cc@cur@cont @adj@width-\the\@tempcnta}{\the\@tempdimb}%
576 \advance\@tempdima\@tempdimb
577 }%
578 \@tempcnta=\z@\relax
579 \@tempdimb=\z@\relax
580 \@tempdimc=\z@\relax
581 \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
582 \edef\@tempa{\CalcRatio{\csname cc@cc@cur@cont @adj@width-\the\@tempcnta\endcsname}{\
@tempdima}}}%
583 \csedef{cc@cc@cur@cont @res@width-\the\@tempcnta}{\dimexpr\@tempa\ccf@calc@width\relax}%
584 \@tempdimc\dimexpr\csname ccf@cc@cur@cont @height-\the\@tempcnta\endcsname\relax
585 \@tempdimc\dimexpr\@tempa\@tempdimc\relax
586 \ifdim\@tempa\@tempdimb<\@tempdimc\@tempdimb\@tempdimc\relax\fi
587 }%
588 \else
589 \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
590 \csletcs{cc@cc@cur@cont @res@width-\the\@tempcnta}{ccf@cc@cur@cont @width-\the\@tempcnta}
591 }%
592 \fi
593 \csedef{cc@cc@cur@cont @res@height}{\the\@tempdimb}%
594 }

```

### 3.9 Caption mechanism

`\ccf@test@caption` tests if the current sub-float has any top or bottom caption that needs to be printed.

- #1 is the value of the sub-float counter
- #2 indicates if the caption belongs to the whole float (empty) or a sub-float (sub)
- #3 top or bottom

We compare the caption of the current `\SubCounter` level with a caption of a non-existing, negative, float level in case there is non-expandable material hard-coded into the `caption-#3` Property. If we were to compare the width of the `\hbox` with `\z@`, this scenario would give us false positives.

**Warning:** Long captions can cause the `\hbox`'s width to exceed `\maxdimen`. To avoid L<sup>A</sup>T<sub>E</sub>X errors in this case, we compare `sp` instead of `pt`. This, however, means that if the difference is less than 1pt, the test fails and no caption is printed!

```

595 \def\ccf@test@caption#1#2#3{%
596 \cc@is@finalfalse
597 \setbox\cc@tempboxa\hbox{\ccGobble\ccSubFloatCnt=0#1\relax\ccUseProperty{#2caption-#3}\relax}%
598 \setbox\cc@tempboxb\hbox{\ccGobble\ccSubFloatCnt\m@ne\relax\ccUseProperty{#2caption-#3}\relax}
%
599 \edef\my@wda{\expandafter\strip@pt\wd\cc@tempboxa sp}%
600 \edef\my@wdb{\expandafter\strip@pt\wd\cc@tempboxb sp}%
601 \ifdim\my@wda>\my@wdb\relax
602 \expandafter\global\expandafter\let\csname ccf@has@#2capt@#3\endcsname\@empty
603 \fi
604 \cc@is@finaltrue
605 }

```

`\ccf@test@subcapt` tests if the current float has any top or bottom captions that need to be printed

```

606 \def\ccf@test@subcapt{%
607   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
608     \ccf@test@caption{\the\@tempcnta}{sub}{top}%
609     \ccf@test@caption{\the\@tempcnta}{sub}{bottom}%
610   }%
611 }

```

`\ccf@capt@top@offset` determines the spacing inserted **above both captions**.

```

612 \def\ccf@capt@top@offset#1{%
613   \ccIfStrEqual{#1}{top}{}%
614   \par\ifccf@break@capt\else\nopagebreak\fi%
615   \expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-bottom}\relax%
616   \advance\@tempskipa\dimexpr-\topskip+\dp\strutbox\relax
617   \ifccf@break@capt\advance\@tempskipa\dimexpr-\baselineskip-\ht\strutbox+\topskip\relax\fi
618   \ifx\ccf@has@subcapt@bottom\empty
619     \ifnum\the\ccSubFloatCnt=\z@
620       %% subcapt-bot exists and capt-bot is rendered
621       \advance\@tempskipa\dimexpr\dp\strutbox\relax
622       \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-bottom}\relax%
623     \fi
624   \fi
625   \vskip\@tempskipa
626   \leavevmode
627 }

```

`\ccf@capt@bottom@offset` determines the spacing inserted **below the captions**.

```

628 \def\ccf@capt@bottom@offset#1{%
629   \ccIfStrEqual{#1}{top}%
630   {\@tempskipa=\z@\relax
631     \expandafter\advance\expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-top}%
632     \ifnum\the\ccSubFloatCnt=\z@\relax
633       \ifx\ccf@has@subcapt@top\empty
634         %% subcapt-top exists and capt-top is rendered
635         \advance\@tempskipa\dimexpr\ht\strutbox-\topskip-\p@\relax
636         \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-top}\relax%
637       \else
638         \advance\@tempskipa\dimexpr-\dp\strutbox\relax
639       \fi
640     \fi
641     \vskip\@tempskipa
642     \par\ifccf@break@capt\else\nopagebreak\fi}
643   {\ifnum\the\ccSubFloatCnt>\z@\relax
644     \vskip\dp\strutbox
645     \fi}}

```

`\ccf@make@caption` prints the caption.

- #1 is the placement (top, bottom)
- #2 is the vertical alignment (top, middle, bottom)

```

646 \long\def\ccf@make@caption#1#2{%
647   \ccf@capt@top@offset{#1}%
648   \ifnum\the\ccSubFloatCnt=\z@\relax
649     \def\ccf@caption@box{%

```



```

650 \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
651   {\setbox\@tempboxa\ vbox\ bgroup\ hsize\ textheight}
652   {\hskip\ccf@margin@1%
653    \setbox\@tempboxa\ vbox\ bgroup\ advance\ hsize-\ccf@margin@1}%
654   }%
655 \else
656   \expandafter\cc@tempskipa\ csname ccf@capt@row@height@#1\ endcsname\ relax
657   \expandafter\ advance\ expandafter\cc@tempskipa\ dimexpr-\ baselineskip+\ topskip\ relax
658   \def\ccf@caption@box{\setbox\@tempboxa\ vbox to \cc@tempskipa\ bgroup}%
659 \fi
660 \ccf@caption@box%
661 \ccIfStrEqual{#2}{top}{ }\ {\if@ccf@break@capt\ else\ vss\ fi}%
662 \ccUseProperty{\ccf@prefix caption-face}%
663 \ccUseProperty{\ccf@prefix caption-face-#1}%

```

The caption as a whole is tagged with `<Caption/>`.

```

664 \ccaStructStart{Caption}%
665 \cc@topstrut\ccUseProperty{\ccf@prefix caption-#1}\ strut%
666 \ccaStructEnd{Caption}%
667 \ifx\ccf@measure\ relax\ else
668   \ccIfPropVal{label-pos}{#1}{%
669     \ccfCreateLabel%
670     \ccf@write@listof%
671   }%
672 \fi
673 \ccIfStrEqual{#2}{bottom}{ }\ {\if@ccf@break@capt\ else\ vss\ fi}%
674 \egroup%
675 \if@ccf@break@capt\ unvbox\@tempboxa\ else\ box\@tempboxa\ fi%
676 \ccf@capt@bottom@offset{#1}%
677 }

```

`\ccf@make@outer@caption` is a shell for the outer captions. #1 is the placement (top or bottom)

```

678 \def\ccf@make@outer@caption#1{%

```

now, we print the actual captions, if they contain contents.

```

679 \expandafter\ifx\ csname ccf@has@capt@#1\ endcsname\@empty
680   \setbox\z@\ vbox{%
681     \cc@is@finalfalse
682     \let\ccf@measure\ relax
683     \ccGobble
684     \ccSubFloatCnt\z@
685     \ccf@make@caption{#1}{top}%
686   }%
687   \immediate\write\@auxout{\string\expandafter\string\gdef\string\ csname\space ccFloat\the\
688     ccf@int@cnt Cap#1\string\endcsname{\the\dimexpr \ht\z@+\dp\z@\relax}}%
689   \bgroup
690   \cc@is@finaltrue
691   \savenotes
692   \if@ccf@break@capt\ else\ nopagebreak\ fi
693   \ccSubFloatCnt\z@
694   \ccf@make@caption{#1}{top}%
695   \spewnotes
696   \egroup
697   \ccIfStrEqual{#1}{top}{ }\ {\if@ccf@break@capt\ else\ nopagebreak\ fi}{ }%
698 \fi}

```

`\ccf@make@subcaption` creates the caption for subfloats. #1 is the position (`top` or `bottom`).

```

698 \def\ccf@make@subcaption#1{%
699   \expandafter\ifx\csname cc@has@\ccf@prefix capt@#1\endcsname\@empty
700     \ccf@make@caption{#1}{\ccUseProperty{\ccf@prefix caption-valign-#1}}%
701   \fi}

```

## 4 Generic User-Level Float Containers

`\ccDeclareFloat` is a user-level macro used to declare a new `ccFloat` environment.

- [#1] Name of the float Container from which the declared Container should inherit Properties (*optional*)
- {#2} top-level name of the float environment (e.g., `\ccPrefix Table`, `\ccPrefix Figure`)
- {#3} caption type (e.g., `table`, `figure`)
- {#4} list (e.g., `lot`, `lof`)
- {#5} additional Component body, use this to add to Types or introduce custom Handlers to the Float Container.

```

702 \def\ccDeclareFloat{\ccopt@empty\ccf@declare@float}
703 \long\def\ccf@declare@float[#1]#2#3#4#5{%

```

First, we check if the Container already exists. If so, we issue an error message. May we force the style programmers learn to make use of CoCoT<sub>E</sub>X's extensive toolbox.

```

704   \ifcsdef{cc@container@#2}{%
705     \ccPackageError{Float}{}
706     {Attempt to re-define pre-existing float Container `#2'}
707     {You cannot re-define an existing float Container. Use
708   \string\ccAddToType{<Type>}{#2}{<code>} to alter the #2 container!}}{%

```

Otherwise, we declare the new Container and invoke all the Initializers.

```

709   \def\ccf@parent{#1}%
710   \ccDeclareContainer{#2}{%
711     \ccPackageInfo{Floats}{}{Declaring float `#2'}%
712     \ifx\ccf@parent\@empty
713       \ccInherit{Properties,Components,Attributes}{float}
714     \else
715       \ccInherit{Properties,Components,Attributes}{\ccf@parent}
716     \fi
717     \ccDeclareType{FloatEnvInfo}{%
718       \ccSetContainer{#2}%
719       \def\ccf@capType{#3}%
720       \def\ccf@cap@list@type{#4}%
721     }% /FloatEnvInfo

```

The macro actually defines two L<sup>A</sup>T<sub>E</sub>X environments; a normal one for one-column floats, and a starred one for page-wide floats in two-column mode.

```

722   \ccDeclareEnv[#2]{\ccf@float}{\endccf@float}%
723   \ccDeclareEnv[#2*]{\if@twocolumn\let\ccf@do@dbl\relax\else\fi\ccf@float}{\if@twocolumn\let\
724     ccf@do@dbl\relax\fi\endccf@float}%
725   \ccDeclareType{Components}{}%
726   \ccDeclareType{Properties}{}%

```

Generating the Handlers for the list-of entries and define the corresponding `l@` macros

```

726   \ccf@generate@listof@handlers{#4}{#3}{#2}%

```

```

727 \bgroup
728 \def\cc@cur@cont{#2}%
729 \cc@init@l@[list-of]{#3}{0}{#3}% Generate listof-Entries for first level floats
730 \cc@init@l@[list-of]{#4}{1}{sub#3}% Generate listof-Entries for sub-floats
731 \egroup
732 #5
733 }% /container
734 }

```

## 5 Image Containers

### 5.1 Abstract Graphics Container

**Graphic** is an abstract Container that represents an image file.

```

735 \ccDeclareContainer{Graphic}{%
736 \ccDeclareType{Components}{%
737 \def\cc@counted@comp@scheme#1{#1-\the\ccSubFloatCnt}%

```

**Fig** holds the `includegraphics` with the path to and the options for the actual image file.

```

738 \ccfMakeComp{Fig}%

```

**AltText** is the alternative text for accessibility.

```

739 \ccfMakeComp{AltText}%
740 }%
741 \ccDeclareType{Properties}{}%
742 }

```

### 5.2 Floating Figure Container

**Figure** is the user-level Container for display-style images or image clusters including their respective captions. Figures may either be placed as free-standing in-situ blocks or as floats.

```

743 \ccDeclareFloat{Figure}{figure}{lof}{%
744 \ccInherit{Properties,Components}{Graphic}%
745 \ccDeclareType{Properties}{%

```

**subfloat-same-height** [true|false] Whether all images in subfloats could be scaled to the same height (true) or not (false).

```

746 \ccSetProperty{subfloat-same-height}{true}%

```

**subfloat-content** <any>

```

747 \ccSetProperty{subfloat-content}{%
748 \ifx\ccf@no@figs\relax
749 \rule{0pt}{1pt}\rule{1pt}{0pt}%
750 \else
751 \ccUseComp{Fig}%
752 \fi}%

```

`float-render` <any> figure specific output routine.

```
753 \ccSetProperty{float-render}{\ccfFigureRender}%
```

`subfloat-render` <any> figure specific output routine for sub-floats.

```
754 \ccSetProperty{subfloat-render}{\ccfSubFigureRender}%
755 }%
756 }
```

### 5.3 Figure Output Routines

`\ccfFigureRender` tells the float Container how the main content Component if Figure-type Floats is to be rendered. It is called via the Property.

```
757 \def\ccfFigureRender{%
758   \bgroup
759   \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
760   {\hsize\dimexpr\textwidth-\ccf@margin@r-\ccf@margin@l\relax}%
761   }%
762   \let\includegraphics\ccf@includesubgraphics
763   \hskip\ccf@margin@l
764   \strut\ccUseComp{Fig}\strut
765   \egroup}
```

`\ccfSubFigureRender` tells the abstract float Container how the main content Component of Figure-type sub-floats are to be rendered. It is called via the Property.

```
766 \def\ccfSubFigureRender{%
767   \hskip\ccf@margin@l
768   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
769     \ccfRenderSubFloats{\the\@tempcnta}{Fig}%
770   }}
```

`\ccf@includesubgraphics` is an override of L<sup>A</sup>T<sub>E</sub>X's `\includegraphics` patched to adjust for maximum width and height, and to capture the `alt` key in order to pass it down to `ltpdfa`.

In order to capture multiple images in the same Figure Container (i.e. real or fake Sub-Floats), tagging as `<Figure/>` of images takes place here, as does assignment of alternative text(s). Those can be submitted by the `AltText` Component or by the `alt` key in the optional argument of `\includegraphics`. If both are given, the `alt` key takes precedence. If neither is given, a `--` is inserted.

```
771 \def\ccf@includesubgraphics{\cc@opt@empty\ccf@includesubgraphics}%
772 \def\@ccf@includesubgraphics[#1]#2{%
773   \def\@igopts{max width=\hsize,max height=\vsize,width=\hsize}%
774   \if!#1!\else
775     \apptocmd\@igopts{,#1}{-}{-}%
776   \fi
777   \gdef\ccf@fig@path{#2}%
778   \ifcc@is@final
779     \ccaStructStart{Figure}%
780     \ccaAddPlacement{Block}%
781   \fi%
782   \expandafter\ccf@ltx@includegraphics\expandafter[\@igopts]{#2}%
783   \ifcc@is@final
784     \ifx\relax\cca@Gin@alt\relax
785       \ccIfComp{AltText}
786       {\ccaAddAltText{\ccUseComp{AltText}}}
```

```

787     {\ccaAddAltText{--}}%
788   \else
789     \ccaAddAltText{\cca@Gin@alt}%
790   \fi
791   \ccaStructEnd{Figure}%
792 \fi
793 }

```

`\ccf@measuresubgraphics` is an override of L<sup>A</sup>T<sub>E</sub>X's `\includegraphics` that is used to measure the natural dimensions of the included image. It also checks if the `\includegraphics` has either an height or width explicitly given. if so, we de-activate the same-height calculations for the entire float.

```

794 \def\ccf@measuresubgraphics{\cc@opt@empty\@ccf@measuresubgraphics}
795 \def\@ccf@measuresubgraphics[#1]#2{%
796   \begingroup
797     \setkeys{Gin}{#1}%
798     \ifx\Gin@ewidth\Gin@exclamation
799       \ifx\Gin@eheight\Gin@exclamation\else
800         \global\@ccf@sameheightfalse
801       \fi
802     \else
803       \global\@ccf@sameheightfalse
804     \fi
805   \endgroup
806   \ccf@ltx@includegraphics[#1]{#2}%
807 }

```

## 5.4 Inline Figures

### Inline Figure Container

`InlineFigure` is the user-level Container for inline graphics (e. g., images in tables or symbols inside the main text body). Note that this Container is *not* derived from the abstract `float` Container. Also, there is no L<sup>A</sup>T<sub>E</sub>X environment for that Container but a simple macro.

```

808 \ccDeclareContainer{InlineFigure}{%
809   \ccInherit{Properties,Components}{Graphic}%
810   \ccDeclareType{Attributes}{}%
811   \ccDeclareType{Properties}{%

```

`smash` [true|false] whether the image is allowed to stretch the line it is in (false) or not (true) if the height exceeds `\baselineskip`.

```

812   \ccSetProperty{smash}{false}

```

`vertical-align` [top|middle|bottom] the vertical alignment of the inline image relative to the baseline of the surrounding text. If the value is `bottom`, the bottom border of the image is aligned with the baseline, `top` aligns the top border of the image at baseline + `\ht\strutbox`, `middle` centers the image at baseline +  $0.5 \times \text{\ht\strutbox}$ .

```

813   \ccSetProperty{vertical-align}{bottom}

```

`float-render` <any> specific output routine for inline figures

```

814   \ccSetProperty{float-render}{\ccUseComp{Fig}}
815 }%
816 }

```

## Inline Figure User Macro

`\ccInlineFigure` is the Handler for an inline figure's main content Component.

`[#1]` is the attribute list for the figure

`{#2}` is the Container Body

```

817 \def\ccInlineFigure{\cc@opt@empty\cc@inline@figure}
818 \def\cc@inline@figure[#1]#2{%
819   \begingroup
820     \ccSetContainer{InlineFigure}%
821     \def\ccfCapType{figure}%
822     \ccToggleCountedConditionals
823     \ccEvalType{Properties}%
824     \ccEvalAttributes{#1}%
825     \ccf@eval@class
826     \ccEvalType{Components}%
827     \ignorespaces
828     #2%
829     \ccSubFloatCnt=\z@\relax
830     \bgroup
831       \ccUseProperty{float-render}%
832       \egroup
833       \ccf@debug%
834       \ccf@store@dimens
835     \endgroup
836 }
837 \csdef{\ccPrefix InlineFigure}{\ccInlineFigure}%

```

## 6 Table Containers

### 6.1 The Abstract Tabular Container

CoCoT<sub>E</sub>X's float module supports the three basic Standard L<sup>A</sup>T<sub>E</sub>X tabular environments (*tabular*, *tabularx* and *tabulary*) as well as *htmltab* from the *htmltabs* package. For the measuring to work correctly, we need to render the tables as a whole and store the result inside `\ccf@floatbox` for measuring and further processing.

*Tabular* is an abstract Container that represents raw table data. Its main purpose is to provide a unified interface to patch some of L<sup>A</sup>T<sub>E</sub>X's standard *tabular* environments, as well as the *htmltab* environment, if the *htmltabs* package is loaded.

```

838 \ccDeclareContainer{Tabular}{%
839   \ccDeclareType{Properties}{}%
840   \ccDeclareType{Components}{%
841     \ccf@reserve@tabular
842   }%
843 }

```

`\ccf@reserve@tabular` is a shell macro that temporarily stores the default macro definitions for various tabular environments and patches them such that the contents are stored inside the `\ccf@floatbox`. The macro is called at the very beginning of the Table Container's environment and the patches only hold inside that environment. Thus, all tabular environments can be used in their vanilla state outside CoCoT<sub>E</sub>X's Table environments.

```

844 \def\ccf@reserve@tabular{%
845   \ccf@reserve@tab{}%
846   \ccf@reserve@tab{x}%

```

```

847 \ccf@reserve@tab{y}%
848 \ccf@reserve@htmltab%
849 }

```

`\ccf@reserve@tab` stores the default definitions for a specific vanilla-L<sup>A</sup>T<sub>E</sub>X tabular environment and re-defines the macros in a way that the tabulars are stored in the `\ccf@floatbox` instead of printed onto the page.

```

850 \def\ccf@reserve@tab#1{%
851   \csletcs{orig@tabular#1}{\tabular#1}%
852   \csletcs{orig@endtabular#1}{\endtabular#1}%
853   \csdef{\tabular#1}{%
854     \global\setbox\ccf@floatbox
855     \vbox\bgroup
856       \if!#1!\else
857         \let\tabular\orig@tabular
858         \let\endtabular\orig@endtabular
859       \fi
860       \csname orig@tabular#1\endcsname}%
861   \csdef{\endtabular#1}{\csname orig@endtabular#1\endcsname\egroup}%
862 }

```

## 6.2 The User-Level Table Container

**Table** is a user-level Container for display-style tables including their captions. They may wither be places as free-standing in-situ blocks or as floats.

```

863 \ccDeclareFloat{Table}{table}{lot}{%
864   \ccInherit{Properties,Components}{Tabular}%
865   \ccDeclareType{Properties}{%
866     \ccSetProperty{subcaption-valign-top}{bottom}%
867     \ccSetProperty{subfloat-content}{%
868       \PackageError{coco-floats.sty}
869       {ccSubFloat does not support sub-tables (yet)!}
870       {You cannot yet use a tables within the `ccSubFloat'!}%
871     }%
872     \ccSetProperty{float-render}{\ccfTableRender}%
873     \ccSetProperty{subfloat-render}{\ccfSubTableRender}%
874   }%
875 }

```

`\ccf@reserve@htmltab` special handler for tables using the `htmltabs` package:

```

876 \AtBeginDocument{%
877   \@ifpackageloaded{htmltabs}{%
878     \def\ccf@reserve@htmltab{%
879       \let\ccf@add@style\@empty
880       \ifx\ccf@floatpos\@empty
881         \expandafter\ifx\csname \ccPrefix Float\the\ccf@int@cnt Captop\endcsname\relax\else
882           \htInitSkip\csname \ccPrefix Float\the\ccf@int@cnt Captop\endcsname
883           \advance\htInitSkip\ccf@sep@top%
884         \fi
885         \expandafter\ifx\csname \ccPrefix Float\the\ccf@int@cnt Capbottom\endcsname\relax\else
886           \htAddToBottom\csname \ccPrefix Float\the\ccf@int@cnt Capbottom\endcsname
887           \advance\htAddToBottom\ccf@sep@bottom%
888         \fi
889       \else
890         \def\ccf@add@style{;break-table:false;}%
891       \fi
892     }
893   }
894 }

```

```

892 \edef\cc@tempa{margin-left:\ccf@margin@l\ccf@add@style}%
893 \expandafter\htAddStyle\expandafter{\cc@tempa}%
894 \global\setbox\htTableBox\box\voidb@x
895 \let\htOutputTable\relax
896 }{\let\ccf@reserve@htmltab\relax}%
897 }

```

### 6.3 The Table Output Handler

`\ccfGetTableContent` returns the `\ccf@floatbox` if it is not un-itialized or void.

```

898 \def\ccfGetTableContent{%
899 \ifx\htTableBox\@undefined\else
900 \ifvoid\htTableBox\else
901 \let\ccf@floatbox\htTableBox%
902 \fi\fi}

```

`\ccfTableRender` is the content of the Property specific for tables.

```

903 \def\ccfTableRender{%
904 \ccfGetTableContent
905 \ccComponent{Content}{\unvbox\ccf@floatbox}%
906 \ccUseComp{Content}%
907 \ccaStructStart{Table}%
908 \ifx\ht@structID@Thead\@undefined\else\ccaMoveStruct{\ht@structID@Thead}\fi%
909 \ifx\ht@structID@TBody\@undefined\else\ccaMoveStruct{\ht@structID@TBody}\fi%
910 \ifx\ht@structID@TFoot\@undefined\else\ccaMoveStruct{\ht@structID@TFoot}\fi%
911 \par\if\ccf@break@capt\else\nopagebreak\fi
912 \vskip\dp\strutbox
913 \ccaStructEnd{Table}%
914 }

```

`\ccfSubTableRender` Is the content of the table-specific Property

**Note** that table sub-floats aren't allowed yet, so this definition is un-used at the moment. TeX will crash with an error message before this Property is ever expanded.

```

915 \def\ccfSubTableRender{%
916 \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
917 \ccfGetTableContent
918 \@cc@is@finalfalse
919 \ccComponent{Content}{\unvbox\ccf@floatbox}%
920 \@cc@is@finaltrue
921 \ccfRenderSubFloats{\the\@tempcnta}{Content}%
922 }}

```

## 7 Other Float-Related Macros

`\ccFloatBarrier` can be used to force all pending floats to be printed at the next shipout.

```

923 \def\ccFloatBarrier{\AtBeginShipoutNext{\clearpage}}

```

Output Driver for the `coco-floats.sty`.



```
</floats>
```



## Module 11

# coco-frame.dtx

This file provides facilities to visualise crop marks and the print area.

```

22 <*frame>

23 %%
24 %% module for CoCoTeX for crop marks and print area frames.
25 %%
26 %% Maintainer: p.schulz@le-tex.de
27 %%
28 %% lualatex - texlive > 2019
29 %%
30 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
31 \ProvidesPackage{coco-frame}
32   [2024/07/16 0.5.0 coco-frame]\relax

```

## 1 Top-Level Interface

```

33 \let\cc@frame@mode n
34 \define@choicekey{coco-frame.sty}{frame}[\cc@frame@mode\nr]{none,crop,frame}{%
35   \ifcase\nr\relax% none
36     \let\cc@frame@mode n
37   \or% crop
38     \let\cc@frame@mode p
39   \else% frame
40     \let\cc@frame@mode w
41   \fi
42 }%
43 \ProcessOptionsX\relax

```

## 2 Cropmark printer

```

44 \ifx\cc@frame@mode p\relax
45   \ifx\bleed\@undefined \newdimen\bleed \bleed4mm\relax\fi
46   \ifx\cc@frame@@offset\@undefined \newdimen\cc@frame@@offset \cc@frame@@offset4em\relax\fi
47   \voffset\dimexpr\cc@frame@@offset-1in\relax
48   \hoffset\dimexpr\cc@frame@@offset-1in\relax
49   \edef\l@offset{\strip@pt\dimexpr\cc@frame@@offset*7200/7227\relax}
50   \edef\r@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperwidth)*7200/7227\relax}
51   \edef\u@offset{\strip@pt\dimexpr(\cc@frame@@offset)*7200/7227\relax}
52   \edef\o@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperheight)*7200/7227\relax}
53   \edef\b@l@offset{\strip@pt\dimexpr(\cc@frame@@offset-\bleed)*7200/7227\relax}

```

```

54 \edef\b@r@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperwidth+\bleed)*7200/7227\relax}
55 \edef\b@u@offset{\strip@pt\dimexpr(\cc@frame@offset-\bleed)*7200/7227\relax}
56 \edef\b@o@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperheight+\bleed)*7200/7227\relax}
57 \edef\@tempa{%
58   /TrimBox [\l@offset\space\u@offset\space\r@offset\space\o@offset]
59   /BleedBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
60   %/CropBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
61   %/MediaBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
62 }
63 \expandafter\pdfpageattr\expandafter{\@tempa}
64 \fi

```

Apparently, the crop package relies on old pdf dimension macros. If they aren't defined, we load the `luatex85` package and set the values of the type area by hand:

```

65 \@ifundefined{pdfpagewidth}{%
66   \RequirePackage{luatex85}
67   \pdfpagewidth\paperwidth
68   \pdfpageheight\paperheight
69 }{}

```

Setting PDF boundaries

```

70 \ifx\cc@frame@mode n\relax\else
71   \ifx\cc@frame@mode p\relax
72     \edef\stockwidth{\the\dimexpr\paperwidth+\cc@frame@offset+\cc@frame@offset\relax}
73     \edef\stockheight{\the\dimexpr\paperheight+\cc@frame@offset+\cc@frame@offset\relax}
74   \fi

```

Cropmarks and page area frames both are painted via the `crop` package.

```

75 \RequirePackage{crop}
76 \renewcommand*\CROP@marks{%
77   \CROP@setmarkcolor
78   \CROP@user@b
79   \vskip1in\hskip1in\relax
80   \CROP@ulc\null\hfill\CROP@@@info\CROP@upedge\hfill\null\CROP@urc\hskip-1in\null
81   \vfill
82   \CROP@ledge\hfill\CROP@redge
83   \vfill
84   \hskip1in\relax
85   \CROP@llc\null\hfill\CROP@loedge\hfill\null\CROP@lrc\hskip-1in\null
86   \vskip-1in}%
87 \ifx\cc@frame@mode p\relax
88   \def\camcross{%
89     \smash{\rlap{%
90       \kern-0.15\p@
91       \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
92       \kern-0.15\p@
93       \kern-1.7mm\relax
94       \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
95       \kern-0.3\p@
96       \raise1.7mm\rlap{\vrule\@width3.4mm\@height\z@\@depth0.3\p@}%
97       \lower1.7mm\rlap{\vrule\@width3.4mm\@height0.3\p@\@depth\z@}%
98       \hbox{\vrule\@width3.4mm\@height0.15\p@\@depth0.15\p@}%
99       \kern-0.3\p@
100      \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax}}}
101   \def\cammcrossleft{%
102     \llap{\camcross\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\kern\bleed}}
103   \def\cammcrossright{%

```

```

104 \rlap{\kern\bleed\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\@
    camcross}}
105 \def\cammcrossup{%
106 \rlap{\smash{\raise\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
107 \kern-0.15\p@\vrule\@width0.3\p@\@height\dimexpr\cc@frame@@offset-2mm\relax\@depth-\@
    bleed}}}}
108 \def\cammcrossdown{%
109 \rlap{\smash{\lower\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
110 \kern-0.15\p@\vrule\@width0.3\p@\@height-\@bleed\@depth\dimexpr\cc@frame@@offset-2mm\@
    relax}}}}
111 \def\CROP@@ulc{\cammcrossup\cammcrossleft}
112 \def\CROP@@urc{\cammcrossup\cammcrossright}
113 \def\CROP@@llc{\cammcrossdown\cammcrossleft}
114 \def\CROP@@lrc{\cammcrossdown\cammcrossright}
115 \renewcommand*\CROP@@info{%
116 \global\advance\CROP@index\@ne
117 \def\x{\discretionary}{\hbox{\kern.5em---\kern.5em}}}%
118 \ifx\CROP@pagecolor\@empty
119 \else
120 \advance\dimen@\CROP@overlap
121 \fi
122 \hb@xt@\z@{%
123 \hss
124 \lower1em\ vbox to\z@\vss
125 \centering
126 \hsize\dimexpr\paperwidth-20\p@\relax
127 \normalfont
128 \large
129 \vskip5mm\relax
130 \addvspace{\bleed}}%
131 \hss}%
132 }%
133 \crop[cam]

```

the code for the page area frame

```

134 \else% w
135 \@tempdima\dimexpr\textheight\relax
136 \divide\@tempdima by\baselineskip
137 \multiply\@tempdima by65536\relax
138 \edef\cnt@baselines{\strip@pt\@tempdima}%
139 \def\cc@frame@lines{%
140 \@tempcnta\z@
141 \loop\advance\@tempcnta\@ne
142 \hsize1em\relax
143 \ifodd\count\z@
144 \vrule\@width1em\@height0.2\p@\@depth0.02\p@
145 \llap{\smash{\the\@tempcnta\,}}%
146 \fi%
147 \rlap{%
148 \ifodd\count\z@\else\fi
149 \vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
150 \if@twocolumn
151 \kern\columnsep\vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
152 \fi
153 \ifodd\count\z@\else
154 \vrule\@width1em\@height0.00005\p@\@depth0\p@%
155 \llap{\smash{\the\@tempcnta\,}}%
156 \fi
157 }%
158 \break

```

```

159 \ifnum\@tempcnta<\cnt@baselines
160 \repeat}
161 \def\cc@frame@margin{%
162 \vrule height\textheight%
163 \hskip-\marginparwidth\relax
164 \vbox to\textheight{\hsize\marginparwidth\relax
165 \rlap{\vbox to\z@{\hrule width\marginparwidth}}}%
166 \null\vss
167 \rlap{\vbox to\z@{\hrule width\marginparwidth}}}%
168 }%
169 \vrule height\textheight%
170 }
171 \renewcommand*\CROP@frame{%
172 \vskip0in%
173 \color[cmym]{0.4,0,0,0}%
174 \ifodd\count\z@\let\@themargin\oddsidemargin\else\let\@themargin\evensidemargin\fi
175 \advance\@themargin1in
176 \moveright\@themargin
177 \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@
178 \vskip\topmargin\vbox to\z@{\vss\hrule width\textwidth}%
179 \vskip\headheight\vbox to\z@{\vss\hrule width\textwidth}%
180 \vskip\headsep\vbox to\z@{\vss\hrule width\textwidth}%
181 \hbox to\textwidth{%
182 \ifodd\count\z@
183 \rlap{\hskip\dimexpr\textwidth+\marginparsep+\marginparwidth\relax\cc@frame@margin}%
184 \else
185 \rlap{\hskip-\marginparsep\relax\cc@frame@margin}%
186 \fi
187 \llap{\vbox to\textheight{\tiny\let\@tempa\fontsize\normalsize\let\fontsize\@tempa\
188 selectfont
189 \vskip\topskip\cc@frame@lines\null\vss}}}%
189 \llap{\vrule height\textheight}%
190 \if@twocolumn
191 \hskip\columnwidth\rlap{\vrule height\textheight}%
192 \hskip\columnsep\rlap{\vrule height\textheight}%
193 \fi
194 \hfil\vrule height\textheight
195 }%
196 \vbox to\z@{\vss\hrule width\textwidth}%
197 \vskip\footskip\vbox to\z@{\vss\hrule width\textwidth}%
198 \vss}%
199 \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@%
200 \vskip-0in\rlap{\hskip1in%
201 \vbox to\z@{\vbox to\z@{\vss\hrule width\paperwidth}%
202 \hbox to \paperwidth{\llap{\vrule height\paperheight}\hfil%
203 \vrule height\paperheight}%
204 \vbox to\z@{\vss\hrule width\paperwidth}%
205 \vss}}\vss}}
206 \crop[frame,noinfo]%
207 \fi
208 \fi

```

```

209 </frame>

```

## Module 12

# coco-lists.dtx

```
<*lists>
```

This module provides handlers for list-like environments like item lists, enumerations, glossaries and descriptions.

**Note:** The `coco-lists` module diverges somewhat from the other CoCoTeX modules insofar as that its main Container does not follow the CoCoTeX's usual “collect all–process later” approach, but all Properties are processed at the beginning of each Container's instances and the contents are processed as they are parsed by the `\LaTeX` interpreter, just like “reguar”  $\LaTeX$  lists. Configuration of lists, however, follows the CoCoTeX playbook.

## 1 Preamble

```
23 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
24 \ProvidesPackage{coco-lists}
25   [2024/07/16 0.5.0 CoCoTeX lists module]
26 \RequirePackage{coco-common}
```

### 1.1 Package Options

If the `replace` option is set, LaTeX's default lists are replaced by `coco-lists` module. This effects  $\LaTeX$ 's `enumerate`, `itemize`, and `description` environments.

```
27 \newif\if@ccl@replace \@ccl@replacefalse
28 \DeclareOptionX{replace}{\global\@ccl@replacetrue}%
```

The option `inherit` defines how nested lists inherit their properties. Currently, there are two ways: `common`: All nested lists of the same type inherit only from the same, generic type definition; `conseq`: nested lists of the same type inherit from the next-higher level list of the same type, and from the generic type definition.

For example, if `inherit=common`, 3rd level `itemize` and 2nd level `itemize` both inherit only the property values of the same generic `itemize` list type. If `inherit=conseq`, 3rd level inherits the property lists from 2nd level `itemize`.

Since inheritance is a transitive relation, 3rd level `itemize` will ultimately also inherit the Properties from generic `itemize`, but in contrast to `common`, `conseq` allows 2nd level `itemize` to override some Properties of generic `itemize`, which will be propagate down to 3rd level `itemize`, while with `inherit=common`, the override on 2nd level `itemize` would have no effect on 3rd level `itemize`.

`\ccl@ih@common` is used for comparisons. It represents the `inherit=common` package option.

```
29 \def\ccl@ih@common{common}
```

`\ccl@ih@conseq` is used for comparisons. It represents the `inherit=conseq` package option.

```
30 \def\ccl@ih@conseq{conseq}%
```

`\ccl@str@local` is a string for comparison. It represents the `nesting=local` option.

```
31 \def\ccl@str@local{local}%
```

`\ccl@str@global` is a string for comparison. It represents the `nesting=global` option.

```
32 \def\ccl@str@global{global}%
```

`\ccl@inherit` stores the value of the `inherit` package option.

```
33 \let\ccl@inherit\ccl@ih@common
34 \define@choicekey{coco-lists.sty}{inherit}[\@ccl@inherit\nr]{conseq,common}{%
35   \ifcase\nr\relax% conseq: nested lists of the same type inherit only from the previous level
36   \global\let\ccl@inherit\ccl@ih@conseq
37   \fi
38 }
```

`\ccl@nesting` The nesting option sets whether the nesting level of a list should be counted list-specific (value `local`), or globally (value `global`, default).

```
39 \let\ccl@nesting\ccl@str@global
40 \define@choicekey{coco-lists.sty}{nesting}[\@ccl@nesting\nr]{local,global}{%
41   \ifcase\nr\relax% local
42   \global\let\ccl@nesting\ccl@str@local
43   \fi
44 }
```

```
45 \ProcessOptionsX
```

## 2 The List Container

`List` is the most abstract Container for lists.

```
46 \ccDeclareContainer{List}{%
```

### 2.1 List Properties

```
47 \ccDeclareType{Properties}{%
```

#### List Boundaries



`before-list` <any> is expanded at the very beginning of a (nested) list.

```
48 \ccSetProperty{before-list}{% at the very beginning of each (nested) list
49   \if@noskipsec \leavevmode \fi
50   \ifvmode\else
51   \unskip \par
52   \fi
```

🔹 `<L>` is the opening List tag

```
53 \ccaStructStart{L}%
54 }
```



`after-list` <any> is expanded at the very end of a (nested) list. By default, it calls the  `after-item` Property.  `</L>` is the closing List tag

```
55 \ccSetProperty{after-list}{%
56   \ccUseProperty{after-item}%
57   \ccaStructEnd{L}% end tag for the (nested) list
58 }%
```


## List Margins

`margin-top` <skip> is the vertical skip at the beginning of each List instance.

```
59 %% list margins
60 \ccSetProperty{margin-top}{\z@}%
```

`margin-bottom` <skip> is the vertical skip at the end of each List instance.

```
61 \ccSetProperty{margin-bottom}{\z@}% vertical space before the list.
```

`margin-left` [auto|<skip>] is the horizontal space to the left of each list instance, from left boundary of the page area. `auto` means that the left margin is set to the width of widest label +  `prev-margin-left`. The value is passed through `\dimexpr`, so basic arithmetic is allowed.

```
62 \ccSetProperty{margin-left}{\csname leftmargin\@roman\cclCurDepth\endcsname-\ccUseProperty{
   label-sep}+\ccUseProperty{prev-margin-left}}%
```


`margin-left` <dimen> is the maximum space reserved for a list item's label.

```
63 \ccSetProperty{max-label-width}{.33\textwidth}%
```

`margin-right` <skip> is the right margin of the list instance.

```
64 \ccSetProperty{margin-right}{\z@}% horizontal space to the right of each list item
```


## Between List Items

`item-sep` <skip> is the vertical space between two adjacent list items. Note that the real value value is advanced by the value of the  `par-skip` Property.


```
65 \ccSetProperty{item-sep}{\z@}%
```

`after-indent` [true|false] determines whether the text paragraph after the (top-level) list is indented (`true`) or not (`false`).

```
66 \ccSetProperty{after-indent}{false}%
```

`at-begin-item-body` <any> is expanded right at the beginning of a new item body and sets the  `<LBody>` tag.

```
67 \ccSetProperty{at-begin-item-body}{\ccaVstructStart{LBody}}%
```

`at-end-item-body` <any> is expanded at the very end of an item body, but before the final `\par`. By default, it only sets the closing  `</LBody>` tag.

```
68 \ccSetProperty{at-end-item-body}{\ccaVstructEnd{LBody}}%
```

`after-item` <any> is expanded after each list item. It calls the `at-end-item-body` Property and closes the item's final paragraph as well as the `</LI>` tag.

```

69 \ccSetProperty{after-item}{%
70   \ccUseProperty{at-end-item-body}%
71   \ccaVstructEnd{LI}% Close list item tags
72   \par}%

```

`before-item` <any> is called at the very beginning of each list item. If the current item is the first item, the `\ifcclFirst` conditional is set to false. All non-first items of the same List instance call the `after-item` Property and add a vertical skip of `item-sep` amount.

After that, the paragraph formatting parameters for the list-item `par-indent`, `par-skip`, and `par-fill-skip`, as well as the starting `<LI>` tag are set.

```

73 \ccSetProperty{before-item}{%
74   \ifcclFirst
75     \global\cclFirstfalse
76   \else
77     \ccUseProperty{after-item}%
78     \vskip\ccUseProperty{item-sep}%
79   \fi
80   \parindent\ccUseProperty{par-indent}\relax%
81   \parskip\ccUseProperty{par-skip}\relax%
82   \parfillskip\ccUseProperty{par-fill-skip}\relax%
83   \noindent
84   \leavevmode
85   \ccaVstructStart{LI}% Start tag for a list item
86 }%

```

`item-offset` <any> calculates `\cclItemIndent` from the `indent` and `label-sep` Properties and sets the horizontal offset of the first line of the list item. After that, the value of the macro is unsigned.

```

87 \ccSetProperty{item-offset}{%
88   \cclItemIndent\ccUseProperty{indent}%
89   \advance\cclItemIndent\dimexpr-\ccUseProperty{label-sep}\relax
90   \hskip\cclItemIndent\relax%
91   \ifdim\cclItemIndent>\z@
92     \cclItemIndent\ccUseProperty{indent}%
93   \else
94     \cclItemIndent-\ccUseProperty{indent}%
95   \fi
96 }%

```

`par-indent` <skip> is the indent of the first line of a \*new\* paragraph inside a list item

```

97 \ccSetProperty{par-indent}{\parindent}%

```

`par-fill-skip` <skip> is the skip at the end of the last line of each paragraph inside a list item

```

98 \ccSetProperty{par-fill-skip}{\@flushglue}%

```

`par-skip` <dimen> vertical space between two adjacent paragraphs inside the same List item

```

99 \ccSetProperty{par-skip}{\z@}%

```


## Label Formatting

`label` <any> prints the `Label` component.

```

100 \ccSetProperty{label}{\ccUseComp{Label}}%

```

**indent** [auto|auto-global|<dimen>] is the indent of each List item's first line (relative to margin-left).

If the value is **auto**, the real indent and left margin of a item's first line is calculated using coco-common's indentation mechanism (see Sect. 3.3 in Module Module 3). The first-line indent will thereby be calculated from the widest width of all labels of the same list type and nesting level.

**Note:** the value **auto-global** is allowed, but it causes \*all\* lists – regardless of the nesting level – to have the same left margin and indent!

```
101 \ccSetProperty{indent}{-\dimexpr\cename leftmargin\@roman\cclCurDepth\endcename-\ccUseProperty{label-sep}\relax}%
```

**label-sep** <dimen> is the horizontal space between the label and the item body.





```
102 \ccSetProperty{label-sep}{.5em}%
```

**label-face** <any> is the style of the label.




```
103 \ccSetProperty{label-face}{}%
```

**label-align** [left|center|right] is the alignment of the label within its local **\hbox**.





```
104 \ccSetProperty{label-align}{left}%
```

**label-format** <any> is the format of the label. It should call the label-face and label properties and enclose the latter with  <Lb1> and  </Lb1>.

```
105 \ccSetProperty{label-format}{%
106 \ccUseProperty{label-face}%
107 \ccaVstructStart{Lb1}%
108 \ccUseProperty{label}%
109 \ccaVstructEnd{Lb1}%
110 }%
```

**label-box** <any> is the property that builds a local **\hbox** into which the Label Component is printed. It should respect the label-align Property and call label-format.

```
111 \ccSetProperty{label-box}{%
112 \hbox to \cclItemIndent{%
113 \ccIfPropVal{label-align}{left}{-}{\hss}%
114 \ccUseProperty{label-format}%
115 \ccIfPropVal{label-align}{right}{-}{\hss}}%
116 }%
```

**item-format** <any> contains material printed at the beginning of a new item. It should call the before-item, item-offset, label-box and label-sep Properties.

```
117 \ccSetProperty{item-format}{%
118 \ccUseProperty{before-item}%
119 \ccUseProperty{item-offset}%
120 \ccUseProperty{label-box}%
121 \hskip\ccUseProperty{label-sep}%
122 }%
123 }%
```

## 2.2 List Components

```
124 \ccDeclareType{Components}{%
```

`Label` represents a List item's local label.

```

125   \ccDeclareComponent{Label}%
126   }%
127   \ccDeclareEnv{cc@list}{\endcc@list}%
128 }

```

### 3 Declaring List Types

List Types are the next layer of abstraction for lists. This layer distinguishes numbered from unnnumbered and description lists.

`\DeclareListType` declares a new list type. #1 is the name of the list type, #2 is the declaration body. Each new list type should declare at least an Attribute handler and a Label handler. #3 is a list of type specific properties that are appended to the generic list's property list.

```

129 \long\def\ccDeclareListType#1#2#3{%

```

`\DeclareAttributeHandler` declares a new handler for a list's attributes. ##1 is the definition body.

```

130   \def\DeclareAttributeHandler##1{\csdef{ccl@eval@attrs@#1}{##1}}%

```

`\DeclareLabelHandler` declares a new handler for each item's label. ##1 is the definition body. It should fill the Label Component with content in case the optional argument of item is omitted.

```

131   \def\DeclareLabelHandler##1{\csdef{ccl@make@label@#1}{##1}}%

```

```

132   \ccDeclareContainer{#1List}{%
133     \ccInherit{Components,Properties}{List}%
134     \ccDeclareType{Properties}{%

```

`list-type` <any> holds the name of the list type.

```

135     \ccSetProperty{list-type}{#1}%

```

```

136     #3%
137   }%
138   \ccDeclareEnv[#1-list]{\cc@list}{\endcc@list}%
139 }%
140 #2%
141 }

```

### 4 Declare Lists

The next layer of abstraction is the user-level List container. Each List container must be assigned to a list type from which it will inherit its type-specific properties.

`\ccDeclareList` defines a new list. #1 is the name of the list environment (sans `\ccPrefix`), #2 is the list type, #3 is the list-specific Property list.

```

142 \def\ccDeclareList#1#2#3{%
143   \csxdef{cc@cur@depth@#1}{\z@}%
144   \ccDeclareContainer{#1}{%
145     \ccInherit{Properties,Components}{#2List}%
146     \ccDeclareType{Properties}{#3}%
147     \ccDeclareEnv{#1}{\cc@list}{\endcc@list}%
148   }%
149   \ccDeclareNested{#1}{\z@}{#3}%
150 }

```

`\ccDeclareNested` can be used to declare Property overrides for nested lists. #1 is the list name, #2 is the nesting depth (#2th nesting level means that the Properties are used for the  $n + 1$ -th list of the same name), #3 is the Property list.

```

151 \def\ccDeclareNested#1#2#3{%
152   \@tempcnta=#2\relax
153   \ifx\@tempcnta<\@ne\relax
154     \ccPackageError{lists}{Nesting}{Invalid nesting level!}{You cannot declare nesting levels
155       less than 1!}%
156   \fi
157   \advance\@tempcnta\@ne\relax
158   \ccDeclareContainer{#1-\the\@tempcnta}{%
159     \ifcsdef{cc@container@#1}
160       {\ccInherit{Properties,Components}{#1}}
161       {\ccPackageError{lists}{Inheritance}
162         {List `#1' undefined!}
163         {You need to define the list `#1' before you can declare nested list overrides!}}%
164     \ccDeclareType{Properties}{#3}%
165   }%

```

We want to count each list type separately to ensure the correct item label is printed, but we also need to keep within the global nesting level limit. Therefore, we set two internal counters, one for the overall nesting level, and another one for each list type. Note that the latter is a macro, not a counter register.

`\ccl@depth` is the counter for the overall nesting level.

```

166 \newcount\ccl@depth

```

`\ccl@item@cnt` is the internal counter for the items within a (nested) list level.

```

167 \newcount\ccl@item@cnt

```

`\ifcclFirst` is true as long as the first item of a list is processed.

```

168 \newif\ifcclFirst \cclFirsttrue

```

`\ccl@advance@depth` is a helper macro to advance both the global list nesting level, as well as the list Container specific nesting level. #1 is the amount by which both counters should be advanced.

```

169 \def\ccl@advance@depth#1{\csname ccl@advance@depth@\ccl@nesting\endcsname{#1}}

```

`\ccl@advance@depth@global` is called when the nesting level should be counted for all lists equally without respecting the list type.

```
170 \def\ccl@advance@depth@global#1{%
171   \edef\cclPrevDepth{\the\ccl@depth}%
172   \global\advance\ccl@depth#1\relax
173   \edef\cclCurDepth{\the\ccl@depth}%
174 }
```

`\ccl@advance@depth@local` is called when the nesting level should be counted for each list type individually.

```
175 \def\ccl@advance@depth@local#1{%
176   \letcs\cclPrevDepth{cc@cur@depth@cc@cur@cont}%
177   \expandafter\@tempcnta\csname cc@cur@depth@cc@cur@cont\endcsname\relax
178   \advance\@tempcnta#1\relax
179   \csxdef{cc@cur@depth@cc@cur@cont}{\the\@tempcnta}%
180   \edef\cclCurDepth{\csname cc@cur@depth@cc@cur@cont\endcsname}%
181   \global\advance\ccl@depth#1\relax
182 }
```

`\cclItemIndent` stores the actual calculated indent of an List item's first line.

```
183 \newskip\cclItemIndent
```

`\cclTopID` is a counter that stores a unique number for each top-level List Instance. It is used to calculate the margins of both top-level items and items of nested lists.

```
184 \newcount\cclTopID \cclTopID\z@ \relax
```

`\cclID` stores a unique “identifier” number for each list, irrespective their nesting levels.

```
185 \newcount\cclID \cclID\z@ \relax
```

An internal global counter register `\ccl@total@list@cnt` is used to count the overall number of opening lists. Currently, the global ID of each list is unused.

```
186 \newcount\ccl@total@list@cnt \ccl@total@list@cnt\z@ \relax
```

`\ccl@incr@count` stores the current list ID counter in a nesting-depth specific macro `ccl@prev@cnt@the\ccl@depth`, advances the global internal list counter by one, and sets the publicly available counter `\cclID` to the resulting value. Also, if the nesting level is 1, the `\cclTopID` counter is incremented.

```
187 \def\ccl@incr@count{%
188   \csxdef{ccl@prev@cnt@the\ccl@depth}{\the\cclID}%
189   \global\advance\ccl@total@list@cnt\@ne \relax
190   \global\cclID\ccl@total@list@cnt \relax
191   \ifnum\cclCurDepth=\@ne \relax
192     \global\advance\cclTopID\@ne \relax
193   \fi
194 }
```

`\ccl@decr@count` resets the list counter for the next lower nesting level, whenever a nested list is closed.

```
195 \def\ccl@decr@count{%
196   \global\cclID\csname ccl@prev@cnt@the\ccl@depth\endcsname \relax
197 }
```

## 4.1 The List Environment

List environments have the same name as their respective containers (preixed by the `\ccPrefix`). However, they all call the low-level macros `\cc@list` and `\endcc@list`.

`\cc@list` is begin macro for the generalized coco-list environment. #1 is the attribute list of the environment.

```
198 \def\cc@list{\cc@opt@empty\cc@list}
199 \def\@cc@list[#1]{%
200   \ccl@advance@depth\@ne%
201   \ccl@incr@count%
202   \edef\ccl@cur@cont{\cc@cur@cont-\cclCurDepth}%
203   \global\cclFirsttrue
```

If the nesting goes deeper than the style programmer anticipated:

```
204 \ifcsdef{cc@container@\ccl@cur@cont}{%
205   {\ifx\ccl@inherit\ccl@ih@common
206     \let\ccl@cur@cont\cc@cur@cont%
207     \else
208       \global\csletcs
209         {cc@type@Properties@\cc@cur@cont-\cclCurDepth}
210         {cc@type@Properties@\cc@cur@cont-\cclPrevDepth}%
211       \fi}%
```

Horizontal margin Properties from the previous nesting level are stored so that the nested lists can use them.

```
212 \edef\ccl@leftskip{\the\dimexpr\leftskip\relax}%
213 \edef\ccl@rightskip{\the\dimexpr\rightskip\relax}%
```

`prev-margin-left <skip>` stores the left margin of the next higher list level (i. e., the left margin of the list item that the current list is nested into)

```
214 \ccSetPropertyX{prev-margin-left}{\ccl@leftskip}%
```

`prev-margin-right <skip>` stores the superior list item's right margin.

```
215 \ccSetPropertyX{prev-margin-right}{\ccl@rightskip}%
216 \ccEvalType[\ccl@cur@cont]{Properties}%
```

`\ccl@list@type` locally stores the current value of the `list-type` Property.

```
217 \edef\ccl@list@type{\ccUseProperty{list-type}}%
```

Processing of the optional argument.

```
218 \cclUseAttributeHandler{#1}%
```

The exact values of the margins are calculated.

```
219 \cclCalculateMarginLeft%
220 \cclCalculateVMargin{top}%
221 \cclCalculateVMargin{bottom}%
```

`\Item` is a used to separate the single items of a list.

```
222 \csdef{\ccPrefix Item}{\cc@opt@empty\ccl@item}%
223 \def\ccl@item[##1]{%
224   \edef\ccl@item@label{##1}%
```

```

225 \ifx\ccl@item@label\@empty
226 \cclUseLabelHandler%
227 \else
228 \ccComponent{Label}{##1}%
229 \fi
230 \sbox\z@{\@cc@is@finalfalse\ccUseProperty{label-format}}%
231 \@tempdima=\dimexpr\ccUseProperty{max-label-width}\relax
232 \ifdim\wd\z@<\@tempdima\relax
233 \@tempdima=\the\wd\z@\relax%
234 \fi
235 \bgroup
236 \def\cc@cur@cont{list}%
237 \cc@store@latest{\the\cclTopID-number-\cclCurDepth-maxwd}{\the\@tempdima}%
238 \cc@store@latest{\the\cclTopID-number-maxwd}{\the\@tempdima}%
239 \egroup
240 \ccSetPropertyX{label-width}{\the\@tempdima}%
241 \ccUseProperty{item-format}%
242 \ccUseProperty{at-begin-item-body}\ignorespaces%
243 }%

```

`\item` If default L<sup>A</sup>T<sub>E</sub>X macros are replaced per package option, `\item` is made into a copy of the local definition of `\ccPrefix Item`.

**Warning:** this might be dangerous when the User tries to embed something inside a CoCoT<sub>E</sub>X list that uses L<sup>A</sup>T<sub>E</sub>X's standard `\list` or `\trivlist` environments!

```

244 \if@cc@replace\letcs\item{\ccPrefix Item}\fi%

```

Up to this point, we only managed Properties. From this point forward, we actually print the list. We start by using the `☛before-list` Property.

```

245 \ccUseProperty{before-list}%

```

then, we add the top vertical skip by `☛int-margin-top` amount.

```

246 \ccUseProperty{int-margin-top}%

```

and set the left and right margins using the `☛margin-left`, `☛label-sep` and `☛margin-right` Properties.

```

247 \leftskip\dimexpr\ccUseProperty{margin-left}+\ccUseProperty{label-sep}\relax%
248 \rightskip\dimexpr\ccUseProperty{margin-right}\relax%
249 }

```

`\endcc@list` is called at the end of each List Container's respective environment. It basically calls the `☛after-list` Property one last time, decrements the depth counter(s) and adds the `☛int-margin-bottom` vertical skip.

```

250 \def\endcc@list{%
251 \ccUseProperty{after-list}%
252 \ccl@decr@count%
253 \ccl@advance@depth\m@ne%
254 \ccUseProperty{int-margin-bottom}%

```

If the List is not nested, we eventually evaluate the `☛after-indent` Property.

```

255 \ifnum\cclCurDepth=\z@\relax
256 \ccIfPropVal{after-indent}{false}{%
257 \global\@afterindentfalse
258 \aftergroup\cc@afterbox}%
259 \fi
260 }

```



`\cclCalculateVMargin` generates a macro that sets the internal margin Properties of the (nested) list. #1 is the orientation (`top` or `bottom`).

```

261 \def\cclCalculateVMargin#1{%
262   \ifdim\ccUseProperty{margin-#1}=\z@\relax
263     \ccSetProperty{int-margin-#1}{\relax}%
264   \else
265     \ccSetProperty{int-margin-#1}{\addvspace{\ccUseProperty{margin-#1}}}%
266   \fi
267 }
```

`\cclCalculateLeftMargin` generates the value that `\leftskip` is set to.

```

268 \def\cclCalculateMarginLeft{%
269   \ifcsdef{cc-list-\the\cclTopID-number-maxwd}
270     {\ccSetPropertyVal{number-width-max}{\csname cc-list-\the\cclTopID-number-maxwd\endcsname}}
271     {\ccSetPropertyVal{number-width-max}{1sp}}%
272   \ifcsdef{cc-list-\the\cclTopID-number-\cclCurDepth-maxwd}
273     {\ccSetPropertyVal{number-width-level-max}{\csname cc-list-\the\cclTopID-number-\cclCurDepth
274       -maxwd\endcsname}}
275     {\ccSetPropertyVal{number-width-level-max}{1sp}}%
276   \cc@get@indent[\ccl@calc@margin@left]{\the\cclTopID}%
277 }
```

`\ccl@calc@margin@left` is an override for coco-common's `\cc@calc@margin@left` specific for lists. According to `\cc@calc@margin@left`'s argument structure, #1 is the internal Property prefix, and #2 is the current value of the list depth counter. However, since we already stored the left margin of the previous depth level in the internal `\prev-margin-left` Property, we can gobble both arguments.

```

277 \def\ccl@calc@margin@left#1#2{%
278   \@tempdima=\ccUseProperty{prev-margin-left}\relax%
279   \ccSetPropertyX{margin-left}{\the\dimexpr\@tempdima-\ccUseProperty{indent}\relax}%
280 }
```

## 4.2 Unpacking the List Type-Specific Handlers

The caller macros for the two list type-specific Handlers for Attributes and Labels are defined here. They do some basic exception catching and then call the Handlers themselves if no error is detected.

`\cclUseLabelHandler` calls the list type specific Label handler to generate a label accordingly in cases where `\item` omits the optional argument.

```

281 \def\cclUseLabelHandler{%
282   \expandafter\ifx\csname ccl@make@label@\ccl@list@type\endcsname\relax
283     \ccPackageError{lists}{type}
284     {List type '\ccl@list@type' does not provide a Label Handler.}
285     {Make sure that the body of \ccl@list@type's declaration contains a \string\
286       DeclareLabelHandler.}
287   \else
288     \csname ccl@make@label@\ccl@list@type\endcsname
289   \fi
290 }
```

`\cclUseAttributeHandler` checks if the list type specific attribute handler exists and applies it to the attribute list #1.

```


290 \def\cclUseAttributeHandler#1{%
291   \ccParseAttributes{\cc@cur@cont-\cclCurDepth}{#1}%
292   \expandafter\ifx\csname ccl@eval@attrs@\ccl@list@type\endcsname\relax
293   \ccPackageError{Lists}{Type}
294     {List type '\ccl@list@type' does not provide an Attribute Handler.}
295     {Make sure that the body of \ccl@list@type's declaration contains a \string\
      DeclareAttributeHandler.}
296 \else
297   \csname ccl@eval@attrs@\ccUseProperty{list-type}\endcsname
298 \fi
299 }

```

## 5 Default List Types

Vanilla CoCoT<sub>E</sub>X supports three list types: numbered lists (corresponds to L<sup>A</sup>T<sub>E</sub>X's *enumerate* environment), unnumbered lists (*itemize*), and description lists (*description*).

### 5.1 Unnumbered Lists

`unnumbered` is technically an abstract child Container of the  `List` parent.

```

300 \ccDeclareListType{unnumbered}{%

```

`\ccl@make@label@unnumbered` generates the  `Label` Component of an unnumbered list type.

```

301   \DeclareLabelHandler{%
302     \ccComponent{Label}{\ccUseProperty{default-label}}

```

`\ccl@eval@attrs@itemize` is the handler for attributes of itemize-like list types. Currently, it does nothing.

```

303   \DeclareAttributeHandler{}}

```

#### Itemize-Type List Specific Properties


`default-label` <any> is a property that holds a fallback label which is used when the optional argument of `\Item` is omitted.

```

304   {\ccSetProperty{default-label}{-}}

```

#### Itemize-Style Default Lists

`Itemize` is the user-level unnumbered  `List` Container.

```

305 \ccDeclareList{Itemize}{unnumbered}{\ccSetProperty{default-label}{\textbullet}}
306 \ccDeclareNested{Itemize}{1}{%
307   \ccSetProperty{label-face}{\normalfont\bfseries}%
308   \ccSetProperty{default-label}{\textendash}}
309 \ccDeclareNested{Itemize}{2}{\ccSetProperty{default-label}{\textasteriskcentered}}
310 \ccDeclareNested{Itemize}{3}{\ccSetProperty{default-label}{\textperiodcentered}}

```

## 5.2 Numbered Lists

`\ccl@item@adv` is an internal counter that holds the amount by which the counter of numbered lists should advance for each item.

```
311 \newcount\ccl@item@adv
```

`numbered` is an abstract child Container of the List parent that represents numbered lists.

```
312 \ccDeclareListType{numbered}{%
```

`\ccl@eval@attrs@numbered` is the handler for attributes specific to the enumerate-like list types.

```
313 \DeclareAttributeHandler{%
```

The attribute `step` indicates by what amount the internal counter should be advanced for each item. Defaults to +1 if none is given.

```
314 \ccIfAttr{\cc@cur@cont-\cclCurDepth}{step}
315 {\ccl@item@adv=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@step\
    endcsname\relax}%
316 {\ccl@item@adv=\@ne}%
```

The attribute `start` indicates the initial internal counter of the items in the list. The number itself is the counter of the first item, so we need to subtract the value of `step` from the given value such that `\item` can advance it by that same value. If the attribute is not given, the internal counter is initialized to 0.

```
317 \ccIfAttr{\cc@cur@cont-\cclCurDepth}{start}
318 {\ccl@item@cnt=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@start\
    endcsname\relax
319 \advance\ccl@item@cnt-\ccl@item@adv}%
320 {\ccl@item@cnt=\z@\relax}%
321 }
```

`\ccl@make@label@numbered` is the [Label](#) handler of a numbered list type.

```
322 \DeclareLabelHandler{%
323 \advance\ccl@item@cnt \ccl@item@adv\relax
324 \expandafter\ifx\csname ccl@label@type@\ccUseProperty{enum-type}\endcsname\relax
325 \ccPackageWarning{lists}{type}{Enum type \ccUseProperty{enum-type} is unknown, revert to
    numeric counters!}
326 \let\ccl@label\ccl@label@type@arabic%
327 \else
328 \letcs\ccl@label{\ccl@label@type@\ccUseProperty{enum-type}}%
329 \fi
330 \ccComponent{Label}{\ccl@label{\ccl@item@cnt}}
331 }%
```

```
332 }{%
```

### Numbered List-Specific Properties

#### New Properties

`enum-type` [`arabic`|`roman`|`Roman`|`Alph`|`alph`] controls how the item counter is rendered when it is not given explicitly with the optional argument of `\item`. The default values are borrowed from LaTeX's default enumerate types and defined below.

```
333 \ccSetProperty{enum-type}{arabic}%
```

### Properties with Deviating Default Values

By default, numeric ➡**Label** are followed by a period to accommodate L<sup>A</sup>T<sub>E</sub>X customs.

```
334 \ccSetProperty{label}{\ccUseComp{Label}.}}
```

### Available Counting Styles

**\ccl@label@type@arabic** transforms the value of the following (implicit) counter to arabic numerals.

```
335 \def\ccl@label@type@arabic{\@arabic}
```

**\ccl@label@type@roman** transforms the value of the following (implicit) counrer to lower case roman numerals.

```
336 \def\ccl@label@type@roman{\@roman}
```

**\ccl@label@type@Roman** transforms the value of the following (implicit) counrer to upper case roman numerals.

```
337 \def\ccl@label@type@Roman{\@Roman}
```

**\ccl@label@type@alph** transforms the value of the following (implicit) counrer to lower case alphabetic letters.

```
338 \def\ccl@label@type@alph{\@alph}
```

**\ccl@label@type@Alph** transforms the value of the following (implicit) counrer to upper case alphabetic letters.

```
339 \def\ccl@label@type@Alph{\@Alph}
```

### Enumerate-Style Default Lists

**Enumerate** is the user-level Container for numbered 📦**List** Containers.

```
340 \ccDeclareList{Enumerate}{numbered}{%
341 \ccDeclareNested{Enumerate}{1}{% (
342 \ccSetProperty{label}{\ccUseComp{Label}}}%
343 \ccSetProperty{enum-type}{alph}}%
344 }
345 \ccDeclareNested{Enumerate}{2}{\ccSetProperty{enum-type}{roman}}
346 \ccDeclareNested{Enumerate}{3}{\ccSetProperty{enum-type}{Alph}}
```

## 5.3 Description Lists

**text** is an abstract child Container of the List parent used for **description**-like list types.

```
347 \ccDeclareListType{text}{%
```

**\ccl@eval@attrs@text** is the handler for the attributes of description-like list types.

```
348 \DeclareAttributeHandler{%
349 \ccIfAttr{\cc@cur@cont-\cclCurDepth}{width}
350 {\ccSetPropertyVal{min-margin-left}{\expandafter\dimexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@width\endcsname\relax}}%
351 {\ccSetProperty{min-margin-left}{2em}}%
352 \ccIfPropVal{label-growth}{down}
353 {\long\def\ccl@vbox##1{\smash{\vtop{##1}}}}
```

```

354   {\long\def\ccl@vbox##1{\vbox{##1}}}%
355 }

```

`\ccl@make@label@text` creates the label of a description-like list type.

```

356 \DeclareLabelHandler{%
357   \ccComponent{Label}{}%
358 }%

```

## Description-Type Specific Properties

### New Properties

`label-growth` [`up`|`down`] controls the direction labels “grow” into when they need more space than `max-label-width`. On  $\text{\TeX}$ -primitive level, it controls whether the label is put into a `\vbox` or `\vtop` with `\hsize=\cclItemIndent`.

**Important note:** If the `label-growth` is set to ‘down’ and the description of an item uses less lines than its label, the label *will* flow into the next item. There is no (easy) way to catch that (automatically) without destroying the possibility to nesting lists.

```

359 {\ccSetProperty{label-growth}{up}%

```

### Properties with Deviating Default Values

The Properties `margin-left` and `indent` of text-type lists are by default set to `auto`.

```

360 \ccSetProperty{indent}{auto}%
361 \ccSetProperty{margin-left}{auto}%

```


To accommodate for the new `label-grow` option, the `label-box` has a conditional that switches between regular `\hbox` labels and the two `\vbox` variants described above.

```

362 \ccSetProperty{label-box}{%
363   \ifdim\ccUseProperty{label-width}<\ccUseProperty{max-label-width}\relax
364     \hbox to \cclItemIndent{%
365       \ccIfPropVal{label-align}{left}{-}{\hss}%
366       \ccUseProperty{label-format}%
367       \ccIfPropVal{label-align}{right}{-}{\hss}}%
368   \else
369     \ccl@vbox{\relax%
370       \hsize\dimexpr\cclItemIndent%
371       \leftskip\z@
372       \rightskip\z@
373       \parindent\z@
374       \leavevmode
375       \ccUseProperty{label-format}%
376       \@@par
377     }%
378   \fi
379 }%

```

## Description-Type Default Lists

`Description` is the user-level Container for text type  List Containers.

As with the standard  $\text{\LaTeX}$ `description` environment, there are no default definitions for nested Description-type lists.

```

380 \ccDeclareList{Description}{text}{%
381   \ccSetProperty{label-face}{\bfseries}
382 }

```

## 5.4 Replacing L<sup>A</sup>T<sub>E</sub>X's Default Lists

At the User's descretion (using the `replace` package option, see Sect. 1.1, above), L<sup>A</sup>T<sub>E</sub>X's default list environments `itemize`, `enumerate`, and `description` are re-defined to use CoCoT<sub>E</sub>X's list mechanism, instead.

```

383 \if@ccl@replace
384   \letcs\itemize{\ccPrefix Itemize}
385   \letcs\enditemize{end\ccPrefix Itemize}
386   \letcs\enumerate{\ccPrefix Enumerate}
387   \letcs\endenumerate{end\ccPrefix Enumerate}
388   \letcs\description{\ccPrefix Description}
389   \letcs\enddescription{end\ccPrefix Description}
390 \fi

```

```
</lists>
```

# Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.

## Macro and Environment Index

In this index, the cc(®)- and module specific ccX(®)-Prefixes were omitted when sorting the entries.

Symbols			
\@endnotemark	108	\cc@assign@res	22
\@gobbleopt	38	\ccf@attr@class	159
\cc@@parse@csv	31	\ccf@attr@debug	160
\cch@2@unique	85	\ccf@attr@orient	159
\cch@3@level	85	\ccf@attr@pos	159
		\ccAttrVal	31
		\cch@auto@number	93
A		B	
\cc@abspage	39	\cct@book@titlepage	129
\cch@add@autoclose	86	\ccBreak	40
\cch@add@before@skip	91		
\cct@add@eval	124	C	
\ccaAddAltText	58	\c@endnote	110
\ccaAddColSpan	62	\cc@calc@margin@left	45
\ccf@addcontentsline	161	\ccl@calc@margin@left	191
\ccaAddFigure	61	\ccf@calc@row@ht	164
\ccaAddID	58	\ccf@calc@sameheight	164
\ccaAddKeep	62	\ccf@calc@width	147
\ccaAddLastLink	58	\CalcModulo	38
\ccaAddNumbering	58	\CalcRatio	38
\ccaAddPlacement	58	\cclCalculateLeftMargin	191
\ccaAddRolemap	58	\cclCalculateVMargin	191
\ccaAddRowSpan	62	\ccf@capt@bottom@offset	166
\ccaAddScope	62	\ccf@capt@top@offset	166
\ccAddTitleEval	124	\cc@check@empty	20
\ccAddTitleRole	124	\ccf@check@empty	161
\ccAddToComponentGroup	21	\ccCheckParent	16
\ccAddToProperties	27	\cc@cnt@grp	21
\ccAddToRole	72	\CoCoTeX	34
\ccaAddToStruct	57	\cc@comp@edef	24
\ccAddToType	15	\ccCompLink	49
\ccl@advance@depth	187	\ccComponent	17
\ccl@advance@depth@global	188	\ccComponentEA	18
\ccl@advance@depth@local	188	\cc@compose@collection	23
\cc@afterbox	37	\ccf@compose@listof	161
\ccAfterClassHook	6	\ccComposeCollection	22, 68
\cc@apply@collection	23	\ccContentsline	41
\ccApplyCollection	22, 68	\ccaConvertPdfString	63
\ccAppPropLocal	28	\cc@counted@comp@scheme	24, 150
\ccAppToProp	26	\ccf@create@counter	162
\cct@article@titlepage	129	\cct@create@editor@string	130
\ccaArtifact	61		

<code>\cc@create@label</code> .....	48	<code>\ccdefFromComp</code> .....	18
<code>\cch@create@parent</code> .....	89	<code>\ccdefFromProperty</code> .....	27
<code>\ccCreateContentListEntries</code> .....	43	<code>\ccaEnable</code> .....	54
<code>\ccCreateLabel</code> .....	47	<code>\cch@end@heading</code> .....	98
<code>\ccfCreateLabel</code> .....	163	<code>\endcc@list</code> .....	190
<code>\CsToStr</code> .....	55	<code>\endccaArtifact</code> .....	61
<code>\ccCurComp</code> .....	72	<code>\endccf@float</code> .....	157
<code>\ccCurCount</code> .....	22	<code>\endccSubFloat</code> .....	158
<code>\ccCurInfix</code> .....	72	<code>\endcctmeta</code> .....	124
<code>\ccCurSecName</code> .....	97	<code>\enoteformat</code> .....	109
<b>D</b>			
<code>\cc@debug@@message</code> .....	13	<code>\enoteheading</code> .....	109
<code>\ccDebugMsg</code> .....	13	<code>\enotesize</code> .....	109
<code>\ccm@declare@affils</code> .....	75	<code>\ccl@eval@attrs@itemize</code> .....	192
<code>\cca@declare@cmd</code> .....	55	<code>\ccl@eval@attrs@numbered</code> .....	193
<code>\cca@declare@cmd@firstopt</code> .....	55	<code>\ccl@eval@attrs@text</code> .....	194
<code>\cca@declare@cmd@secopt</code> .....	55	<code>\ccf@eval@class</code> .....	159
<code>\ccm@declare@comp</code> .....	73	<code>\ccm@eval@role</code> .....	69
<code>\cch@declare@heading</code> .....	86, 89	<code>\ccEvalAttributes</code> .....	31
<code>\cct@declare@role</code> .....	123	<code>\ccEvalType</code> .....	15
<code>\DeclareAccessibilityCommand</code> .....	55	<code>\cc@expand@l@contents</code> .....	42
<code>\DeclareAttributeHandler</code> .....	186	<code>\ccm@extended@common@macros</code> .....	74
<code>\ccDeclareAttributeHandler</code> .....	31	<code>\cc@extract@generic</code> .....	42
<code>\ccDeclareAttributeHandler*</code> .....	31	<b>F</b>	
<code>\ccDeclareClass</code> .....	33	<code>\ccaFigureEnd</code> .....	62
<code>\ccDeclareComponent</code> .....	17	<code>\ccfFigureRender</code> .....	170
<code>\ccDeclareComponentGroup</code> .....	21	<code>\ccaFigureStart</code> .....	61
<code>\ccDeclareContainer</code> .....	14	<code>\ccf@float</code> .....	156
<code>\ccDeclareContentList</code> .....	42	<code>\ccFloatBarrier</code> .....	174
<code>\ccDeclareCountedComponent</code> .....	25	<code>\ccf@floatbox</code> .....	147
<code>\ccDeclareEnv</code> .....	15	<code>\footnote</code> .....	109
<code>\ccDeclareFloat</code> .....	168	<code>\cc@format@number</code> .....	44
<code>\ccDeclareGlobalComponent</code> .....	17	<code>\cct@fundings@comp</code> .....	130
<code>\ccDeclareGroupHandler</code> .....	21	<code>\cct@fundings@eval</code> .....	130
<code>\ccDeclareHeading</code> .....	84, 86, 86, 106, 106	<b>G</b>	
<code>\ccDeclareLabeledComp</code> .....	72	<code>\ccpgdefFromComp</code> .....	18
<code>\DeclareLabelHandler</code> .....	186	<code>\ccpgdefFromProperty</code> .....	28
<code>\ccDeclareList</code> .....	187	<code>\ccf@generate@listof@handlers</code> .....	160
<code>\DeclareListType</code> .....	186	<code>\ccm@generic@comp</code> .....	67
<code>\ccDeclareNested</code> .....	187	<code>\ccm@generic@eval</code> .....	68
<code>\ccDeclareRole</code> .....	69	<code>\cc@get@indent</code> .....	45
<code>\ccDeclareRoleBlock</code> .....	68	<code>\ccf@get@seps</code> .....	149
<code>\ccDeclareTitlepage</code> .....	124	<code>\ccGetAttribute</code> .....	32
<code>\ccDeclareType</code> .....	15	<code>\ccGetComp*</code> .....	19
<code>\ccl@decr@count</code> .....	188	<code>\ccaGetCurrentStruct</code> .....	57
<code>\cc@def@counted@comp</code> .....	25, 149	<code>\ccaGetCurStruct</code> .....	57
<code>\ccgdefFromComp</code> .....	18	<code>\ccaGetStructParent</code> .....	59
<code>\ccedefFromCountedComp</code> .....	24	<code>\ccfGetTableContent</code> .....	174
<code>\ccxdefFromCountedComp</code> .....	24	<code>\cca@Gin@alt</code> .....	58
<code>\ccgdefFromProperty</code> .....	27	<code>\ccGlobalComponent</code> .....	18
<code>\ccl@depth</code> .....	187	<code>\ccGobble</code> .....	38
<code>\ccaDisable</code> .....	54	<b>H</b>	
<code>\cca@do@doparas</code> .....	52	<code>\Hack</code> .....	38
<code>\cca@do@dospaces</code> .....	52	<code>\hack</code> .....	38
<code>\cc@do@inherit</code> .....	16	<code>\Hackfor</code> .....	38
<code>\cca@do@nodetree</code> .....	52	<code>\hackfor</code> .....	38
<code>\cca@do@showspaces</code> .....	52	<code>\cch@heading</code> .....	97
<b>E</b>			
<code>\cct@eds@eval</code> .....	130	<code>\cchHeadTagEnd</code> .....	106
		<code>\cchHeadTagStart</code> .....	106



<code>\cch@highest@level</code> .....	89	<code>\cch@lowest@level</code> .....	89
<code>\ccf@ltx@includegraphics</code> .....	146	<code>\cc@ltx@label</code> .....	47
<b>I</b>			
<code>\cclID</code> .....	188	<b>M</b>	
<code>\if@cc@is@final</code> .....	14	<code>\cch@make@block</code> .....	96
<code>\if@cc@odd</code> .....	39	<code>\cch@make@bookmarks</code> .....	95
<code>\if@ccf@break@capt</code> .....	148	<code>\ccf@make@caption</code> .....	166
<code>\if@ccf@sameheight</code> .....	148	<code>\cch@make@inline</code> .....	96
<code>\if@ccn@en@links</code> .....	107	<code>\ccl@make@label@numbered</code> .....	193
<code>\if@ccn@use@en</code> .....	107	<code>\ccl@make@label@text</code> .....	195
<code>\if@enotesopen</code> .....	108	<code>\ccl@make@label@unnumbered</code> .....	192
<code>\ccIfAlly</code> .....	14, 52	<code>\ccf@make@outer@caption</code> .....	167
<code>\ccIfAttr</code> .....	32	<code>\cch@make@run</code> .....	95
<code>\ccIfAttrIsSet</code> .....	33	<code>\ccf@make@subcaption</code> .....	168
<code>\ccIfAttrIsStr</code> .....	33	<code>\cch@make@toc</code> .....	94
<code>\ifcclFirst</code> .....	184, 187	<code>\ccf@make@comp</code> .....	149
<code>\ccIfComp</code> .....	19	<code>\ccf@make@compL</code> .....	150
<code>\ccIfCompEmpty</code> .....	20	<code>\cct@maketitle</code> .....	129
<code>\ccIfCompFrom</code> .....	19	<code>\ccf@margin@i</code> .....	148
<code>\ccIfCompFromEmpty</code> .....	20	<code>\ccf@margin@l</code> .....	148
<code>\ccIfCompFromVal</code> .....	19	<code>\ccf@margin@o</code> .....	148
<code>\ccIfComponentOverride</code> .....	22	<code>\ccf@margin@r</code> .....	148
<code>\ccIfPreamble</code> .....	36	<code>\cch@max@level</code> .....	89
<code>\ccIfProp</code> .....	29	<code>\ccf@measuresubgraphics</code> .....	171
<code>\ccIfPropVal</code> .....	29	<code>\cctmeta</code> .....	124
<code>\ccIfStrEqual</code> .....	28	<code>\cch@min@level</code> .....	89
<code>\ccl@ih@common</code> .....	181	<code>\minusvspace</code> .....	38, 101
<code>\ccl@ih@conseq</code> .....	181	<code>\ccaMoveStruct</code> .....	57
<code>\ccf@includesubgraphics</code> .....	170	<b>N</b>	
<code>\ccl@incr@count</code> .....	188	<code>\ccl@nesting</code> .....	182
<code>\ccInherit</code> .....	14	<code>\ccNewPar</code> .....	106
<code>\cc@inherit</code> .....	16	<b>O</b>	
<code>\ccl@inherit</code> .....	182	<code>\cc@opt@curcont</code> .....	29
<code>\cch@init@cnt</code> .....	92	<code>\cc@opt@empty</code> .....	29
<code>\cch@init@hooks</code> .....	92	<code>\cc@opt@second</code> .....	29
<code>\cc@init@l@</code> .....	41	<b>P</b>	
<code>\ccInlineFigure</code> .....	172	<code>\ccPackageError</code> .....	12
<code>\ccf@int@cnt</code> .....	147	<code>\ccPackageInfo</code> .....	12
<code>\ccf@int@sub@flt@cnt</code> .....	147	<code>\ccPackageWarning</code> .....	12
<code>\Item</code> .....	189, 192	<code>\ccPageLabel</code> .....	49
<code>\item</code> .....	190	<code>\ccaPagestyleArtifacts</code> .....	60
<code>\ccl@item@adv</code> .....	193	<code>\ccn@parindent</code> .....	108
<code>\ccl@item@cnt</code> .....	187	<code>\cc@parse@attributes</code> .....	30
<code>\cclItemIndent</code> .....	184, 188	<code>\cc@parse@csv</code> .....	31
<code>\cc@iterate</code> .....	29	<code>\cc@parse@inherit</code> .....	16
<b>J</b>		<code>\cc@parse@kv</code> .....	30
<code>\cct@journal@titlepage</code> .....	129	<code>\ccParseAttributes</code> .....	30
<b>L</b>		<code>\cca@patch@error</code> .....	52
<code>\ccl@label@type@Alph</code> .....	194	<code>\PDF@FinishDoc</code> .....	63
<code>\ccl@label@type@alph</code> .....	194	<code>\ccPI</code> .....	29
<code>\ccl@label@type@arabic</code> .....	194	<code>\ccPrefix</code> .....	13
<code>\ccl@label@type@Roman</code> .....	194	<code>\ccPrePropLocal</code> .....	28
<code>\ccl@label@type@roman</code> .....	194	<code>\ccPreToProp</code> .....	26
<code>\cca@lang@id</code> .....	51	<code>\cc@print@generic</code> .....	42
<code>\cc@list</code> .....	189	<code>\printnotes</code> .....	109
<code>\ccl@list@type</code> .....	189	<code>\ccf@process</code> .....	164
<code>\cc@listof@extract@data</code> .....	160	<code>\ccPropertyLet</code> .....	27
<code>\cc@listof@print@entry</code> .....	161		
<code>\cc@long@empty</code> .....	20		

<code>\ccPropertyLetX</code> .....	27	<code>\cc@str@table</code> .....	36
<code>\ccaProtect</code> .....	54	<code>\cc@str@top</code> .....	37
<code>\cca@protected@defs</code> .....	54	<code>\strip@longprefix</code> .....	18
<code>\cch@provide@authors</code> .....	99	<code>\ccaStructEnd</code> .....	56
<code>\cch@provide@comp</code> .....	99	<code>\ccaStructStart</code> .....	56
<code>\cc@provide@overrides</code> .....	99	<code>\ccf@sub@box</code> .....	147
<code>\cch@provide@quotes</code> .....	98	<code>\ccf@sub@maxheight</code> .....	147
<code>\ccaPstructEnd</code> .....	57	<code>\ccf@sub@sep</code> .....	147
<code>\ccaPstructStart</code> .....	56	<code>\ccfSubFigureRender</code> .....	170
<b>R</b>		<code>\ccSubFloat</code> .....	158
<code>\cca@relaxed@defs</code> .....	54	<code>\ccSubFloatCnt</code> .....	147
<code>\ccf@render@sub</code> .....	159	<code>\ccfSubTableRender</code> .....	174
<code>\ccfRenderSubFloats</code> .....	158	<b>T</b>	
<code>\ccaReplaceStruct</code> .....	57	<code>\ccfTableRender</code> .....	174
<code>\cch@reserve</code> .....	98	<code>\cca@temp@signature</code> .....	55
<code>\ccf@reserve@htmltab</code> .....	173	<code>\cc@tempboxa</code> .....	37
<code>\ccf@reserve@tab</code> .....	173	<code>\cc@tempboxb</code> .....	37
<code>\ccf@reserve@tabular</code> .....	172	<code>\cc@tempskipa</code> .....	37
<code>\cch@reset</code> .....	98	<code>\ccf@test@caption</code> .....	165
<code>\cc@reset@components</code> .....	25	<code>\ccf@test@subcapt</code> .....	166
<code>\ccf@reset@defaults</code> .....	148	<code>\ccTestPage</code> .....	40
Role Block .....	68	<code>\the@cc@thispage</code> .....	39
<code>\ccm@role@apply</code> .....	69	<code>\thecc@abspage</code> .....	39
<code>\ccm@role@compose</code> .....	69	<code>\cct@title@insert@xmp</code> .....	128
<code>\ccm@role@eval</code> .....	68	<code>\cct@title@insert@xmp@direct</code> .....	128
<code>\cct@role@handlers</code> .....	123	<code>\cct@title@insert@xmp@ltpdfa</code> .....	128
<b>S</b>		<code>\cct@title@process@bka</code> .....	127
<code>\cca@saved@defs</code> .....	54	<code>\cct@title@process@bkc</code> .....	127
<code>\ccSavePage</code> .....	40	<code>\cct@title@process@bkk</code> .....	127
<code>\ccf@sep@bottom</code> .....	148	<code>\cct@title@process@bkt</code> .....	126
<code>\ccf@sep@top</code> .....	147	<code>\TitleBreak</code> .....	106
<code>\ccf@set*@sep</code> .....	149	<code>\cc@toc@extract@data</code> .....	94
<code>\cch@set@author@name@list</code> .....	93	<code>\cc@toc@print@entry</code> .....	94
<code>\cca@set@docinfo</code> .....	58	<code>\ccTocLink</code> .....	41
<code>\ccf@set@env</code> .....	149	<code>\ccToggleCountedConditionals</code> .....	25
<code>\cc@set@hang</code> .....	45	<code>\ccToggleCountedConditionalsHook</code> .....	6
<code>\ccf@set@hsize</code> .....	163	<code>\cclTopID</code> .....	188
<code>\ccf@set@margins</code> .....	163	<code>\cc@topstrut</code> .....	37
<code>\cct@set@pdfmeta</code> .....	125	<code>\ccf@total@depth</code> .....	147
<code>\cc@set@property@local</code> .....	28	<code>\ccf@total@height</code> .....	147
<code>\ccf@set@subcounter</code> .....	162	<code>\ccf@total@width</code> .....	147
<code>\ccf@set@top@counter</code> .....	162	<b>U</b>	
<code>\ccSetBabelLabel</code> .....	48	<code>\ccUnlessAally</code> .....	14, 52
<code>\ccSetContainer</code> .....	15	<code>\ccUnlessAttr</code> .....	33
<code>\ccSetProperty</code> .....	26	<code>\ccUnlessComp</code> .....	19
<code>\ccSetPropertyVal</code> .....	27	<code>\cch@use@hook</code> .....	91
<code>\ccSetPropertyX</code> .....	27	<code>\ccn@use@TeX@heading</code> .....	108
<code>\ccSetPropLocal</code> .....	28	<code>\cclUseAttributeHandler</code> .....	191
<code>\cct@simple@comps</code> .....	130	<code>\ccUseComp</code> .....	18
<code>\cct@split@pdf@meta</code> .....	126	<code>\ccUseCompByIndex</code> .....	21
<code>\cc@store@comp</code> .....	18	<code>\ccUseComponentFrom</code> .....	19
<code>\ccf@store@dimens</code> .....	157	<code>\ccUseHeading</code> .....	85, 89
<code>\cc@store@latest</code> .....	43	<code>\ccUseLabeledComp</code> .....	72
<code>\cc@store@prop</code> .....	27	<code>\cclUseLabelHandler</code> .....	191
<code>\cc@str@bottom</code> .....	37	<code>\ccUseProperty</code> .....	27
<code>\cc@str@default</code> .....	36	<code>\ccUsePropertyEnv</code> .....	28
<code>\cc@str@figure</code> .....	36	<code>\ccUsePropFrom</code> .....	21
<code>\ccl@str@global</code> .....	182	<code>\ccUseStyleClass</code> .....	34
<code>\ccl@str@local</code> .....	182		

<b>V</b>		
\ccaVstructEnd .....	<u>56</u>	\cct@write@hr@infodict ..... <u>126</u>
\ccaVstructStart .....	<u>56</u>	\ccf@write@listof ..... <u>162</u>
		\cct@write@pdf@meta@list ..... <u>125</u>
		\cct@write@pdf@meta@string ..... <u>125</u>
<b>W</b>		
\ccWhenAlly .....	<u>14</u> , <u>52</u>	
\ccWhenAttr .....	<u>32</u>	
\ccWhenComp .....	<u>19</u>	
		<b>X</b>
		\ccpxdefFromCountedComp ..... <u>24</u>

## Container Index

<b>A</b>	
article-meta .....	<u>73</u>
<b>C</b>	
CommonMeta .....	<u>67, 83</u>
<b>D</b>	
Description .....	<u>195</u>
<b>E</b>	
Enumerate .....	<u>194</u>
<b>F</b>	
Figure .....	<u>169</u>
float .....	<u>150</u>
<b>G</b>	
Graphic .....	<u>169</u>
<b>H</b>	
Heading .....	<u>83</u>
<b>I</b>	
InlineFigure .....	<u>171</u>
Itemize .....	<u>192</u>
<b>L</b>	
List .....	<u>182, 192, 194, 195</u>
<b>N</b>	
numbered .....	<u>193</u>
<b>T</b>	
Table .....	<u>173</u>
Tabular .....	<u>172</u>
text .....	<u>194</u>
titlepage .....	<u>123</u>
<b>U</b>	
unnumbered .....	<u>192</u>

## Component Index

In this index, the name in parentheses denote the (abstract) Container within which the Component entry is defined.

<b>A</b>	
Abstract (Generic) . . . . .	74
Abstract (Heading) . . . . .	101
Accepted (Generic) . . . . .	74
Acknowledgements (titlepage) . . . . .	130
AddNoteI (titlepage) . . . . .	136
AddNoteII (titlepage) . . . . .	136
AddNoteIII (titlepage) . . . . .	137
AddNoteIV (titlepage) . . . . .	137
Address (Generic) . . . . .	75
Advertise (titlepage) . . . . .	135
Affil (Generic) . . . . .	79
AffilBlock (Generic) . . . . .	75, 75, 79, 80
AffilBlock (Heading) . . . . .	101
Affiliation (Generic) . . . . .	70, 70, 75, 79
AffilID (Generic) . . . . .	75
AffilRef (Generic) . . . . .	70, 70, 75
AffilRef (Heading) . . . . .	105
AltNames (titlepage) . . . . .	131
AltText (float) . . . . .	169, 170
AltTitle (titlepage) . . . . .	131
Author (Heading) . . . . .	83, 99
AuthorCitationList (Generic) . . . . .	71, 79
AuthorContact (Heading) . . . . .	99, 99, 105
AuthorContactBlock (Heading) . . . . .	99, 105
AuthorContribution (Generic) . . . . .	79
AuthorCorrespondence (Generic) . . . . .	72
AuthorNameList (Generic) . . . . .	71, 71, 79
AuthorNameList (Heading) . . . . .	99, 101, 105
AuthorNameList (titlepage) . . . . .	131
AuthorPDFInfo (Generic) . . . . .	71, 79
AuthorShortCitationList (Generic) . . . . .	71, 79
<b>B</b>	
BibleISSN (titlepage) . . . . .	134
Biblio (titlepage) . . . . .	134
BiblioTitle (titlepage) . . . . .	134
BiblISSN (titlepage) . . . . .	134
BMNumber (Heading) . . . . .	84, 105
BMSubtitle (Heading) . . . . .	84
BMTitle (Heading) . . . . .	84, 105
<b>C</b>	
Caption (float) . . . . .	150
CC (Generic) . . . . .	67
CitationList (Generic) . . . . .	71, 77
CiteAs (Generic) . . . . .	74
CiteName (Generic) . . . . .	70, 70, 71, 77, 78
COIStatement (Generic) . . . . .	74
Component (titlepage) . . . . .	135
Content (float) . . . . .	150
Conversion (titlepage) . . . . .	135
Copyright (Generic) . . . . .	73
CopyrightDisclaimer (titlepage) . . . . .	136
CorrespondanceAs (Generic) . . . . .	72
Correspondence (Generic) . . . . .	72, 78
CorrespondenceAs (Generic) . . . . .	71, 77, 79
Country (Generic) . . . . .	75
Cover (titlepage) . . . . .	130
CoverConcept (titlepage) . . . . .	135
CoverDesign (titlepage) . . . . .	135
<b>D</b>	
Dedication (titlepage) . . . . .	130
Department (Generic) . . . . .	75
DocTitle (titlepage) . . . . .	131
DOI (Generic) . . . . .	75
<b>E</b>	
Edition (titlepage) . . . . .	133
EditionNote (titlepage) . . . . .	133
Editorial (titlepage) . . . . .	131
EditorNameList (titlepage) . . . . .	131, 132
EISBN (titlepage) . . . . .	134
EISSN (titlepage) . . . . .	133
ElibPDF (titlepage) . . . . .	134
Email (Generic) . . . . .	71
EndPage (Generic) . . . . .	74
EnvDisclaimer (titlepage) . . . . .	135
EpubISBN (titlepage) . . . . .	134
EpubPreText (titlepage) . . . . .	134
<b>F</b>	
Fig (float) . . . . .	169
FirstName (Generic) . . . . .	70, 70, 76
FullName (Generic) . . . . .	69, 71, 76, 79
FullName (Heading) . . . . .	105
FundingPostText (titlepage) . . . . .	134
FundingPreText (titlepage) . . . . .	134
<b>H</b>	
Heading (Generic) . . . . .	68
Honorific (Generic) . . . . .	70
<b>I</b>	
Initial (Generic) . . . . .	70, 76
Institute (Generic) . . . . .	75
ISBN (titlepage) . . . . .	133
ISBNPreText (titlepage) . . . . .	133
ISSN (titlepage) . . . . .	133
Issue (titlepage) . . . . .	136
<b>J</b>	
JournalAbbrev (titlepage) . . . . .	136
JournalName (titlepage) . . . . .	136
<b>K</b>	
Keywords (Generic) . . . . .	74
Keywords (Heading) . . . . .	101

- L**
- Label (Generic) ..... 80, 81
- Label (List) ..... 184, 185, 186, 192–194
- LastName (Generic) ..... 70
- Lectorate (titlepage) ..... 135
- Legend (float) ..... 150
- LicenceLink (titlepage) ..... 136
- LicenceLogo (Generic) ..... 73
- LicenceName (Generic) ..... 73
- LicenceName (titlepage) ..... 136
- LicenceText (titlepage) ..... 135
- LicenseLogo (titlepage) ..... 135
- Lineage (Generic) ..... 70
- ListofCaption (float) ..... 150
- ListofEntry (float) ..... 151
- ListofLegend (float) ..... 150
- ListofNumber (float) ..... 150
- ListofSource (float) ..... 150
- M**
- MemberList (titlepage) ..... 136
- MidName (Generic) ..... 70, 70, 76
- N**
- NameList (Generic) ..... 71, 71, 77, 78
- Number (Heading) ..... 84, 101, 102
- Number (float) ..... 150
- Number (titlepage) ..... 132
- O**
- ORCID (Generic) ..... 70
- ORCID (Heading) ..... 105
- P**
- PDFCreator (titlepage) ..... 133
- PDFInfo (Generic) ..... 71, 77
- PDFInfoName (Generic) ..... 70, 77, 79
- PDFProducer (titlepage) ..... 133
- Prices (titlepage) ..... 136
- Print (titlepage) ..... 134
- PrintNote (titlepage) ..... 134
- PubCycle (titlepage) ..... 136
- PubDivInfo (titlepage) ..... 132
- PubDivision (titlepage) ..... 132
- Published (Generic) ..... 74
- Publisher (titlepage) ..... 132
- PubLogo (titlepage) ..... 132
- PubNote (titlepage) ..... 133
- PubPlace (titlepage) ..... 132
- PubWeb (titlepage) ..... 133
- Q**
- QA (titlepage) ..... 135
- Quote (Heading) ..... 98, 98
- QuoteBlock (Heading) ..... 98, 101
- QuoteSource (Heading) ..... 99, 100
- QuoteText (Heading) ..... 98, 100
- R**
- Received (Generic) ..... 74
- RefLabel (Heading) ..... 84
- RefLabel (float) ..... 150
- Reviewed (Generic) ..... 74
- Revised (Generic) ..... 74
- RunAuthorNameList (Heading) ..... 103
- RunNames (titlepage) ..... 131
- RunNumber (Heading) ..... 84
- RunSubtitle (Heading) ..... 84
- RunTitle (Heading) ..... 84, 103, 110
- RunTitle (titlepage) ..... 131
- S**
- Series (titlepage) ..... 132
- SeriesEditorNameList (titlepage) ..... 132
- SeriesNote (titlepage) ..... 132
- ShortCitationList (Generic) ..... 71, 77
- ShortCiteName (Generic) ..... 70, 71, 77, 78
- ShortTitle (titlepage) ..... 131
- Source (float) ..... 150
- StartPage (Generic) ..... 73
- Startpage (titlepage) ..... 136
- Statement (titlepage) ..... 131
- Submitted (Generic) ..... 74
- SubSeries (titlepage) ..... 132
- Subtitle (Heading) ..... 84, 101
- Subtitle (titlepage) ..... 131
- T**
- Title (Heading) ..... 84, 101, 110
- Title (titlepage) ..... 131
- TitleEn (Generic) ..... 75
- TitleNote (titlepage) ..... 131
- TocAuthorNameList (Heading) ..... 105
- TocNumber (Heading) ..... 84, 104, 105
- TocPage (Heading) ..... 105
- TocSubtitle (Heading) ..... 84
- TocTitle (Heading) ..... 84, 105
- Translator (titlepage) ..... 135
- Typesetter (titlepage) ..... 135
- U**
- UsedFont (titlepage) ..... 135
- V**
- Volume (titlepage) ..... 132
- X**
- XmpFile (titlepage) ..... 133
- Y**
- Year (titlepage) ..... 133

## Property Index

In this index, the name in parentheses denote the (abstract) Container within which the Property entry is defined.

<b>A</b>	
affil-block-face (Generic) .....	80
affil-block-format (Generic) .....	80
affil-block-item-face (Generic) .....	79
affil-block-item-format (Generic) .....	80
affil-sep (Generic) .....	79
affiliation-format (Generic) .....	75, 79
after-heading-block (Heading) .....	101
after-heading-par (Heading) .....	100
after-indent (Heading) .....	102
after-indent (List) .....	183, 190
after-item (List) .....	183, 184, 184
after-list (List) .....	183, 190
after-skip (Heading) .....	102
at-begin-item-body (List) .....	183
at-end-item-body (List) .....	183, 184
author-cite-name-format (Generic) .....	78
author-contact-block-format (Heading) .....	105
author-contact-format (Heading) .....	105
author-correspondence-as-format (Generic) .....	79
author-face (Heading) .....	100
author-full-name-format (Generic) .....	79
author-list-cite-format (Generic) .....	79
author-list-correspondence-format (Generic) .....	79
author-list-format (Heading) .....	105
author-list-pdfinfo-format (Generic) .....	79
author-list-print-format (Generic) .....	79
author-list-short-cite-format (Generic) .....	79
author-pdfinfo-name-format (Generic) .....	79
author-short-cite-name-format (Generic) .....	78
auto-number-prefix (float) .....	153
auto-number-prefix-sep (float) .....	153
<b>B</b>	
before-float (float) .....	151, 156
before-heading (Heading) .....	100
before-heading-block (Heading) .....	102
before-item (List) .....	184, 185
before-list (List) .....	182, 190
before-skip (Heading) .....	101
bookmark (Heading) .....	105
bookmark-level (Heading) .....	105
<b>C</b>	
caption-bottom (float) .....	152, 153, 165, 166
caption-face (float) .....	152, 164, 166
caption-face-bottom (float) .....	152, 153, 164, 166
caption-face-top (float) .....	152, 153, 164, 166
caption-sep-bottom (float) .....	153, 166
caption-sep-top (float) .....	152, 153, 166
caption-top (float) .....	152, 153, 165, 166
caption-valign-bottom (float) .....	168
corresp-mark (Generic) .....	78
corresp-sep (Generic) .....	78
counted-name-sep (Generic) .....	78
counted-name-sep (Heading) .....	105
<b>D</b>	
default-label (List) .....	192
<b>E</b>	
enum-type (List) .....	193
extended (Heading) .....	101
extended-heading (Heading) .....	101
<b>F</b>	
float-number (float) .....	154, 162
float-render (float) .....	152, 164, 170, 170, 171, 174
float-skip-bottom (float) .....	149, 151
float-skip-top (float) .....	149, 151
<b>H</b>	
heading-block (Heading) .....	101
heading-par (Heading) .....	100
<b>I</b>	
indent (Heading) .....	102, 102
indent (List) .....	184, 185, 191, 195
initials-format (Generic) .....	70, 76
initials-period (Generic) .....	76
initials-sep (Generic) .....	76
int-margin-bottom (List) .....	190, 191
int-margin-top (List) .....	190, 191
interline-para (Heading) .....	100
interline-para-sep (Heading) .....	100
intext-skip-bottom (float) .....	149, 151
intext-skip-top (float) .....	149, 151
item-format (List) .....	185
item-offset (List) .....	184, 185
item-sep (List) .....	183, 184
<b>K</b>	
keywords-sep (Generic) .....	78
<b>L</b>	
label (List) .....	184, 185, 194
label-align (List) .....	185, 185
label-box (List) .....	185, 185, 195
label-face (List) .....	185, 185, 195
label-format (List) .....	185, 185
label-grow (List) .....	195
label-growth (List) .....	195
label-pos (float) .....	154, 166
label-sep (List) .....	184, 185, 185, 190
labeled-meta-[ccCurInfix]-format (Generic) .....	72
labeled-meta-[infix]-after (Generic) .....	81
labeled-meta-[infix]-before (Generic) .....	81
labeled-meta-[infix]-face (Generic) .....	81
labeled-meta-[infix]-format (Generic) .....	80
labeled-meta-[infix]-label-face (Generic) .....	81
labeled-meta-[infix]-label-format (Generic) .....	80
labeled-meta-[infix]-label-sep (Generic) .....	81
labeled-meta-after (Generic) .....	81
labeled-meta-before (Generic) .....	81
labeled-meta-before-[infix]-label (Generic) .....	80

- labeled-meta-face (Generic) ..... 81  
 labeled-meta-format (Generic) ..... 72, 80  
 labeled-meta-label-face (Generic) ..... 81, 81  
 labeled-meta-label-format (Generic) ..... 80  
 labeled-meta-label-sep (Generic) ..... 81  
 legend-face (float) ..... 152, 152  
 list-of-after-entry (float) ..... 156, 161  
 list-of-before-entry (float) ..... 155, 161  
 list-of-block (float) ..... 155, 161  
 list-of-caption-sep (float) ..... 155  
 list-of-indent (float) ..... 155  
 list-of-margin-left (float) ..... 155, 155  
 list-of-margin-right (float) ..... 155, 155  
 list-of-number-align (float) ..... 155  
 list-of-number-face (float) ..... 155, 155  
 list-of-number-format (float) ..... 155  
 list-of-number-sep (float) ..... 155, 155  
 list-of-page-face (float) ..... 160  
 list-of-page-sep (float) ..... 154, 155  
 list-of-parfillskip (float) ..... 155, 155  
 list-type (List) ..... 186, 189
- M**
- main-title-format (Heading) ..... 101  
 margin-bottom (List) ..... 183, 191  
 margin-inner (float) ..... 151, 163  
 margin-left (Heading) ..... 102, 102  
 margin-left (List) ..... 183, 185, 190, 191, 195  
 margin-left (float) ..... 151, 163  
 margin-outer (float) ..... 151, 163  
 margin-right (Heading) ..... 102  
 margin-right (List) ..... 183, 190  
 margin-right (float) ..... 151, 163  
 margin-top (List) ..... 183, 191  
 max-label-width (List) ..... 195
- N**
- name-and (Generic) ..... 78  
 name-et-al (Generic) ..... 78  
 name-sep (Generic) ..... 78  
 no-BM (Heading) ..... 103  
 no-toc (Heading) ..... 103  
 number-align (Heading) ..... 102  
 number-face (Heading) ..... 102  
 number-face (float) ..... 152, 154  
 number-format (Heading) ..... 102  
 number-sep (Heading) ..... 102, 102  
 number-sep (float) ..... 152, 154  
 number-width (Heading) ..... 102  
 number-width-level-max (List) ..... 191  
 number-width-max (List) ..... 191  
 numbering (Heading) ..... 103  
 numbering (float) ..... 153, 154
- O**
- orcid-link (Heading) ..... 105, 105  
 output-intent (titlepage) ..... 143
- P**
- par-fill-skip (List) ..... 184, 184  
 par-indent (List) ..... 184, 184
- par-skip (List) ..... 183, 184, 184  
 prev-margin-left (List) ..... 183, 189, 191  
 prev-margin-right (List) ..... 189
- Q**
- quote-block-format (Heading) ..... 100  
 quote-face (Heading) ..... 100  
 quote-source-face (Heading) ..... 100
- R**
- role-block-cite-format (Generic) ..... 71, 77  
 role-block-correspondence-format (Generic) ..... 72, 78  
 role-block-pdfinfo-format (Generic) ..... 71, 77  
 role-block-print-format (Generic) ..... 71, 77  
 role-block-short-cite-format (Generic) ..... 71, 77  
 role-cite-name-format (Generic) ..... 70, 77  
 role-correspondence-as-format (Generic) ..... 71, 77  
 role-full-name-format (Generic) ..... 69, 76  
 role-pdfinfo-name-format (Generic) ..... 70, 77  
 role-short-cite-name-format (Generic) ..... 70, 77  
 running-heading (Heading) ..... 103  
 running-level (Heading) ..... 103
- S**
- smash (float) ..... 171  
 source-face (float) ..... 152, 152  
 sub-float-sep (float) ..... 151, 163  
 sub-number (float) ..... 154, 163  
 sub-number-after (float) ..... 163  
 sub-number-before (float) ..... 154, 163  
 sub-number-face (float) ..... 154, 163  
 sub-number-format (float) ..... 154, 163  
 sub-number-sep (float) ..... 154, 154  
 sub-number-style (float) ..... 154, 163  
 subcaption-add-sep-bottom (float) ..... 153, 166  
 subcaption-add-sep-top (float) ..... 153, 166  
 subcaption-bottom (float) ..... 153, 165, 166  
 subcaption-face (float) ..... 152, 164, 166  
 subcaption-face-bottom (float) ..... 153, 164, 166  
 subcaption-face-top (float) ..... 153, 164, 166  
 subcaption-sep-bottom (float) ..... 166  
 subcaption-sep-top (float) ..... 153, 166  
 subcaption-top (float) ..... 153, 165, 166  
 subcaption-valign-bottom (float) ..... 153, 168  
 subcaption-valign-top (float) ..... 153, 168  
 subfloat-content (float) ..... 151, 158, 169  
 subfloat-render (float) ..... 152, 164, 170, 170, 174  
 subfloat-same-height (float) ..... 169  
 sublabel-pos (float) ..... 154, 166  
 subtitle-face (Heading) ..... 100
- T**
- title-face (Heading) ..... 100, 102  
 toc-after-entry (Heading) ..... 104  
 toc-before-entry (Heading) ..... 104  
 toc-format (Heading) ..... 105  
 toc-hang-number (Heading) ..... 105  
 toc-hook (Heading) ..... 102  
 toc-indent (Heading) ..... 103  
 toc-level (Heading) ..... 104  
 toc-margin-bottom (Heading) ..... 103, 104



toc-margin-left (Heading)	<a href="#">103</a> , <a href="#">104</a>	toc-number-width (Heading)	<a href="#">103</a>
toc-margin-right (Heading)	<a href="#">103</a> , <a href="#">104</a>	toc-page-face (Heading)	<a href="#">104</a> , <a href="#">104</a>
toc-margin-top (Heading)	<a href="#">103</a> , <a href="#">104</a>	toc-page-format (Heading)	<a href="#">104</a> , <a href="#">105</a>
toc-number-align (Heading)	<a href="#">104</a>	toc-page-sep (Heading)	<a href="#">104</a> , <a href="#">104</a>
toc-number-face (Heading)	<a href="#">104</a> , <a href="#">104</a>	toc-title-face (Heading)	<a href="#">103</a> , <a href="#">105</a>
toc-number-format (Heading)	<a href="#">104</a>		V
toc-number-sep (Heading)	<a href="#">104</a> , <a href="#">104</a>	vertical-align (float)	<a href="#">171</a>

## Hook Index

<b>B</b>	
before-hook-chapter .....	110
begin-hook .....	100
<b>C</b>	
cc/headings/[level]/after .....	<u>92</u>
cc/headings/[level]/before .....	<u>92</u>
cc/headings/[level]/print/before .....	<u>92</u>
cc/headings/[level]/run/after .....	<u>92</u>
cc/headings/[level]/toc/after .....	<u>92</u>
cc/headings/[level]/toc/before .....	<u>92</u>
cca/document/begin .....	<u>63</u>
cct/document/meta .....	<u>125</u> , <u>128</u>
cct/maketitle/after .....	<u>128</u>
cct/maketitle/before .....	<u>128</u> , <u>129</u>
<b>E</b>	
env/heading/[level]/begin .....	<u>92</u> , <u>97</u>
env/meta/begin .....	<u>128</u>

## Tag Index

<b>A</b>	
Aside .....	157
<b>C</b>	
Caption .....	167
<b>D</b>	
Div .....	157
<b>F</b>	
Figure .....	170
FootnoteMark .....	112, 113
FootnoteText .....	113
<b>H</b>	
H .....	101
H1 .....	139
Hn .....	101
<b>L</b>	
L .....	182, 183
Lbl .....	105, 112, 185
LBody .....	183
LI .....	184
<b>N</b>	
Note .....	113
<b>P</b>	
P .....	105, 144
<b>R</b>	
Reference .....	105, 112
<b>S</b>	
Sect .....	97
Span .....	105
<b>T</b>	
Title .....	139, 144
TOCI .....	105

# Attribute Index

<b>B</b>		<b>F</b>	
break-caption (float)	<u>156</u>	float-pos (float)	<u>156</u>
<b>C</b>		<b>N</b>	
class (Heading)	<u>84</u>	no-same-height (float)	<u>156</u>
class (float)	<u>156</u>		
<b>D</b>		<b>O</b>	
debug (float)	<u>156</u>	orientation (float)	<u>156</u>