

# The cocotex.dtx Package

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

Version 0.4.1

(2024/03/23)

Lupino

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

# Table of contents

---

<b>Introduction</b>	<b>vii</b>
1 Basic concepts . . . . .	vii
2 Flow of macro definitions and their expansions in modules that use the Property and Component mechanism . . . . .	vii

<b>Modul 1 cocotex.dtx</b>	<b>3</b>
1 Hard-coded requirements . . . . .	3
2 Class Options . . . . .	3
3 Class Hook . . . . .	4
4 Internal Requirement . . . . .	5
5 Loading and Adjusting Underlying DocumentClass . . . . .	5
5.1 General Typography . . . . .	5
6 Loading other CoCoT <sub>E</sub> X Modules . . . . .	6
6.1 coco-accessibility . . . . .	6
6.2 coco-script . . . . .	6
6.3 coco-headings . . . . .	6
6.4 coco-floats . . . . .	7
6.5 coco-title . . . . .	7
6.6 coco-notes . . . . .	7
7 Further Hard Dependencies . . . . .	7
7.1 Index . . . . .	7
7.2 Hyperref . . . . .	7
8 End of Dcument Class Hook . . . . .	8

## Part I: Core Functions

<b>Modul 2 coco-kernel.dtx</b>	<b>11</b>
1 Preamble . . . . .	11
1.1 Hard dependencies . . . . .	11
1.2 Package Options . . . . .	11
2 Exception handlers . . . . .	11
3 Global Switches . . . . .	12
4 Containers . . . . .	13
5 Components . . . . .	15
5.1 Simple Components . . . . .	15
5.2 Counted Components . . . . .	18
6 Hooks . . . . .	23
7 Properties . . . . .	23
7.1 Setting Properties . . . . .	23
7.2 Using Properties . . . . .	24
7.3 Processing Instructions . . . . .	26
7.4 Property Conditionals . . . . .	26
8 Helper macros . . . . .	26
8.1 Handling of Optional Arguments . . . . .	26
8.2 Iterators . . . . .	26
8.3 Attributes . . . . .	27

8.4	Style Classes	29
-----	---------------	----

### Modul 3 coco-common.dtx 31

1	Package options	31
1.1	Accessibility Features	31
2	Commonly Used Low-Level Macros and Registers	32
2.1	Hard Dependencies	32
2.2	Common Variables	32
2.3	Helper macros	33
2.4	Masks	33
2.5	Arithmetics	34
2.6	Determine actual page number	35
3	Re-Thinking L <sup>A</sup> T <sub>E</sub> X Core Functions	36
3.1	Keeping .aux-Files Up-to-Date	36
3.2	Content lists	36
3.3	Indentation and Left Margins of Potentially Numbered Items	38
3.4	Labelling and Cross referencing	42
3.5	Linguistic Name generation and selection	43
3.6	Link Generation	44

### Modul 4 coco-accessibility.dtx 45

1	LaTeX code	45
1.1	General Processing	45
1.2	Activating and Deactivating Accessibility Features	46
1.3	Accessibility-specific additions	46
1.4	Generic Macro to Declare Accessibility Features	48
1.5	Lua injection	51
1.6	Hyperlink handling	51
1.7	Tagging Page Styles as Artifacts	53
1.8	generic artifacts	53
1.9	Tagging for Floats	53
1.10	Transformation of Typographic Unicode characters	54
1.11	Automatic PDF Tagging	55
2	Default Role Mapping	55
3	Lua code	55
3.1	Local Variables and Tables	55
3.2	Meta Data Extraction	56
3.3	Public Methods	56

### Modul 5 coco-meta.dtx 59

1	Counted Container Handlers	59
1.1	Generic Blocks	59
1.2	Contributor Roles	60
2	Labeled Components	62
3	Meta Data Rolemaps for Tagged PDFs	62
4	Common Meta Data	63
4.1	Affiliations	64

## Part II: Document Level Structures

### Modul 6 coco-headings.dtx 69

1	Facility for declaring heading levels and their layouts	69
1.1	Initializers for New Heading Levels	76
1.2	Initializers for Instances of Heading Levels	77

2	Externalisation of Heading Components . . . . .	77
2.1	Common Stuff . . . . .	77
2.2	Table of Contents Entry . . . . .	78
2.3	Facility to create the running title macros . . . . .	79
2.4	Facility to create PDF bookmarks . . . . .	79
3	Rendering the Headings . . . . .	80
3.1	Inline Headings . . . . .	80
3.2	Block Headings . . . . .	80
4	The Heading environment . . . . .	80
4.1	Environment Macros . . . . .	80
4.2	Content Handlers . . . . .	82
5	Defaults . . . . .	83
6	Miscellaneous . . . . .	86
6.1	Alternative paragraph separation . . . . .	86

## **Modul 7    coco-notes.dtx    87**

## **Modul 8    coco-script.dtx    93**

1	Default fallback font . . . . .	93
2	Generic Fonts Declaration Mechanism . . . . .	94
3	Predefined script systems . . . . .	95
3.1	Support for Armenian script . . . . .	95
3.2	Support for Chinese script . . . . .	95
3.3	Support for Japanese script . . . . .	95
3.4	Support for Hebrew script . . . . .	95
3.5	Support for Arabic script . . . . .	96
3.6	Support for Greek script . . . . .	96
3.7	Support for Syrian script . . . . .	96
3.8	Support for medieval scripts and special characters . . . . .	97

## **Modul 9    coco-title.dtx    99**

1	Top-Level Interface . . . . .	99
2	Processing of PDF Meta Data . . . . .	101
2.1	Processing of the Document's Title . . . . .	102
2.2	Processing of the Document's Author . . . . .	102
2.3	Processing of the PDF's Creator, Producer, and Keywords Meta Data . . . . .	103
2.4	Including the XMP Meta Data . . . . .	103
3	Intermediate Level Interfaces . . . . .	103
3.1	Funds, Grants, and Supporters . . . . .	104
3.2	Simple Component Declarations . . . . .	105
4	Default Settings . . . . .	107
5	Accessibility Features . . . . .	112
5.1	Output Intent and ICC Profiles . . . . .	112
5.2	Encoding of the PDF-A Conformance . . . . .	113
5.3	Titlepage Specific Role Maps . . . . .	114

## **Modul 10    coco-floats.dtx    115**

1	Package Setup . . . . .	115
1.1	Hard requirements . . . . .	115
1.2	Document Class Option overrides . . . . .	115
2	.clo . . . . .	115
2.1	Internal registers . . . . .	116
2.2	AtBeginDocument hook . . . . .	117
3	Internal macros . . . . .	117
3.1	Generic resetter . . . . .	117
3.2	Internal macros that handle Attributes . . . . .	118

4	Float Container and Component Declarations . . . . .	120
5	Label and Referencing mechanisms . . . . .	124
5.1	Generation of Number Components . . . . .	124
5.2	Generation of L <sup>A</sup> T <sub>E</sub> X Labels . . . . .	124
6	Processing the Float . . . . .	125
6.1	Common Float and Sub-Float Environments . . . . .	125
6.2	Processing the Contents of the Float Environment . . . . .	127
6.3	Caption mechanism . . . . .	127
7	Handlers for different float types . . . . .	131
7.1	Handlers for generic floats . . . . .	131
7.2	Handlers for figures . . . . .	131
7.3	Handlers for tables . . . . .	132
7.4	Helpers . . . . .	134
8	Default Settings . . . . .	134
<b>Modul 11 coco-frame.dtx</b>		<b>139</b>
1	Top-Level Interface . . . . .	139
2	Cropmark printer . . . . .	139
<b>Modul 12 coco-lists.dtx</b>		<b>143</b>
1	Preamble . . . . .	143
1.1	Package Options . . . . .	143
2	The List Container . . . . .	144
3	Declaring List Types . . . . .	146
4	Declare Lists . . . . .	146
4.1	The List Environment . . . . .	148
4.2	Unpacking the List Type-Specific Handlers . . . . .	150
5	Default List Types . . . . .	151
5.1	Unnumbered Lists . . . . .	151
5.2	Numbered Lists . . . . .	152
5.3	Description Lists . . . . .	153
5.4	Replacing L <sup>A</sup> T <sub>E</sub> X's Default Lists . . . . .	155



# Introduction

---

## 1 Basic concepts

The core concept of the CoCoTeX Framework is the strict separation between document specific information bearing units and publisher specific layout and rendering instructions to a degree that is far more versatile and delicate than L<sup>A</sup>T<sub>E</sub>X's usual distinction between form and content.

The basic data type in the Framework is the **Container**. On the end-user level, this is virtually always a L<sup>A</sup>T<sub>E</sub>X environment that contain a specific set of macros used to store the atomic units of information. Those macros and their contents are called **Components**.

The instructions on how those Components are to be processed and ultimately rendered are called **Properties**.

## 2 Flow of macro definitions and their expansions in modules that use the Property and Component mechanism

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

Modules, that utilize the Property and Component mechanisms, define a *Declare macro*. This Declare macro is basically a constructor for a new L<sup>A</sup>T<sub>E</sub>X environment which should share some common *Properties* and *Components* with other environments that are defined with the same Declare macro. Modules, therefore, constitute what in other programming languages may be referred to as *Namespaces*.

The purpose of the Declare macro is

1. to define a L<sup>A</sup>T<sub>E</sub>X environment to be used in tex documents,
2. to define the Component macros available and allowed within that environment
3. to define the available Properties used to determine the appearance of the environment's content in the final render.
4. to define the processing of the information specific to each instance of the environment.

Within the body of the Declare macro's definition, a Use macro is defined which determines the Namespace-specific processing of an environment's contents. This macro is (usually) expanded at the **\end** of the declared environment. The Use macro is where the actual processing of an environment's contents takes place. Since it is part of the body of the Declare macro, each environment declared with this Declare macro defines it's own Use macro.

The Declare macro usually has at least two arguments: one argument to give a *name* to the soon-to-be-defined environment, and a second one to define the Properties *specific* to that environment *on top of* the Namespace's default Properties. Some environments may also have a Parent which causes Properties cascade across different inter-dependend environments.

Within the tex-document, whenever an environment is used, the flow is as follows:

1. *store* the contents of all Components used within the environment in internal, locally defined, tex macros
2. expand the property lists:

- (a) expand the Default Properties of the Namespace
  - (b) If necessary, expand the specific Properties of the parent environment (overwriting the default properties of the same name). This step may occur recursively for each of the parent's own parents.
  - (c) expand the Specific Properties of the Environment itself.
- 3. Expand the Use-Macro
  - (a) Process the components, depending on contents, presence, or absence of Components alter other Components or trigger property manipulations, etc.
  - (b) Calculate the final states of variable properties (in dependency on the available components, other properties or global parameters)
  - (c) Print the overall result of those calculations.



One more driver function

```
24 %<*driver>
```

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

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

End driver function

```
26 %</driver>
```



## Modul 1

# cocotex.dtx

---

This is the main class file for the CoCoT<sub>E</sub>X L<sup>A</sup>T<sub>E</sub>X package.

```
24 %<class>
```

### File Preamble

```
25 %%
26 %% Common document class for \textit{xerif} projects.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesClass{cocotex}
34 [2024/03/23 0.4.1 cocotex]
```

## 1 Hard-coded requirements

```
35 \RequirePackage{kvoptions-patch}
36 \RequirePackage{xkeyval}
```

## 2 Class Options

Passing options down to the L<sup>A</sup>T<sub>E</sub>X standard packages

```
37 \DeclareOptionX{main}{\PassOptionsToPackage{\CurrentOption}{babel}}
38 \DeclareOption{es-noindentfirst}{\PassOptionsToPackage{es-noindentfirst}{babel}}
39 \DeclareOption{es-noshorthands}{\PassOptionsToPackage{es-noshorthands}{babel}}
40 \PassOptionsToPackage{shorthands=off}{babel}
```

The option **pubtype** (short for “publication type”) has possible four 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**.

```

41 \newif\ifcollection \collectionfalse
42 \newif\ifarticle \articlefalse
43 \newif\ifmonograph \monographfalse
44 \newif\ifjournal \journalfalse
45 \define@choicekey{cocotex.cls}{pubtype}[\cc@pub@type\nr]{collection,article,journal,mono}{%
46   \ifcase\nr\relax% collection
47     \global\collectiontrue
48   \or% article
49     \global\articletrue
50   \or% journal
51     \global\journaltrue
52   \else% monograph
53     \global\monographtrue
54   \fi
55 }
56 \DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{article}}
57 \DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{book}}

```

Passing options down to various CoCoT<sub>E</sub>X modules:

```

58 \DeclareOptionX{debug}{\PassOptionsToPackage{\CurrentOption}{coco-kernel}}
59 \DeclareOptionX{ally}{\PassOptionsToPackage{init}{coco-accessibility}}
60 \DeclareOptionX{lang-id}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
61 \DeclareOptionX{nodetree}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
62 \DeclareOptionX{showspaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
63 \DeclareOptionX{no-spaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
64 \DeclareOptionX{no-paras}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
65 \DeclareOptionX{no-compress}{\let\cc@no@pdf@compression\relax}
66 \DeclareOptionX{color-enc}{\PassOptionsToPackage{\CurrentOption}{coco-common}}
67 \DeclareOptionX{usescript}{\PassOptionsToPackage{\CurrentOption}{coco-script}}
68 \DeclareOptionX{nofigs}{\PassOptionsToPackage{\CurrentOption}{coco-floats}}
69 \DeclareOptionX{ennotoc}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
70 \DeclareOptionX{endnotes}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
71 \DeclareOptionX{resetnotesperchapter}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
72 \DeclareOptionX{endnotesperchapter}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
73 \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). 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*.

Note that this hook is temporary. As soon as all legacy styles are adjusted to the new macro names, this hook will be removed!

```

74 \def\ccAfterClassHook{}

```

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

Note that this hook is temporary. As soon as all legacy styles are adjusted to the new macro names, this hook will be removed!

```
75 \def\ccToggleCountedConditionalsHook{}%
```

## 4 Internal Requirement

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

## 5 Loading and Adjusting Underlying DocumentClass

All publication types supported by CoCoT<sub>E</sub>X are based on one of L<sup>A</sup>T<sub>E</sub>X's default classes `book` or `article`:

```
77 \ifarticle
78   \LoadClass[10pt,a4paper]{article}
79 \else
80   \LoadClass[10pt,a4paper]{book}
81 \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.

```
82 \voffset-1in\relax
83 \hoffset-1in\relax
```

Automatted typesetting needs some room to play

```
84 \emergencystretch=2em
```

and strong restrictions:

```
85 \frenchspacing
86 \clubpenalty10000
87 \widowpenalty10000
```

### Empty Pagestyle

Page style without any headers or footers

```
88 \def\ps@empty{%
89   \let\@oddhead\@empty
90   \let\@evenhead\@empty
91   \let\@oddfoot\@empty
92   \let\@evenfoot\@empty
93 }
```

### Vacancy Pages

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

```

94 \def\cleardoublepage{\clearpage\if@twoside \ifodd\c@page\else
95   \hbox{}\thispagestyle{empty}\newpage\if@twocolumn\hbox{}\newpage\fi\fi}

```

### Book Parts

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

```

96 \ifarticle\else
97   \newif\if@frontmatter \@frontmatterfalse
98   \renewcommand\frontmatter{%
99     \cleardoublepage
100    \@mainmatterfalse
101    \@frontmattertrue
102    \pagenumbering{arabic}}
103   \renewcommand\mainmatter{%
104     \cleardoublepage
105     \@frontmatterfalse
106     \@mainmattertrue}
107   \renewcommand\backmatter{%
108     \cleardoublepage
109     \@mainmatterfalse
110     \@frontmatterfalse}
111 \fi

```

#### WARNING!

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

```

112 \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.

```

113 \RequirePackage{coco-accessibility}

```

### 6.2 coco-script

Inclusion of the script module which also loads the babel package

```

114 \ifLuaTeX
115 \RequirePackage{coco-script}
116 \else
117 \RequirePackage{babel}
118 \fi

```

### 6.3 coco-headings

```
119 \RequirePackage{coco-headings}
```

## 6.4 coco-floats

Inclusion of the float module

```
120 \RequirePackage{coco-floats}
```

## 6.5 coco-title

Inclusion of the title page module

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

## 6.6 coco-notes

Inclusion of the end-/footnotes module

```
122 \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:

```
123 \RequirePackage{index}
124 \makeindex
```

## 7.2 Hyperref

```
125 \RequirePackage{hyperref}
```

Finally, some `hyperref` settings (TODO: check, which of those are better placed inside the local publisher's styles)

```
126 \hypersetup{%
```

first, we want links to be breakable

```
127 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.

```
128 ,linktoc=none%
```

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

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

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

```
130 ,pdfencoding=auto%
```

Bookmarks are numbered by default.

```
131 ,bookmarksnumbered=true%
132 }
```

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

```
133 \ifx\cc@no@pdf@compression\relax
134 \ifx\pdfobjcompresslevel\@undefined
135 \edef\pdfobjcompresslevel{\pdfvariable objcompresslevel}%
136 \fi
137 \pdfcompresslevel=0
138 \pdfobjcompresslevel=0
139 \fi
```

## 8 End of Document Class Hook

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

```
140 \ccAfterClassHook
```

```
141 %</class>
```



## **Part I**

# **Core Functions**



## Modul 2

# coco-kernel.dtx

---

This file provides the object-oriented interfaces for all other CoCoT<sub>E</sub>X modules.

```
24 %<*kernel>
```

## 1 Preamble

```
25 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
26 \ProvidesPackage{coco-kernel}
27 [2024/03/23 0.4.1 cocotex kernel]
```

### 1.1 Hard dependencies

```
28 \RequirePackage{kvoptions-patch}
29 \RequirePackage{xkeyval}
30 \RequirePackage{etoolbox}
```

### 1.2 Package Options

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

```
31 \newif\if@cc@debug \@cc@debugfalse
32 \DeclareOption{debug}{\global\@cc@debugtrue}%
```

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

```
33 \DeclareOptionX{prefix}[]{\gdef\cc@prefix{#1}}%
34 \ProcessOptionsX
```

## 2 Exception handlers

The CoCoT<sub>E</sub>X 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.

```
35 \def\ccPackageError#1#2#3#4{%
```

```

36 \GenericError{%
37   (#1)\@spaces\@spaces\@spaces\@spaces
38 }{%
39   [CoCoTeX #1 #2 Error] #3%
40 }{}{#4}%
41 }

```

**\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.

```

42 \def\ccPackageWarning#1#2#3{%
43   \GenericWarning{%
44     (#1)\@spaces\@spaces\@spaces\@spaces
45   }{%
46     [CoCoTeX #1 \if!#2!\else#2 \fi Warning] #3%
47   }%
48 }

```

**\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.

```

49 \def\ccPackageInfo#1#2#3{%
50   \GenericInfo{%
51     (#1)\@spaces\@spaces\@spaces\@spaces
52   }{%
53     [CoCoTeX #1\if!#2!\else\space#2\fi] #3%
54   }%
55 }

```

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

**\ccKernelDebugMsg** prints a debug message if and only if the **debug** package option is set.

```

56 \def\ccKernelDebugMsg#1{\ifcc@debug\message{[CoCo Kernel Debug]\space\space#1^^}\fi}

```

## 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 **t**ranspect, 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.

```

57 \ifx\ccPrefix\undefined\edef\ccPrefix{\cc@prefix}\fi
58 \ccPackageInfo{Kernel}{Info}{The macro prefix is now '\ccPrefix'.}

```

**\ifcc@is@final** is a boolean switch that indicates whether or not a process is final. This is mainly used in the accessibility module where it matters if a macro is actually used to print stuff, or if it is just processed.

```

59 \newif\ifcc@is@final \cc@is@finalfalse
60 \AtBeginDocument{\cc@is@finaltrue}

```



**\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*. With the optional argument **##1** you can override 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 beginning of the environment, **##3** for the stuff done at the end.

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

```
78 \def\ccDeclareEnv{\ifnextchar [{\cc@declare@env}{\cc@declare@env[#1]}}%
79 \def\cc@declare@env[#1]##2##3{%
80 \csgdef{\ccPrefix #1}{\global\let\reserved@cont\cc@cur@cont\def\cc@cur@cont{#1}##2}%
81 \csgdef{end\ccPrefix #1}{##3}\global\let\cc@cur@cont\reserved@cont}%
```

```
82 \def\x{%
83 #2%
84 }%
85 \expandafter\x\endgroup
86 }
87 %\onlypreamble\ccDeclareContainer
```

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

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

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

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

**\ccEvalType** calls the Declaration list for data Type #2. With optional #1 the Container Class can be overridden.

```
90 \def\ccEvalType{\cc@opt@curcont\cc@eval@type}
91 \def\cc@eval@type[#1]#2{%
92 \expandafter\ifx\csname cc@type@#2@#1\endcsname\relax
93 \ccPackageError{Kernel}{Class}
94 {Data Type #2 in Container #1 undefined!}
95 {You try to evaluate a data type '#2' from container '#1', but that data type has not been
96 declared.}%
97 \else
98 \ccKernelDebugMsg{Evaluating cc@type@#2@#1:^^J \csmeaning{\cc@type@#2@#1}}%
99 \csname cc@type@#2@#1\endcsname
100 \fi
101 }
```

**\ccCheckParent** checks if a Container #1 is declared so that another container #2 can inherit.

```
101 \def\ccCheckParent#1#2{%
102 \expandafter\ifx\csname cc@container@#1\endcsname\relax
103 \ccPackageError{Kernel}{Class}
104 {Parent Container '#1' undeclared}
105 {You tried to make a Container named '#2' inherit from a Container named '#1', but a
106 Container with that name does not exist.\MessageBreak
107 Please make sure that parent Containers are declared before their descendents.}%
108 \else
109 \csgdef{\cc@parent@#2}{#1}%
110 \fi
111 }
```

`\cc@inherit` is the low-level inherit function. #1 is a comma-separated list of things to be inherited, and #2 is the Container-list that should be inherited from, and #3 is the name of the descending Container.

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

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

```
112 \def\cc@parse@inherit #1,#2,\@nil #3,#4,\@nil #5\@nil{%
113   \let\next\relax
114   \if!#1!\else
115     \if!#3!\else
116       \cc@do@inherit{#1}{#3}{#5}%
117       \def\argii{#2}\def\argiv{#4}%
118       \ifx\argii\@empty
119         \ifx\argiv\@empty\else
120           \def\next{\cc@parse@inherit #1,,\@nil #4,\@nil #5\@nil}%
121           \fi
122         \else
123           \ifx\argiv\@empty
124             \def\next{\cc@parse@inherit #2,\@nil #3,,\@nil #5\@nil}%
125             \else
126               \def\next{%
127                 \@cc@parse@inherit #1,,\@nil #4,\@nil #5\@nil
128                 \@cc@parse@inherit #2,\@nil #3,#4,\@nil #5\@nil
129               }%
130             \fi\fi\fi\fi
131   \next}
```

Ultimately, this function is called for each Type–Container combination invoked by the `\ccInherit` macro.

```
132 \def\cc@do@inherit#1#2#3{%
133   \ccKernelDebugMsg{#3 inherits #1 from #2.}%
134   \ccCheckParent{#2}{#3}%
135   \expandafter\ifx\csname cc@type@#1@#2\endcsname\relax
136     \ccPackageError{Kernel}{Type}{Type ‘#1’ was not declared for
      Container ‘#2’}.}%
137   \else
138     \edef\x{\noexpand\csgappto{cc@type@#1@#3}}%
139     \expandafter\x\expandafter{\csname cc@type@#1@#2\endcsname}%
140     \ccKernelDebugMsg{value cc@type@#1@#3:^^J \expandafter\meaning\csname cc@type@#1@#3\endcsname}%
141   \fi
142 }
```

## 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 simple component macros.

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

```

143 \def\ccDeclareComponent{\cc@opt@second\cc@declare@comp}
144 \def\cc@declare@comp[#1]#2#3#4{%
145   \ltx@LocalExpandAfter\global\expandafter\let\csname cc@\cc@cur@cont @#1\endcsname\relax
146   \expandafter\long\expandafter\def\csname \ccPrefix#2\endcsname##1{%
147     #3\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#1\endcsname{##1}\ignorespaces
148     #4}%
149 }
```

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

```

149 \def\ccDeclareGlobalComponent{\cc@opt@empty\cc@declare@global@comp}%
150 \def\cc@declare@global@comp[#1]#2{%
151   \ccDeclareComponent{#2}{\expandafter\global}{}}%
152   \if!#1!\else\csname \ccPrefix #2\endcsname{#1}\fi%
153 }
```

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 which takes two arguments: #1 is the name of the Component, #2 is the value. This macro checks whether an Component of name #1 has actually been declared and does so, if not.

**\cc@counted@comp@scheme** gives the scheme how counted components are defined internally. It consumes one argument #1, which contains the name of the Counted Component.

```

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

**\ccComponent** This is the preferred way to fill a Component with content. #1 is the Component's name, #2 is the value.

```

155 \long\protected\def\ccComponent#1#2{%
156   \ifx\cc@is@counted\relax
157     \ifcsdef{cc@\cc@cur@cont @#1}{}
158     {\cc@def@counted@comp{\cc@counted@comp@scheme{#1}}{#1}}}%
159     \csgdef{cc@\cc@cur@cont @\cc@counted@comp@scheme{#1}}{#2}%
160   \else
161     \ifcsdef{cc@\cc@cur@cont @#1}{}{\ccDeclareComponent{#1}}}%
162     \csdef{cc@\cc@cur@cont @#1}{#2}%
163   \fi
164 }
```

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

```

165 \long\protected\def\ccComponentEA#1#2{%
166   \def\x{\ccComponent{#1}}\expandafter\x\expandafter{#2}%
167 }
```

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

```

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

**\ccdefFromComp** is a high level command to store the value of a Component #2 into a CS token #1.

```

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



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

```
170 \def\ccgdefFromComp#1#2{\cc@store@comp{x}#1{#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.

```
171 \def\strip@longprefix#1\long macro:->#2{#2}
172 \long\def\cc@store@comp#1#2#3{%
173   \edef\@tempa{\expandonce{\csname protected@#1def\endcsname}\noexpand#2}%
174   \protected@edef\@tempb{\csname cc@\cc@cur@cont @#3\endcsname}%
175   \ifx\@tempb\relax
176     \let#2\relax
177   \else
178     \expandafter\@tempa\expandafter{\@tempb}%
179   \fi
180 }
```

`\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.

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

`\ccGetComp`, `\ccGetComp*` is a high 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 auto-tagging for that Component.

```
182 \def\ccGetComp{\@ifstar\cc@sget@comp\cc@get@comp}
183 \def\cc@get@comp#1{\ccWhenComp{#1}{%
184   \ccWhenAlly{\ccaStructStart{Para}}%
185   \ccUseComp{#1}%
186   \ccWhenAlly{\ccaStructEnd{Para}}%
187   \par}}
188 \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.

```
189 \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. #2 is code that is expanded when the Component #1 is used in a container (empty or non-empty).

```
190 \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. #2 is the code that is expanded when a Container #1 is *not* used in a Container (neither empty nor non-empty).

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

`\ccIfCompFrom` 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.

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

`\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.

```
193 \long\def\cc@long@empty{}
194 \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.

```
195 \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 (=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.

Optional #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.

```
196 \def\cc@check@empty{\cc@opt@empty\@cc@check@empty}%
197 \def\@cc@check@empty[#1]#2#3#4{%
198   \ccIfComp{#4#3}
199   {\ccIfCompEmpty{#4#3}
200    {\expandafter\global\expandafter\let\csname cc@#2@#4#3\endcsname\relax}
201    {}}
202   {\ccIfComp{#1#3}
203    {\expandafter\expandafter\expandafter\let\expandafter\csname cc@#2@#4#3\expandafter\
      endcsname\csname cc@#2@#1#3\endcsname}
204    {}}}
```

## 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 #1 and the body #2.

```
205 \def\ccDeclareComponentGroup#1#2{%
206   \csnumgdef{cc#1Cnt}{\z@}%
207   \csdef{ccPrefix#1}{\cc@opt@empty{\csname cc@group@#1\endcsname}}%
208   \csdef{cc@group@#1}[##1]{%
209     \def\cc@cnt@grp{cc#1}%
210     \csxdef{cc#1Cnt}{\expandafter\the\expandafter\numexpr\csname cc#1Cnt\endcsname+\@ne\relax}%
211     \if!##1!\else\csgdef{cc@\cc@cur@cont @#1-\csname cc#1Cnt\endcsname @attrs}{##1}\fi
212     #2%
213     \csname @#1@hook\endcsname
214   }%
215   \csdef{end\ccPrefix#1}{\ccToggleCountedConditionals\csuse{cc@compose@group@#1}}}%
216 }
```

**\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".

```
217 \def\ccDeclareGroupHandler#1#2{%
218   \ifcsdef{cc@group@#1}
219     {\ifcsdef{cc@compose@group@#1}
220       {\csgappto{cc@compose@group@#1}{#2}}
221       {\csgdef{cc@compose@group@#1}{#2}}}
222   {\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.}}%
223 }
```

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

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

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

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

**\ccUsePropFrom** picks a specific Property of a group.

```
226 \def\ccUsePropFrom#1#2#3{%
227   \begingroup
228     \@tempcnta\numexpr#2\relax
229     \letcs\ccTotalCount{cc#1Cnt}%
230     \def\cc@cnt@grp{cc#1}%
231     \ccToggleCountedConditionals
232     \csnumdef{cc#1Cnt}{\the\@tempcnta}%
233     \ccCurCount=\the\@tempcnta\relax%
234     \csname cc@\cc@cur@cont @#3\endcsname%
235   \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.

```
236 \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.

```
237 \def\cc@assign@res#1{%
238   \ifx\cc@iterate@res\relax
239     \cslet{#1}\relax
```

```

240 \else
241   \expandafter\csname #1\expandafter\endcsname\expandafter{\cc@iterate@res}%
242 \fi
243 \global\let\cc@iterate@res\relax
244 }

```

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

```

245 \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.

```

246 \def\ccComposeCollection#1#2#3{%
247   \ccIfComp{#3}{\cslet{cc@used@#3@override}\@empty}{%
248     \ifcsdef{cc#1Cnt}{%
249       \expandafter\ifnum\csname cc#1Cnt\endcsname > \z@\relax
250       \edef\cc@iterate@res{%
251         \noexpand\bgroup
252         \noexpand\def\noexpand\ccTotalCount{\csname cc#1Cnt\endcsname}%
253         \noexpand\ccToggleCountedConditionals
254         \noexpand\def\noexpand\cc@cnt@grp{cc#1}%
255         \expandafter\@tempcntb=\csname cc#1Cnt\endcsname\relax
256         \cc@iterate{\@tempcnta}{\@ne}{\@tempcntb}{%
257           \edef\@tempb{%
258             %% top-level counter for user interaction
259             \noexpand\ccCurCount=\the\@tempcnta
260             %% evaluating group attributes
261             \ifcsdef{cc@cc@cur@cont @#1-\the\@tempcnta @attrs}{\noexpand\ccParseAttributes{#1-\
262               the\@tempcnta}{\csname cc@cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}
263             %% internal counter for macro grabbing
264             \noexpand\csnumdef{cc#1Cnt}{\ccCurCount}%
265             \noexpand\ccUseProperty{#2}%
266             \expandafter\expandafter\expandafter\def
267             \expandafter\expandafter\expandafter\cc@iterate@res
268             \expandafter\expandafter\expandafter{\expandafter\cc@iterate@res\@tempb}%
269             }%
270             \expandafter\def\expandafter\cc@iterate@res\expandafter{\cc@iterate@res\egroup}%
271             \cc@assign@res{\ccPrefix#3}%
272           }{}%
273 }{}%

```

**\ccApplyCollection** is an alternative version of **\ccComposeCollection** and fully expands the Property #2 before it is stored inside the Component #3.

```

274 \def\ccApplyCollection#1#2#3{%
275   \ccIfComp{#3}{\cslet{cc@used@#3@override}\@empty}
276   {\cc@apply@collection{#1}{#2}%
277     \cc@assign@res{\ccPrefix#3}%
278   }%
279 }

```

#1 is the group name, #2 is the property to format the collection

```

280 \def\cc@apply@collection#1#2{%
281   \beginngroup

```

```

282 \global\let\cc@iterate@res\relax
283 \letcs\ccTotalCount{cc#1Cnt}%
284 \cc@iterate{\@tempcnta}{\@ne}{\ccTotalCount}{%
285   \bgroup
286     \ccToggleCountedConditionals
287     \def\cc@cnt@grp{cc#1}%
288     \csnumdef{cc#1Cnt}{\the\@tempcnta}%
289     \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}{\ccParseAttributes{#1-\the\@tempcnta
290       }{\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}
291     \ccCurCount=\the\@tempcnta
292     \protected@xdef\@tempb{\csname cc@\cc@cur@cont @#2\endcsname}%
293     \@temptokena \expandafter{\@tempb}%
294     \def\@tempc{\csgappto{cc@iterate@res}}%
295     \expandafter\@tempc\expandafter{\@tempb}%
296   \egroup
297 }%
298 \endgroup
299 }

```

`\cc@comp@def` 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, and #4 is the Property that should be applied to all Members of the Counted Component.

```

299 \def\cc@comp@def{\cc@opt@empty\@cc@comp@def}
300 \def\@cc@comp@def[#1]#2#3#4{%
301   \cc@apply@collection{#3}{#4}%
302   \ifx\cc@iterate@res\relax
303     #1\let#2\relax%
304   \else
305     \def\@tempa{#1\def#2}%
306     \cc@assign@res{\@tempa}%
307   \fi
308 }

```

`\ccdefFromCountedComp` is the user-level command for *local* `\cc@comp@def`.

```

309 \def\ccdefFromCountedComp{\cc@comp@def}

```

`\ccgdefFromCountedComp` is the user-level command for *global* `\cc@comp@def`.

```

310 \def\ccgdefFromCountedComp{\cc@comp@def[\global]}
311 \def\ccpgdefFromCountedComp#1{\expandafter\ccgdefFromCountedComp\csname \ccPrefix #1\endcsname}

```

## Declaring Counted Component

`\ccDeclareCountedComponent` is a user-level macro to create a new Counted Component. #1 is the user-level name of the Component.

```

312 \def\ccDeclareCountedComponent#1{%
313   \cc@def@counted@comp
314     {\cc@counted@comp@scheme{#1}}
315     {#1}
316     {}
317     {\expandafter\global}%
318 }

```

`\cc@def@counted@comp` registers counter dependent 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; #3 is some custom code passed to the second argument of `\ccDeclareComponent`; and #4 is a modifier to the internal macro definition.

```

319 \def\cc@def@counted@comp#1#2#3#4{%
320   \ccDeclareComponent[#1]{#2}
321   {\bgroup#3\expandafter\global}
322   {\def\@tempa{\@cc@reset@components@\cc@cur@cont}}}%
323   \edef\@tempb{\noexpand\csgundef{cc@noexpand\cc@cur@cont @#1}}}%
324   \expandafter\expandafter\expandafter\csgappto\expandafter\@tempa\expandafter{\@tempb}%
325   \egroup}%
326   #4\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#2\endcsname{\csname cc@
327   cc@cur@cont @#1\endcsname}%

```

### 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 type of the Component set.

```

328 \def\cc@reset@components#1{%
329   \csname @cc@reset@components@#1\endcsname
330   \global\cslet{@cc@reset@components@#1}\relax%
331 }

```

### 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!

```

332 \long\def\ccToggleCountedConditionals{%
333   \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`.

```

334 \long\def\ccIfComp##1{%
335   \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
336   csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@secondoftwo\else\expandafter\
337   @firstoftwo\fi%
338 }%
339 \long\def\ccWhenComp##1{%
340   \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
341   csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@gobble\else\expandafter\
342   @firstofone\fi%
343 }%
344 \long\def\ccUnlessComp##1{%

```

```

341 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
      csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@firstofone\else\expandafter\
      @gobble\fi%
342 }%
343 \long\def\ccIfCompEmpty##1{%
344 \expandafter\expandafter\expandafter\ifx\csname cc@\cc@cur@cont @##1\endcsname\cc@long@empty
      \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}%
345 \ccToggleCountedConditionalsHook% legacy
346 }

```

## 6 Hooks

TODO: Use latex3's hook facility instead.

Hooks are used to patch code into different parts of a Container's processing chain.

**\ccDeclareHook** registers a new hook. Optional #1 is the container for which the Hook is declared. If omitted, this defaults to **\cc@cur@cont**. #2 is the Hook's user-level name. Hooks always default to an empty string.

```

347 \def\ccDeclareHook{\cc@opt@curcont\cc@declare@hook}
348 \def\cc@declare@hook[#1]#2{\expandafter\global\expandafter\let\csname cc@hook@#1@#2\endcsname\
      @empty}

```

**\ccAddToHook** adds new material to a Hook. If the hook has not yet been declared, a **\ccDeclareHook** for that hook is applied first. In that case, use the optional #1 to specify the Container name that hook is intended for. If it is omitted, the current Container is used. #2 is the name of the hook the material in #3 is to be appended to.

```

349 \def\ccAddToHook{\cc@opt@curcont\cc@add@to@hook}
350 \def\cc@add@to@hook[#1]#2#3{%
351 \expandafter\ifx\csname cc@hook@#1@#2\endcsname\relax
352 \ccDeclareHook[#1]{#2}%
353 \fi
354 \csgappto{cc@hook@#1@#2}{#3}%
355 }

```

**\ccUseHook** expands the current state of the hook with the name #2 from Container #1 (current Container if omitted).

```

356 \def\ccUseHook{\cc@opt@curcont\cc@use@hook}
357 \def\cc@use@hook[#1]#2{\csuse{cc@hook@#1@#2}}

```

## 7 Properties

### 7.1 Setting Properties

**\ccSetProperty** 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.

```

358 \long\def\ccSetProperty#1#2{\long\csdef{cc@\cc@cur@cont @#1}{#2}}

```

**\ccAppToProp** 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.

```
359 \def\ccAppToProp#1#2{%
360   \long\csappto{cc@\cc@cur@cont @#1}{#2}%
361 }
```

**\ccPreToProp** 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 to previous value of that Property.

```
362 \def\ccPreToProp#1#2{%
363   \long\cspreto{cc@\cc@cur@cont @#1}{#2}%
364 }
```

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

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

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

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

**\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.

```
367 \long\def\ccSetPropertyVal#1#2{\def\@tempa{\ccSetProperty{#1}}\expandafter\@tempa\expandafter
{#2}}
```

**\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.

```
368 \long\def\ccSetPropertyX#1#2{\long\csedef{cc@\cc@cur@cont @#1}{#2}}
```

**\ccAddToProperties** adds the material in the next token to a Container of name #1's **Properties** Type.

```
369 \long\def\ccAddToProperties#1#2{\ccAddToType{Properties}{#1}{#2}}
```

## 7.2 Using Properties

**\ccUseProperty** is a user-level command to directly access a previously set Property.

```
370 \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**.

```
371 \def\cc@store@prop{\cc@opt@empty\cc@store@prop}%
372 \long\def\cc@store@prop[#1]#2#3{%
```



```

373 \protected@edef\@tempa{\ccUseProperty{#3}}%
374 #1\expandafter\def\expandafter#2\expandafter{\@tempa}%
375 }

```

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

```

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

```

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

```

377 \def\ccgdefFromProperty{\cc@store@prop[\global]}
378 \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).

```

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

```

**\ccIfStrEqual** is variant of etoolbox's **\ifstrequal** that first fully expands both comparatives before evaluating them.

```

380 \def\ccIfStrEqual#1#2{%
381 \edef\@argi{#1}\edef\@argii{#2}%
382 \expandafter\expandafter\expandafter\ifstrequal
383 \expandafter\expandafter\expandafter{\expandafter\@argi\expandafter}%
384 \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, and #3 is the new (or added) Value.

```

385 \def\cc@set@property@locally#1#2#3{%
386 \let\@cc@cur@cont\cc@cur@cont
387 \ifdefstring\@cc@cur@cont{Heading}{\let\@cc@cur@cont\ccCurSecName}{}%
388 \csappto{\cc@type@Properties@\@cc@cur@cont}{#1{#2}{#3}}%
389 }

```

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.

**\ccPrefix SetPropLocal**, **\ccPrefix AppPropLocal**, **\ccPrefix PrePropLocal** 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.

```

390 \def\ccSetPropLocal{\cc@set@property@locally\ccSetProperty}
391 \cslet{\ccPrefix SetPropLocal}\ccSetPropLocal%
392 \def\ccAppPropLocal{\cc@set@property@locally\ccAppToProp}
393 \cslet{\ccPrefix AppPropLocal}\ccAppPropLocal%
394 \def\ccPrePropLocal{\cc@set@property@locally\ccPreToProp}
395 \cslet{\ccPrefix PrePropLocal}\ccPrePropLocal%

```

### 7.3 Processing Instructions

In general, processing instructions are commands that are only visible to a specific process and ignored by others. In CoCo<sub>TeX</sub>, 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.

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

### 7.4 Property Conditionals

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

```
397 \long\def\ccIfProp#1#2#3{%
398   \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\relax#3\else
399   \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\cc@long@empty #3\else#2\fi
400   \fi
401   \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`!

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

## 8 Helper macros

### 8.1 Handling of Optional Arguments

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

`\cc@opt@curcont` overrides Container Names with the optional argument.

```
404 \long\def\cc@opt@curcont#1{\@ifnextchar[{\#1}{\#1[\cc@cur@cont]}}%
```

`\cc@opt@empty` passes an empty string if the optional argument is missing.

```
405 \long\def\cc@opt@empty#1{\@ifnextchar[{\#1}{\#1[]}}%
```

`\cc@opt@second` passes the first mandatory argument to the optional argument if the latter is missing.

```
406 \let\cc@opt@second\@dblarg
```

### 8.2 Iterators

`\cc@iterate` traverses in #1-th steps (optional, defaults to +1) through counter #2 start at number #3 until and including number #4 and do at every loop #5 (from `forloop.sty`):

```

407 \long\def\cc@iterate{\@ifnextchar[{\@cc@iterate}{\@cc@iterate[\@ne]}}%
408 \long\def\@cc@iterate[#1]#2#3#4#5{%
409   #2=#3\relax%
410   \expandafter\ifnum#2>#4\relax%
411   \else
412     #5%
413     \advance#2 by #1\relax
414     \cc@iterate[#1]{#2}{\the#2}{#4}{#5}%
415   \fi}%

```

### 8.3 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.

```

416 \begingroup
417 \catcode'"=12

```

**\ccParseAttributes** High level wrapper for the attribute parser; #1 is the parent node of the attribute, #2 is the attribute chain

```

418 \gdef\ccParseAttributes#1#2{%
419   \if!#1!\else
420     \if!#2!\else
421       \def\cc@cur@node{#1}%
422       \cc@parse@attributes #2,,\@nil
423     \fi\fi}

```

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

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

```

424 \gdef\cc@parse@attributes #1,#2,\@nil{%
425   \if!#1!\else
426     \cc@parse@kv#1==\@nil
427   \if!#2!\else
428     \cc@parse@attributes#2,\@nil
429   \fi\fi}

```

and

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

```

430 \gdef\cc@parse@kv#1=#2=#3\@nil{%
431   \edef\@argii{#2}%
432   \ifx\@argii\@empty
433     \expandafter\let\csname cc@\cc@cur@node @attr@#1\endcsname\@empty%
434   \else
435     \ifx #2 =\else
436       \expandafter\def\csname cc@\cc@cur@node @attr@#1\endcsname{#2}%
437     \fi
438   \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`.

```

439 \gdef\cc@parse@csv#1#2{%
440   \if!#1!\else
441     \let\cc@parser@callback#1%
442     \edef\cc@tempa{\csname #2\endcsname}%
443     \ifx\cc@tempa\empty\else
444       \expandafter\@cc@parse@csv\cc@tempa,,\@nil
445     \fi
446   \fi}

```

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

```

447 \gdef\@cc@parse@csv #1,#2,\@nil{%
448   \if!#1!\else
449     \cc@parser@callback{#1}%
450   \fi
451   \if!#2!\else
452     \@cc@parse@csv#2,\@nil
453   \fi
454 }
455 \endgroup

```

`\ccGetAttribute` returns the value of an attribute.

#1 is the attribute node, #2 is the attribute name.

```

456 \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 node, #2 is the attribute name, #3 and #4 are the true and false branch, respectively.

```

457 \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 node, #2 is the attribute name, #3 is what happens if the attribute *is set*.

```

458 \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 node, #2 is the attribute name, #3 is what happens if the attribute *is not set*.

```

459 \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 node, #2 is the attribute name, #3 is the comparison value (a string!), #4 and #5 are the true and false branch, respectively.

```

460 \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 node, #2 is the attribute name, #3 and #4 are the true and false branch, respectively.

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

## 8.4 Style Classes

Style Classes are locally usable sub-Containers.

`\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. 7. 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.

```
462 \long\def\ccDeclareClass{\@ifnextchar [{\@cc@set@class}{\@cc@set@class[default]}}%
463 \long\def\@cc@set@class[#1]#2{\cc@opt@empty{\cc@set@class[#1]{#2}}}%
464 \long\gdef\cc@default@class@default{}
465 \long\def\cc@set@class[#1]#2[#3]#4{%
466   \def\@argii{#2}\ifx\@argii\@empty\let\@argii\cc@str@default\fi%
467   \if!#3!\else
468     \expandafter\long\expandafter\def\csname cc@#1@class@\@argii @parent\endcsname{#3}%
469   \fi
470   \expandafter\long\expandafter\def\csname cc@#1@class@\@argii\endcsname{#4}%
471 }
```

`\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.

```
472 \def\ccUseStyleClass#1#2{%
473   \expandafter\ifx\csname cc@#2@class@#1\endcsname\relax
474     \expandafter\ifx\csname cc@default@class@#1\endcsname\relax
475       \PackageError{cocotex.cls}{Class '#1' with scope '#2' not defined!}{Please declare the
         class '#1'!}%
476     \fi
477   \fi
478   \csname cc@default@class@#1\endcsname%
479   \expandafter\ifx\csname cc@#2@class@#1@parent\endcsname\relax\else
480     \expandafter\ccUseStyleClass\expandafter{\csname cc@#2@class@#1@parent\endcsname}{#2}%
481   \fi
482   \csname cc@#2@class@#1\endcsname}
```

`\CoCoTeX` the CoCoTeX Logo.

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

```
484 %</kernel>
```



## Modul 3

# coco-common.dtx

---

This file provides some macros that are used in more than one CoCoTeX module.

```

24 %<common>

25 %%
26 %% module for CoCoTeX that provides some commonly used base macros.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-common}
34 [2024/03/23 0.4.1 CoCoTeX common module]

```

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

```

35 \RequirePackage{kvoptions-patch}
36 \RequirePackage{xkeyval}
37 \RequirePackage{iftex}

```

## 1 Package options

### 1.1 Accessibility Features

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

```

38 \def\cc@color@enc{cmyk}
39 \define@choicekey{coco-common.sty}{color-enc}[\@cc@color@enc\nr]{srgb,rgb,gray,cmy,cmyk,natural}
40   \let\cc@color@enc\@cc@color@enc
41   \ifcase\nr\relax% srgb
42     \def\cc@color@enc{rgb}%
43   \or% rgb
44   \or% gray
45   \or% cmy
46     \def\cc@color@enc{cmyk}%
47   \or% cmyk
48   \else% natural, i.e. no conversion of color spaces takes place
49   \fi
50 }
51 \ProcessOptionsX
52 \PassOptionsToPackage{\cc@color@enc}{xcolor}%

```

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

```
53 \def\cc@if@preamble{\ifx\nodocument\relax\expandafter\@secondoftwo\else\expandafter\@firstoftwo
    \fi}
54 \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.

```
55 \IfFileExists{coco-xerif.sty}{\RequirePackage{coco-xerif}}{}
```

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

```
56 \RequirePackage{coco-kernel}
```

### 2.1 Hard Dependencies

Hard requirements for all CoCoTeX modules:

```
57 \RequirePackage{xcolor}
```

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

```
58 \RequirePackage{graphicx}
59 \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.

```
60 \def\cc@str@default{default}
```

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

```
61 \def\cc@str@table{table}
```

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

```
62 \def\cc@str@figure{figure}
```

#### Box Registers

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

---

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



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

```
63 \newbox\cc@tempboxa
```

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

```
64 \newbox\cc@tempboxb
```

### Temporary Length and Skip Registers

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

```
65 \newskip\cc@tempskipa
```

## 2.3 Helper macros

`\afterfi` used to execute code after the next `\fi`:

```
66 \def\afterfi#1\fi{\fi#1}
```

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

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

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

```
68 \def\@afterbox{%
69   \everypar{%
70     \if@nobreak
71       \@nobreakfalse
72       \clubpenalty \@M
73       \if@afterindent \else
74         {\setbox\z@\lastbox}%
75         \everypar{}}%
76     \fi
77   \else
78     \clubpenalty \@clubpenalty
79     {\setbox\z@\lastbox}%
80     \everypar{}}%
81   \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.

```
82 \let\hack\@firstofone
```

`\hackfor` intended to hide line breaking macros.

```
83 \let\hackfor\@gobble
```

`\Hack` intended to mask page breaking macros.

```
84 \let\Hack\@firstofone
```

`\Hackfor` intended to hide page breaking macros.

```
85 \let\Hackfor\@gobble
```

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

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

```
86 \long\def\@gobbleopt{\@ifnextchar[\@gobbleopt{\@gobbleopt[]}}%
87 \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.

```
88 \def\ccGobble{%
89   \renewcommand\footnote[2][\the\c@footnote]{\def\@thefnmark{##1}\@makefnmark}%
90   \renewcommand\index[2][]{}%
91   \renewcommand\marginpar[2][]{}%
92   \renewcommand\glossary[2][]{}%
93   \let\hypertarget\@gobbletwo
94   \let\label\@gobble
95 }%
```

## 2.5 Arithmetics

`\CalcRatio` is used to calculate the ratio between two integers.

```
96 \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.

```
97 \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.

```
98 \def\@xminusvskip{%
99   \ifdim\lastskip<\@tempskipb
100   \else
101     \ifdim\lastskip<\z@
102     \else
103       \ifdim\@tempskipb<\z@
```

```

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

```

Compatibility to texlive pre 2020:

```

114 \ifx\@vspace@calcif\@undefined
115 \vskip #1\relax
116 \else
117 \@vspace@calcif{#1}%
118 \fi
119 \else
120 \setlength\@tempskipb{#1}%
121 \@xminusvskip
122 \fi
123 \fi
124 \else
125 \@noitemerr
126 \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`

```

127 \def\the@cc@thispage{%

```

`\cc@abspage`

```

128 \newcount\cc@abspage \cc@abspage\z@

```

`\thecc@abspage`

```

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

```

`\if@cc@odd`

```

130 \newif\if@cc@odd \@cc@oddtrue

```

```

131 \AtBeginDocument{%
132 \global\cc@abspage=\c@page\relax%
133 \g@addto@macro\@outputpage{\global\cc@abspage\c@page}%
134 }

```

We split this into two parts:

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

```

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

```

**\ccSavePage** the second macro writes the actual position of the floating object into the aux files. This macro has to be placed inside the float environment/macro.

```

158 \def\ccSavePage{%
159   \protected@write\@auxout{\def\the@cc@cur@cont{\cc@cur@cont}\let\thecc@abspage\relax}{%
160     \string\expandafter\string\gdef\string\csname\space \cc@cur@cont-\the@cc@thispage-\
161       the@cc@atthispage\string\endcsname{\thecc@abspage}}%
162 }

```

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

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

**\ccPrefix Break** is a general line break macro intended to be re-defined if necessary without touching LaTeX's kernel page and line breaking macros.

```

162 \expandafter\DeclareRobustCommand\expandafter*\expandafter{\csname\ccPrefix Break\endcsname}{\
  hfill\break}

```

### 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. Optional #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.

```

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

```

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

```

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

```

```

181 \csname cc@#2@extract@data\endcsname{#3}{#4}{##1}{##2}%
182 \csname cc@#2@print@entry\endcsname{#4}%
183 \egroup
184 \fi
185 \ifLuaTeX\suppresslongerror=0\fi
186 }}

```

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

```

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

```

`\cc@extract@generic`

```

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

```

`\cc@print@generic`

```

190 \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 namespace, #3 is the Component prefix and #4 is the name of the Content component.

```

191 \def\cc@expand@l@contents#1#2#3#4{%
192   \global\let\cc@tempa\relax
193   \sbox\z@{\def\numberline#1{\xdef\cc@tempa{\noexpand\csdef{cc@#2@#3Number}{##1}}{#1}}%
194   \ifdim\wd\z@>\z@
195     \let\numberline\@gobble%
196     \protected\csdef{cc@#2@#3#4}{#1}%
197     \cc@tempa
198   \else
199     #1%
200   \fi
201   \global\let\cc@tempa\relax
202 }

```

### Custom Content Lists

`\ccDeclareExtraToc` 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.

```

203 \def\ccDeclareContentList#1#2{%
204   \def\cc@add@extra@cl##1{%
205     \expandafter\ifx\csname cc@##1@extra@cl\endcsname\relax
206       \csgdef{cc@##1@extra@cl}{#1}%
207     \else
208       \csgappto{cc@##1@extra@cl}{, #1}%
209     \fi}%
210   \edef\@argii{#2}%
211   \cc@parse@csv\cc@add@extra@cl{\@argii}%
212   \expandafter\newwrite\csname cc@cl@#1\endcsname\relax
213 }

```

`\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.

```

214 \def\ccCreateContentListEntries#1#2#3#4#5{%
215   \def\cc@add@extra@cl##1{%
216     \expandafter\protected@write\csname cc@cl@##1\endcsname
217       {\ccGobble}%
218     {\protect\ccContentsline{#2}{#3}{#4}{#5}\protected@file@percent}\relax
219   }%
220   \ifcsdef{cc@#1@extra@cl}{%
221     \cc@parse@csv\cc@add@extra@cl{cc@#1@extra@cl}{}}%
222 }

```

## 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 `0pt`, 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.

```

223 \def\cc@store@latest#1#2{%
224   \expandafter\ifx\csname cc-\cc@cur@cont-#1\endcsname\relax
225     \csxdef{cc-\cc@cur@cont-#1}{#2}%
226   \else
227     \expandafter\ifdim\csname cc-\cc@cur@cont-#1\endcsname<#2\relax
228       \csxdef{cc-\cc@cur@cont-#1}{#2}%
229     \fi
230   \fi
231   \expandafter\ifx\csname cc-\cc@cur@cont-#1-local\endcsname\relax
232     \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
233   \else
234     \expandafter\ifdim\csname cc-\cc@cur@cont-#1-local\endcsname<#2\relax
235       \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
236     \fi
237   \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 L<sup>A</sup>T<sub>E</sub>X runs before the values stabilize.

```

238 \ifcsdef{cc-\cc@cur@cont-#1-stored-trigger}{}
239 {\edef\@tempa{%
240   \noexpand\immediate\noexpand\write\noexpand\@auxout{%
241     \noexpand\string\noexpand\csgdef{cc-\cc@cur@cont-#1}{%
242       \noexpand\csname cc-\cc@cur@cont-#1-local\noexpand\endcsname}}}%
243   \expandafter\AtEndDocument\expandafter{\@tempa}%
244   \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.

```

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

```

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

```

247 \ccIfComp{#2Number}
248   {\sbox\z@{\ccUseProperty{#1number-format}}}
249   {\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.

```

250 \cc@store@latest{#1number-#3-maxwd}{\the\wd\z@}%
251 \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`.

```

252 \ccSetPropertyVal{#1number-width-level-max}{\csname cc-\cc@cur@cont-#1number-#3-maxwd\
    endcsname}%
253 \ccSetPropertyVal{#1number-width-max}{\csname cc-\cc@cur@cont-#1number-maxwd\endcsname}%
254 \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.

```

255 \cc@get@indent{#1}{#3}%
256 \cc@set@hang{#1}%
257 }
```

`\cc@set@hang` determines and sets the hanging indent of a counter. `#1` is the internal Property prefix.

```

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

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

```

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

Then, we check for `#1indent`.

```

260 \ccIfProp{#1indent}
261   {\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).

```

262   \ccSetProperty{#1hang-number}{%
263     \hskip\ccUseProperty{#1indent}%
264     \hbox to -\ccUseProperty{#1indent}{%
265       \ccIfPropVal{#1number-align}{left}{\hss}%
266       \ccUseProperty{#1number-format}%
267       \ccIfPropVal{#1number-align}{right}{\hss}}}%
268   \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.



```

269 \def\cc@calc@margin@left#1#2{%
270   \@tempcnta\numexpr#2-\@ne\relax
271   \expandafter\ifx\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname\relax
272     \@tempdima=-\ccUseProperty{#1indent}\relax%
273   \else
274     \@tempdima=\dimexpr\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname-\
      ccUseProperty{#1indent}\relax
275   \fi
276   \cc@store@latest{#1#2-margin-left}{\the\@tempdima}%
277   \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.

```

278 \def\cc@get@indent{\@ifnextchar[{\@cc@get@indent}{\@cc@get@indent[\cc@calc@margin@left]}}
279 \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`.

```

280 \ccPropertyLetX{int-#2margin-left}{#2margin-left}%
281 \ccPropertyLetX{int-#2indent}{#2indent}%
282 \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`.

```

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

```

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

```

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

```

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

```

285 \ccIfPropVal{#2margin-left}{}
286 {\ccSetProperty{int-#2margin-left}{\z@}}
287 {\ccPropertyLetX{int-#2margin-left}{#2margin-left}}%
288 \ccSetPropertyX{#2margin-left}{\dimexpr\ccUseProperty{#2number-width-max}+\ccUseProperty{int
  -#2margin-left}\relax}}

```

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

```

289 {\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.

```

290 {\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.

```

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

```

otherwise it is set to the value of `indent`, or `0pt` if it was not set at all.

```

292     {\ccIfProp{int-#2indent}
293      {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}
294      {\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.

```

295     #1{#2}{#3}}

```

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

```

296     {\ccIfProp{int-#2margin-left}
297      {\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.

```

298         {\ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}
299         {\ccIfProp{int-#2indent}

```

Otherwise, if `indent` is set to a specific width, apply that value, or else set the inden to `0pt`.

```

300         {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}
301         {\ccSetProperty{#2indent}{\z@}}}%

```

If `margin-left` is not set,

```

302         {\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.

```

303         {\ccPropertyLetX{#2margin-left}{#2number-width-level-max}%
304          \ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}
305         {\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 `0pt`. In this branch, `indent` should have a positive value, otherwise the content would probably lap over the left edge of the type area.

```

306         {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}%
307         \ccSetProperty{#2margin-left}{\z@}}

```

otherwise set both `indent` nad `margin-left` to `0pt`.

```

308         {\ccSetProperty{#2indent}{\z@}%
309          \ccSetProperty{#2margin-left}{\z@}}}%
310     }

```

### 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.

```

311 \AtBeginDocument{%

```

Storing the final definitions of `\label`

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

```
312 \global\let\cc@ltx@label\label
313 }
```

`\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!

```
314 \def\ccCreateLabel#1{%
315   \ifx\Hy@MakeCurrentHrefAuto\undefined\else
316     \Hy@MakeCurrentHrefAuto{cc:#1}%
317     \Hy@raisedlink{\hyper@anchorstart{\@currentHref}\hyper@anchorend}%
318   \fi
319   \let\cc@ref@label\relax
320   \ccWhenComp{RefLabel}
321     {\ccgdefFromComp\cc@ref@label{RefLabel}%
322      \expandafter\cc@create@label\expandafter{\cc@ref@label}}%
323   \ccIfAttr{\cc@cur@cont}{label}
324     {\cc@parse@csv\cc@create@label{cc@\cc@cur@cont @attr@label}}%
325     {\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 L<sup>A</sup>T<sub>E</sub>X `\label`). #1 is the label ID.

```
326 \def\cc@create@label#1{%
327   \ccIfComp{Number}
328   {\ifx\cc@labelname@comp\undefined
329     \def\cc@labelname@comp{Title}%
330     \fi
331     \begingroup
332       \ccGobble
333       \ccgdefFromComp\@currentlabel{Number}%
334       \ccgdefFromComp\@currentlabelname{\cc@labelname@comp}%
335     \endgroup}%
336   {\cc@fallback@anchor}%
337   %% leaving this will generate lots of "duplicate destination"
338   %% messages from pdfbackend
339   %%\expandafter\hypertarget\expandafter{#1}{}%
340   \expandafter\label\expandafter{#1}%
341 }
342 \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, and #3 is the language specific label.

```
343 \def\ccSetBabelLabel#1#2#3{%
344   \def\ccc@lang{#1}%
345   \expandafter\def\expandafter\ccc@tempa\expandafter{\expandafter\def\csname #2name\endcsname
346     {#3}}%
347   \ifdefstring\ccc@lang{german}{%
348     \expandafter\addto\expandafter\captionsgerman\expandafter{\ccc@tempa}%
349     \expandafter\addto\expandafter\captionsgerman\expandafter{\ccc@tempa}%
350   }
```

```

349 } \relax%
350 \ifdefstring\ccc@lang{english}{%
351   \expandafter\addto\expandafter\captionsbritish\expandafter{\ccc@tempa}%
352   \expandafter\addto\expandafter\captionsUKenglish\expandafter{\ccc@tempa}%
353   \expandafter\addto\expandafter\captionsenglish\expandafter{\ccc@tempa}%
354   \expandafter\addto\expandafter\captionsamerican\expandafter{\ccc@tempa}%
355   \expandafter\addto\expandafter\captionsUSenglish\expandafter{\ccc@tempa}%
356 } \relax%
357 }

```

### 3.6 Link Generation

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

```

358 \def\ccCompLink#1#2{%
359   \protected@edef\@argi{\expandonce{\ccUseComp{#1}}}%
360   \expandafter\href\expandafter{\@argi}{#2}%
361 }

```

**\ccPageLabel** enables referencing pages via ??y using to create a hyperref anchor for label #1.

```

362 \def\ccPageLabel#1{\phantomsection\label{#1}}

```

```

363 %</common>

```

## Modul 4

# coco-accessibility.dtx

---

This file provides code for the interaction between the CoCoT<sub>E</sub>X 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-accrssibility.lua`.

## 1 LaTeX code

```
24 %<*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.

```
25 %%
26 %% Accessibility features for \textit{xerif} projects.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% luatex - texlive > 2018
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-accessibility}
34   [2024/03/23 0.4.1 CoCoTeX accessibility module]
35 \RequirePackage{kvoptions-patch}
36 \RequirePackage{xkeyval}
37 \RequirePackage{atbegshi}
38 \RequirePackage{xparse}
```

The `ltpdfa` package re-defines too many standard LaTeX macros, so we only use its lua code and define the interface ourself. 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 lifes 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.

```
39 \def\cca@lang@id{eng}%
40 \DeclareOptionX{lang-id}{\gdef\cca@lang@id{#1}}

41 \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.

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

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

```
43 \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.

```
44 \let\cca@do@dospaces\relax
45 \DeclareOptionX{no-spaces}{\let\cca@do@dospaces\@undefined}
```

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

```
46 \let\cca@do@doparas\relax
47 \DeclareOptionX{no-paras}{\let\cca@do@doparas\@undefined}
```

Processing the options.

```
48 \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.

```
49 \def\cca@patch@error#1{%
50   \ccPackageError{ally}{compatibility}
51   {Could not patch \noexpand#1}
52   {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.

```
53 \def\cc@if@ally{\ifx\cca@do@ally\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
54 \let\ccIfAlly\cc@if@ally
```

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

```
55 \def\ccWhenAlly{\ifx\cca@do@ally\relax\expandafter\@firstofone\else\expandafter\@gobble\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.

```

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

```

### Additional Hyperref Setup

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

```

64 \AtBeginDocument{%
65   \hypersetup{%
66     % pdfa=true% already set elsewhere
67     ,unicode=true%
68     ,pdfinfo={}%
69     % ,pdfpagelabels=true% already set elsewhere
70     ,pageanchor=true%
71   }%
72   \Hy@pdfatru
73 }

```

### 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:

```

74 \directlua{ltpdfa = require('ltpdfa')}
75 \directlua{ltpdfa.config.final = true}
76 \directlua{ltpdfa.config.debug = \ifcc@debug true\else false\fi}
77 \directlua{ltpdfa.config.nodetree = \ifx\cca@do@nodetree\relax true\else false\fi}
78 \directlua{ltpdfa.config.showspaces = \ifx\cca@do@showspaces\relax true\else false\fi}
79 \directlua{ltpdfa.config.dospaces = \ifx\cca@do@dospaces\relax true\else false\fi}
80 \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:

```

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

```

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

```

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

```

Initial setup of `ltpdfa`

```

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

```

We need the absolute last page of the document

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

## 1.4 Generic Macro to Declare Accessibility Features

In order to selectively enable and disable accessibility macros during runtime, we need each tagging markup macro to exist in two states, one where they trigger tagging into the pdf, and one where they do nothing.

The enabled and disabled versions of each macro are stored inside two separate lists:

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

```
92 \def\cca@relaxed@defs{}
```

and

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

```
93 \def\cca@saved@defs{}
```

The next two macros are used to disable and enable accessibility markup:

`\ccaDisable` disables all ltpdfa commands

```
94 \def\ccaDisable{\cca@relaxed@defs}
```

and

`\ccaEnable` enables all ltpdfa commands.

```
95 \def\ccaEnable{\cca@saved@defs}
```

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

```
96 \ExplSyntaxOn
97 \newcommand{\CsToStr}[1]{\cs_to_str:N #1}
98 \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.

```
99 \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.

```
100 \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:

```
101 \def\cca@declare@cmd@firstopt#1[#2]{\edef\cca@temp@signature{[\unexpanded{#2}]]}%
102 \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.

```
103 \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.

```
104 \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.

```
105 \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.

```
106 \edef\x{\noexpand\let\noexpand#1\expandafter\noexpand\csname cc@saved@\CsToStr{#1}\endcsname}%
107 \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`.

```
108 \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.

```
109 \edef\y{\noexpand\let\noexpand#1\expandafter\noexpand\csname cc@no@\CsToStr{#1}\endcsname}%
110 \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:

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

Finally, we reset the temporary argument signature macro.

```
112 \let\cca@temp@signature\@empty
113 }
```

Some macros from `ltpdfa.sty`:

```
114 \DeclareAccessibilityCommand{\ccaAddToConfig}[2]{\directlua{ltpdfa.addToConfig('\luaescapestring
    {#1}', '\luaescapestring{#2}')}}
115 \@onlypreamble\ccaAddToConfig
```

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

This tagging macro inserts `\bgroup` and `\egroup` around the tagged area.

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

**\ccaVstructStart**, **\ccaVstructEnd** are the same as **\ccaStructStart** and **\ccaStructEnd** but without grouping the area.

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

**\ccaPstructStart**, **\ccaPstructEnd** same as **\ccaStructStart** and **\ccaStructEnd** but no grouping and not setting any attributes. Implies that the element has no content children, at all.

```
120 \DeclareAccessibilityCommand{\ccaPstructStart}[2][\directlua{ltpdfa.tagger.pstructStart('\
    luaescapestring{#2}', '\luaescapestring{#1}')}\fi}
121 \DeclareAccessibilityCommand{\ccaPstructEnd}[1][\directlua{ltpdfa.tagger.pstructEnd('\
    luaescapestring{#1}')}\fi}
```

**\ccaGetCurStruct** returns the internal ID of the currently open structural element. #1 is table attribute that should be returned(?)

```
122 \DeclareAccessibilityCommand{\ccaGetCurStruct}[1][\directlua{ltpdfa.tagger.getCurrentStruct('\
    luaescapestring{#1}')}\fi}
```

**\ccaAddToStruct** adds the current structural element to the structural element #1 (retrieved using **\ccaGetCurStruct**).

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

**\ccaAddID** adds the ID #1 to the current node. If #1 is “auto” the ID is calculated by ltpdfa.

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

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

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

**\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.

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

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

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

**\ccaAddNumbering** ???

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

### `\ccaAddScope`

```
129 \DeclareAccessibilityCommand{\ccaAddScope}[1]{\relax\directlua{ltpdfa.tagger.addScope('\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:

```
130 \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.

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

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

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

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

```
133 \DeclareAccessibilityCommand{\ccaGetStructParent}{\directlua{ltpdfa.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.

```
134 \begingroup
135 \@makeother\#
136 \ccWhenAlly{%
137 \AtBeginDocument{%
138   \patchcmd\Hy@StartlinkName
139     {\pdfstartlink}
140     {\ccaStructStart{Link}\ccaAddAltText{#2}\edef\@ltpdfmy@parent{\ccaGetStructParent}%
141       \pdfstartlink}
142     {}{\cca@patch@error\Hy@StartlinkName}
```

and the parent node inside the link attribute:

```
143   \patchcmd\Hy@StartlinkName
144     {#1}
145     {#1 /StructParent \@ltpdfmy@parent}
146     {}{\cca@patch@error\Hy@StartlinkName}
```

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

```
147   \patchcmd\hyper@linkurl
```

```

148     {\pdfstartlink}
149     {\ccaStructStart{Link}\ccaAddAltText{#2}\edef\ltpdfmy@parent{\ccaGetStructParent}%
150      \pdfstartlink}
151     {}{\cca@patch@error\hyper@linkurl}

```

and secondly for the Parent:

```

152     \patchcmd\hyper@linkurl
153       {/C[\@urlbordercolor]%
154        \fi
155       }
156       {/C[\@urlbordercolor}%
157        \fi
158        /StructParent \ltpdfmy@parent%
159       }{}{\cca@patch@error\hyper@linkurl}

```

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

```

160     \patchcmd\close@pdflink
161       {\pdfendlink}
162       {\pdfendlink
163        \ccaAddLastLink\ccaStructEnd{Link}}
164       {}{\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:

```

165     \let\cca@hy@setref\@setref
166     \let\@setref\real@setref
167     \patchcmd\@setref
168       {\else}
169       {\else\ccaStructStart{Reference}}
170       {}{\cca@patch@error\orig@setref@new}%
171     \patchcmd\@setref
172       {\fi}
173       {\ccaStructEnd{Reference}\fi}
174       {}{\cca@patch@error\orig@setref@new}%

```

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

```

175     \let\real@setref\@setref
176     \let\@setref\cca@hy@setref
177     \patchcmd\@setref
178       {\expandafter\Hy@setref@link}
179       {\ccaStructStart{Reference}\expandafter\Hy@setref@link}
180       {}{\cca@patch@error\@setref}
181     \patchcmd\@setref
182       {{#2}}
183       {{#2}\ccaStructEnd{Reference}}
184       {}{\cca@patch@error\@setref}
185     }% /AtBeginDocument
186 }% /ccWhenAlly
187 \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.

```

188 \DeclareAccessibilityCommand{\ccaPagestyleArtifacts}{%
189   \ifx\@oddhead\@empty\else
190     \pretocmd\@oddhead{\ccaStructStart[document]{header}}{}{}%
191     \apptocmd\@oddhead{\ccaStructEnd{header}}{}{}%
192   \fi
193   \ifx\@evenhead\@empty\else
194     \pretocmd\@evenhead{\ccaStructStart[document]{header}}{}{}%
195     \apptocmd\@evenhead{\ccaStructEnd{header}}{}{}%
196   \fi
197   \ifx\@oddfoot\@empty\else
198     \pretocmd\@oddfoot{\ccaStructStart[document]{footer}}{}{}%
199     \apptocmd\@oddfoot{\ccaStructEnd{footer}}{}{}%
200   \fi
201   \ifx\@evenfoot\@empty\else
202     \pretocmd\@evenfoot{\ccaStructStart[document]{footer}}{}{}%
203     \apptocmd\@evenfoot{\ccaStructEnd{footer}}{}{}%
204   \fi}

```

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

```

205 \apptocmd\ps@empty{\ccaPagestyleArtifacts}{}{}
206 \apptocmd\ps@plain{\ccaPagestyleArtifacts}{}{}
207 \apptocmd\ps@headings{\ccaPagestyleArtifacts}{}{}
208 \apptocmd\ps@myheadings{\ccaPagestyleArtifacts}{}{}

```

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

```

209 \ccWhenAlly{%
210   \ccaAddToConfig{artifact}{header={Type:Pagination}{Subtype:Header}}
211   \ccaAddToConfig{artifact}{footer={Type:Pagination}{Subtype:Footer}}

```

## 1.8 generic artifacts

```

212 \ccaAddToConfig{artifact}{leaders={Type:Layout}}
213 \ccaAddToConfig{artifact}{footnoterule={Type:Layout}}
214 \ccaAddToConfig{artifact}{Rule={Type:Layout}}
215 \ccaAddToConfig{artifact}{Artifact={Type:Layout}}
216 }

```

## 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.

```

217 \DeclareAccessibilityCommand{\ccaAddFigure}[7]{\directlua{ltpdfa.tagger.addFigure(
218   '\luaescapestring{#1}',
219   '\luaescapestring{#2}',
220   '\luaescapestring{#3}',
221   '\luaescapestring{#4}',
222   '\luaescapestring{#5}',
223   '\luaescapestring{#6}',
224   '\luaescapestring{#7}')}}

```

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

```

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

```

**\ccaFigureEnd** injects the ending tag for images

```

226 \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.

```

227 \AtBeginDocument{%
228   \ifcc@modern
229     \let\ltx@Gininclude@graphics\Gininclude@graphics
230     \def\Gininclude@graphics#1{\ifcc@is@final\ccaFigureStart{}\fi\ltx@Gininclude@graphics{#1}\ifcc@is@final\ccaFigureEnd{}\fi}%
231   \else
232     \@ifpackageloaded{grffile}
233     {\pretocmd\grffile\Gininclude@graphics{\ifcc@is@final\ccaFigureStart{}\fi}{}%
234      \apptocmd\grffile\Gininclude@graphics{\ifcc@is@final\ccaFigureEnd{}\fi}{}%
235      {\pretocmd\Gininclude@graphics{\ifcc@is@final\ccaFigureStart{}\fi}{}%
236       \apptocmd\Gininclude@graphics{\ifcc@is@final\ccaFigureEnd{}\fi}{}%
237     }
238   \fi
239 }

```

```

239 \apptocmd\Gininclude@pdftex{\ifcc@is@final%
240   \def\@tempa{!}%
241   \ccaAddFigure{\Ginllx}{\Ginlly}{\Gin@urx}{\Gin@ury}
242   {\ifx\Gin@scalex\@tempa\else \Gin@scalex\fi}
243   {\ifx\Gin@scaley\@tempa\else \Gin@scaley\fi}
244   {\ifGin@clip true\else false\fi}\fi}%rwi/rhi
245   {}{}
246 \AtBeginDocument{%
247   \@ifpackageloaded{htmltabs}%
248   {\let\ltx@ht@valign@box\ht@valign@box
249    \def\ht@valign@box{\ifht@final@render\cc@is@finaltrue\fi\ltx@ht@valign@box}
250    \let\ltx@ht@RenderCell\ht@RenderCell
251    \def\ltx@ht@RenderCell{\@cc@is@finalfalse\ltx@ht@RenderCell}}{}

```

## 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 underlying orthographic characters. This is done via pdfTeX's **glyphtounicode** tables:

```

252 \ifx\pdfextension\@undefined\else

```

```

253 \protected\def\pdfglyptounicode{\pdfextension glyptounicode}
254 \input glyptounicode
255 \edef\pdfgentounicode{\pdfvariable gentounicode}
256 \pdfgentounicode = 1
257 \fi

```

## 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:

```

258 \ccWhenAlly{%
259   \ccDeclareHook[document]{cca/at/begin/document}
260   \AtBeginDocument{%
261     \directlua{ltpdfa.beginDocument('\luaescapestring{ltpdfa@last@page}')}
262     \ccUseHook[document]{cca/at/begin/document}%
263     \directlua{ltpdfa.configAutoclose()}
264     \ccaVstructStart{document}%
265   }
266   \AtEndDocument{%
267     \ccaVstructEnd{document}
268     \directlua{ltpdfa.endDocument()}%
269   }
270 }

```

## 2 Default Role Mapping

Note that this section contains only the role mappings that didn't thematically fit into other CoCo $\TeX$  modules.

```

271 \ccaAddRolemap{document}{Document}
272 \ccaAddRolemap{Para}{P}

```

Finally, we hook `ltpdfa`'s page processor into `AtBeginShipoutBox`:

```

273 \ccWhenAlly{\AtBeginShipout{\directlua{ltpdfa.pageprocessor(tex.box["AtBeginShipoutBox"])}}}%

```

End of  $\TeX$  source code.

```

274 %</ally-sty>

```

```

275 %<*ally-lua>

```

## 3 Lua code

### 3.1 Local Variables and Tables

`ltpdfa` is an instance of the `ltpdfa` Lua table.

```

276 local ltpdfa = require('ltpdfa')

```

### 3.2 Meta Data Extraction

`meta` is a table that holds the metadata that are extracted from the `\jobname.xmp` file via its `extract` member.

```
277 local meta = {
278   Author = '',
279   Title = '',
280   Creator = '',
281   Producer = '',
282   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@write@pdf@meta` from the coco-title module (see Sect. 2).

```
283 extract = function ()
284   local xmpfile = ltpdfa.metadata.xmhandler.fromFile(ltpdfa.config.metadata.xmpfile)
285   local f = io.open(xmpfile, "r")
286   local content = f:read("*all")
287   f:close()
288   if (content:find('<dc:title>')) then
289     Title = content:gsub('.*<dc:title>[<]*<rdf:Alt>[<]*<rdf:li>[>]*>(.*)</rdf:li>[<]*</rdf:
      Alt>[<]*</dc:title>.*', "%1")
290     -- log(">>>" .. meta.Title)
291   end
292   local authors
293   local author = {}
294   if (content:find('<dc:creator>')) then
295     authors = content:gsub('.*<dc:creator>[<]*<rdf:Seq>(.*)</rdf:Seq>[<]*</dc:creator>.*', "%
      1")
296     for k in string.gmatch(authors, "<rdf:li>([>]+)</rdf:li>") do
297       table.insert(author, k)
298     end
299     Author = table.concat(author, ', ')
300   end
301 end
302 }
```

### 3.3 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.

```
303 if type(cocotex) ~= 'table' then
304   cocotex = {}
305 end
```

`cocotex.ally` is a globally available namespace for coco-accessibility specific lua tables.

```
306 cocotex.ally = {
307   meta = meta
308 }
```

After loading `coco-accessibility.lua` via the `require()` method, a `cocotex.ally` table is returned.

```
309 return cocotex.ally
```

no more lua code.



```
310 %</ally-lua>
```



## Modul 5

# coco-meta.dtx

---

This file provides some macros that are used to process meta data, both for the whole document, as well as parts of a document.

```
24 %<*meta>
```

File preamble

```
25 %%
26 %% module for CoCoTeX that provides handling of a document's meta data.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-meta}
34   [2024/03/23 0.4.1 CoCoTeX meta module]
35 \RequirePackage{coco-common}
```

**Container CommonMeta** is an abstract Container for commonly used meta data, both for whole documents as well as parts of documents.

```
36 \ccDeclareContainer{CommonMeta}{%
37   \ccDeclareType{Components}{%
38     \ccDeclareRole[author]{Author}%
39     \ccm@declare@comp
40     \ccm@extended@common@macros
41     \ccm@declare@affils
42   }%
43   \ccDeclareType{Properties}{}%
44 }
```

## 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 and one for its Content.

```
45 \def\ccm@generic@comp{%
46   \ccDeclareComponent{GenericMetaBlock}{\expandafter\global}{}%
47   \ccDeclareComponentGroup{GenericMeta}{%
48     \ccDeclareCountedComponent{Heading}%
49     \ccDeclareCountedComponent{Content}%
50   }}
```

`\ccm@generic@eval` evaluates the Components and tells the Framework how the generic counted Sub-Containers should be rendered.

```

51 \def\ccm@generic@eval{%
52   \def\cc@cur@cont{titlepage}%
53   \ccComposeCollection{GenericMeta}{generic-meta-format}{GenericMetaBlock}
54 }%

```

## 1.2 Contributor Roles

Contributors are counted sub-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.

`\ccDeclareRole` is used to declare the Components that belong to each member of a contributor role. #2 is the name of the role, optional #1 is the internal name of the Role's formatting Property. If omitted, it is the same as #2.

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.

```

55 \def\ccDeclareRole{\cc@opt@second\cc@declare@role}%
56 \def\cc@declare@role[#1]#2{%
57   \ccDeclareComponentGroup{#2}{%
58     \ccDeclareCountedComponent{FullName}%
59     \ccDeclareCountedComponent{CiteName}%
60     \ccDeclareCountedComponent{ShortCiteName}%
61     \ccDeclareCountedComponent{PDFInfoName}%
62     \ccDeclareCountedComponent{Initial}%
63     \ccDeclareCountedComponent{LastName}%
64     \ccDeclareCountedComponent{FirstName}%
65     \ccDeclareCountedComponent{MidName}%
66     \ccDeclareCountedComponent{Honorific}%
67     \ccDeclareCountedComponent{Lineage}%
68     \ccDeclareCountedComponent{ORCID}%
69     \ccDeclareCountedComponent{AffilRef}% for references to the Affil Group
70     \ccDeclareCountedComponent{Affiliation}% for affiliations as direct Author meta data
71     \ccDeclareCountedComponent{Email}%
72     \ccDeclareCountedComponent{CorrespondenceAs}%
73   }%
74   \ccDeclareGroupHandler{#2}{%
75     \ccUnlessComp{FullName}{\ccComponent{FullName}{\ccUseProperty{#1-full-name-format}}}%
76     \ccUnlessComp{Initial}{\ccComponent{Initial}{\ccUseProperty{initials-format}}}%
77     \ccUnlessComp{CiteName}{\ccComponent{CiteName}{\ccUseProperty{#1-cite-name-format}}}%
78     \ccUnlessComp{ShortCiteName}{\ccComponent{ShortCiteName}{\ccUseProperty{#1-short-cite-name-
79       format}}}%
80     \ccUnlessComp{PDFInfoName}{\ccComponent{PDFInfoName}{\ccUseProperty{#1-pdfinfo-name-format
81       }}}%
82     \ccUnlessComp{CorrespondenceAs}{\ccComponent{CorrespondenceAs}{\ccUseProperty{#1-
83       correspondence-as-format}}}%
84     \ccWhenComp{AffilRef}{\ccWhenComp{Affiliation}{%
85       \ccPackageError{Meta}{Ambiguity}
86       {You cannot use both Containers AffilRef and Affiliation in the same '\ccPrefix#2' Sub-
87         Container}
88       {At least one '\ccPrefix#2' Sub-Container contains both AffilRef and Affiliation. This
89         is not allowed. Please decide for one affiliation strategy: Either two lists with
90         cross-references, or affiliations directly as an author's meta-data.}}}%
91   }%
92   \ccDeclareRoleBlock{#2}{NameList}{#1-list-print-format}%
93   \ccDeclareRoleBlock{#2}{CitationList}{#1-list-cite-format}%

```

```

88 \ccDeclareRoleBlock{#2}{ShortCitationList}{#1-list-short-cite-format}%
89 \ccDeclareRoleBlock[apply]{#2}{PDFInfo}{#1-list-pdfinfo-format}%
90 \ccDeclareRoleBlock{#2}{Correspondence}{#1-list-correspondence-format}%
91 }

```

**\ccAddToRole** appends another Component declaration block #2 to a pre-defined Role #1.

```

92 \def\ccAddToRole#1#2{%
93   \csgappto{@#1@hook}{#2}%
94 }

```

**\ccDeclareRoleBlock** is used to create a new output container (named **\ccPrefix#2#3**) for a given Role #2. A Role Block is a Component of the parent Container which contains certain Components of all members of the Role within its parent Container. Format and selection of the utilised Components are specified via the Property given in #4. The optional argument #1 tells the evaluator in the Container's **end** macro how the collector is to be composed. Valid values are **compose** (default) or **apply**.

```

95 \def\ccDeclareRoleBlock{\ifnextchar[\cc@declare@role@block{\cc@declare@role@block[compose]}}%
96 \def\cc@declare@role@block[#1]#2#3#4{%
97   \ifcsdef{ccm@role@#1}
98     {\ccDeclareComponent{#2#3}{\expandafter\global}{}}%
99     \csgdef{ccm@role@\cc@cur@cont @#2@#3}{#4}%
100     \csappto{ccm@role@eval@\cc@cur@cont @#2}
101       {\csname ccm@role@#1\endcsname{#2}{#3}}
102     {\ccPackageError{Meta}{Argument}
103       {Invalid optional argument in \string\ccDeclareRoleBlock!}
104       {Only 'apply' or 'compose' are allowed as values^^in the optional argument of \string\
        ccDeclareRoleBlock!}}}%

```

**\ccm@role@eval** creates the name lists for the role. #1 is the name of the role.

```

105 \def\ccm@role@eval#1{\csname @ccm@role@eval@\cc@cur@cont @#1\endcsname}

```

**\@ccm@role@eval** #1 is the name of the macro used to compose the Collection (either **\ccComposeCollection**, or **\ccApplyCollection**), #2 is the name of the role and #3 is the name of the list. The access Component is **#2#3**, i.e., both arguments together.

```

106 \def\@ccm@role@eval#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.

```

107 \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.

```

108   \ifcsdef{cc#2Cnt}
109     {\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**.

```

110       #1{#2}{\csname ccm@role@\cc@cur@cont @#2@#3\endcsname}{#2#3}%
111       \fi
112     }{}}

```

**\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.

```
113 \def\ccm@role@apply#1#2{\@ccm@role@eval\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.

```
114 \def\ccm@role@compose#1#2{\@ccm@role@eval\ccComposeCollection{#1}{#2}}
```

## 2 Labeled Components

**\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.

```
115 \def\ccDeclareLabeledComp{\cc@opt@empty\cc@declare@labeled@comp}
116 \def\cc@declare@labeled@comp[#1]#2#3{%
117   \ccDeclareComponent{#2}{\expandafter\global}{}%
118   \ccDeclareComponent{#2Label}{\expandafter\global}{}%
119   \csxdef{labeled-meta-property-infix-\cc@cur@cont-#2}{#3}%
120   \if!#1!\else
121     \long\csgdef{cc@\cc@cur@cont @#2Label}{#1}%
122   \fi
123 }
```

**\ccUseLabeledComp** declares two Components: one named **\ccPrefix#1** for the value and another one named **\ccPrefix#1Label** for its corresponding label. An optional Argument allows to set a default value for the Label.

```
124 \def\ccUseLabeledComp{\@ifstar{\global\let\ccm@no@tag\relax\cc@use@labeled@comp}{\
  cc@use@labeled@comp}}
125 \def\cc@use@labeled@comp#1{%
126   \ccWhenComp{#1}{%
```

**\ccCurInfix** stores the currently active property infix for the Labeled Component

```
127 \letcs\ccCurInfix{labeled-meta-property-infix-\cc@cur@cont-#1}%
```

**\ccCurComp** stores the currently active Component name

```
128 \def\ccCurComp{#1}%

129 \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatum}\fi
130 \ccIfProp{labeled-meta-\ccCurInfix-format}
131   {\ccUseProperty{labeled-meta-\ccCurInfix-format}}
132   {\ccUseProperty{labeled-meta-format}}%
133 \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatum}\fi
134 }\global\let\ccm@no@tag\@undefined}
```

## 3 Meta Data Rolemaps for Tagged PDFs

Role mapping for accessibility tagging:

```

135 \ccaAddRolemap{Authors}{Para}
136 \ccaAddRolemap{Affiliations}{Para}
137 \ccaAddRolemap{MetaDatum}{Div}
138 \ccaAddRolemap{MetaDatumLabel}{Para}
139 \ccaAddRolemap{MetaDatumValue}{Para}
140 \ccaAddRolemap{Abstract}{Div}
141 \ccaAddRolemap{AbstractLabel}{P}
142 \ccaAddRolemap{AbstractText}{Div}
143 \ccaAddRolemap{Keywords}{Div}
144 \ccaAddRolemap{KeywordsLabel}{P}
145 \ccaAddRolemap{KeywordsText}{Div}

```

## 4 Common Meta Data

`\ccm@declare@comp` defines some commonly used meta Components

```

146 \def\ccm@declare@comp{%
147   \ccDeclareComponent{Copyright}{\expandafter\global{}}% Copyright text
148   \ccDeclareComponent{DOI}{\expandafter\global{}}% DOI
149 }%

```

### Container article-meta

```

150 %% for single articles
151 \ccDeclareContainer{article-meta}{%
152   \ccDeclareType{Components}{%
153     \ccDeclareGlobalComponent{StartPage} % Start page of a single article
154     \ccDeclareGlobalComponent{EndPage} % End page of a single article
155     \ccDeclareLabeledComp[Cite as]{CiteAs}{cite-as} % As what the article should be cited
156     \ccDeclareLabeledComp[Submitted]{Submitted}{submitted} % Date the article was submitted
157     \ccDeclareLabeledComp[Received]{Received}{received} % Date the article was recieved
158     \ccDeclareLabeledComp[Revised]{Revised}{revised} % Date the article was revised
159     \ccDeclareLabeledComp[Reviewed]{Reviewed}{reviewed} % Date the article was reviewed
160     \ccDeclareLabeledComp[Accepted]{Accepted}{accepted} % Date the article was accepted
161     \ccDeclareLabeledComp[Published]{Published}{published} % Date the article was published
162     \ccDeclareLabeledComp[Conflict of Interest]{COIStatement}{coi-statement}% Conflict of Interest
163     statement
164   }%
165 }

```

`\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.

```

165 \def\ccm@extended@common@macros{%
166   \ccDeclareLabeledComp[Abstract]{Abstract}{abstract}%
167   \ccDeclareLabeledComp[Keywords]{Keywords}{keyword}%
168   \ccDeclareLabeledComp{DOI}{doi}%
169   \ccDeclareLabeledComp{TitleEn}{title-en}%
170   \ccm@generic@comp
171 }

```

## 4.1 Affiliations

`\ccm@declare@affils` is a wrapper that creates the user-level macros for the affiliations.

```

172 \def\ccm@declare@affils{%
173   \ccDeclareComponent{AffilBlock}{\expandafter\global}{}%
174   \ccDeclareComponentGroup{Affil}{%
175     \ccDeclareCountedComponent{Affiliation}%
176     \ccDeclareCountedComponent{Address}%
177     \ccDeclareCountedComponent{Institute}%
178     \ccDeclareCountedComponent{Country}%
179     \ccDeclareCountedComponent{Department}%
180     \ccDeclareCountedComponent{AffilID}%
181   }%
182   \ccDeclareGroupHandler{Affil}{%
183     \ccUnlessComp{AffilID}{\ccComponentEA{AffilID}{\ccAffilCnt}}%
184     \ccUnlessComp{Affiliation}{\ccComponent{Affiliation}{\ccUseProperty{affiliation-format}}}%
185   }%
186 }

```

Default Property settings for the Meta Container.

```

187 \ccAddToProperties{CommonMeta}{%
188   \ccSetProperty{initials-format}{%
189     \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
190       cc@long@empty\else
191     \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
192       relax\else
193     \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\
194       the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
195     \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
196       cc@long@empty\else
197     \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
198       relax\else
199     \ccUseProperty{initials-sep}%
200     \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\
201       the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
202   \fi\fi
203   \fi\fi
204 }
205 \ccSetProperty{initials-sep}{~}
206 \ccSetProperty{initials-period}{.}
207 %
208 %% Properties that control how the composed compoents WITHIN each item in a Role are formatted:
209 %
210 \ccSetProperty{role-full-name-format}{%
211   \if\ccUseComp{Honorific}\relax
212   \else
213     \ccUseComp{Honorific}\space
214   \fi
215   \ccUseComp{FirstName}\space
216   \if\ccUseComp{MidName}\relax
217   \else
218     \ccUseComp{MidName}\space
219   \fi
220   \ccUseComp{LastName}%
221   \if\ccUseComp{Lineage}\relax
222   \else
223     \space\ccUseComp{Lineage}%
224   \fi%
225 }% How FullName for each name is built

```



```

220 \ccSetProperty{role-cite-name-format}{\ccIfComp{LastName}{\ccUseComp{LastName},~\ccUseComp{
    Initial}}{\ccUseComp{FullName}}}% How CiteName for each name is built
221 \ccSetProperty{role-short-cite-name-format}{\ccUseComp{LastName}}% how ShortCiteName for each name
    is built
222 \ccPropertyLet{role-pdfinfo-name-format}{role-cite-name-format}% How PDFInfoName for each item is
    built
223 \ccSetProperty{role-correspondence-as-format}{\ccUseComp{Email}}% How PDFInfoName for each item is
    built
224 %% Properties that control how the single items in a compoent list are formatted:
225 \ccSetProperty{role-block-print-format}{\ccUseComp{FullName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}% How <Role>NameList for each name is build
226 \ccSetProperty{role-block-cite-format}{\ccUseComp{CiteName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}% How each item in Component <Role>CitationList is formatted
227 \ccSetProperty{role-block-short-cite-format}{\ccUseComp{ShortCiteName}\ifnum\ccCurCount<\
    ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How each item in the Component <Role>
    ShortCitationList is formatted
228 \ccSetProperty{role-block-pdfinfo-format}{\ccUseComp{PDFInfoName}\ifnum\ccCurCount<\
    ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How each item in the Component <Role>PDFInfo
    is formatted
229 \ccSetProperty{role-block-correspondence-format}{%
230 \ccIfAttrIsSet{\cc@cnt@grp\the\ccCurCount}{corresp}
231 {\ifx\is@first@corresp\relax
232 \ccUseProperty{corresp-sep}%
233 \else
234 \global\let\is@first@corresp\relax
235 \fi
236 \ccUseComp{CorrespondenceAs}%
237 }{}}% How each item in the Component <Role>Correspondence is formatted
238 % Aliasses
239 % for Role "Author":
240 \ccPropertyLet{author-cite-name-format}{role-cite-name-format}%
241 \ccPropertyLet{author-short-cite-name-format}{role-short-cite-name-format}%
242 \ccPropertyLet{author-full-name-format}{role-full-name-format}%
243 \ccPropertyLet{author-pdfinfo-name-format}{role-pdfinfo-name-format}%
244 \ccPropertyLet{author-correspondence-as-format}{role-correspondence-as-format}%
245 %
246 \ccPropertyLet{author-list-print-format}{role-block-print-format}%
247 \ccPropertyLet{author-list-cite-format}{role-block-cite-format}%
248 \ccPropertyLet{author-list-short-cite-format}{role-block-short-cite-format}%
249 \ccPropertyLet{author-list-pdfinfo-format}{role-block-pdfinfo-format}%
250 \ccPropertyLet{author-list-correspondence-format}{role-block-correspondence-format}%
251 %
252 \ccSetProperty{counted-name-sep}{,\space}%
253 \ccSetProperty{name-and}{\space and\space}%
254 \ccSetProperty{name-et-al}{\space et~al.}%
255 \ccSetProperty{name-sep}{,\space}%
256 \ccSetProperty{corresp-mark}{*}%
257 \ccSetProperty{corresp-sep}{,\space}%
258 %
259 % Affiliation Properties
260 %
261 \ccSetProperty{affiliation-format}{% Format of the affiliation block
262 \ccWhenComp{Institute}{\ccUseComp{Institute}}%
263 \ccWhenComp{Department}{,\ccUseComp{Department}}%
264 \ccWhenComp{Address}{,\ccUseComp{Address}}%
265 }%
266 \ccSetProperty{affil-sep}{\par}
267 \ccSetProperty{affil-block-item-face}{% Font of a single item in the affiliation list
268 \ccSetProperty{affil-block-item-format}{% Format of a single item in the affiliation list
269 \textsuperscript{\ccUseComp{AffilID}}%
270 \bgroup

```

```

271 \ccUseProperty{affil-block-item-face}%
272 \ccUseComp{Affiliation}
273 \egroup%
274 \ifnum\ccCurCount<\ccTotalCount\relax\ccUseProperty{affil-sep}\fi%
275 }
276 \ccSetProperty{affil-block-face}{\small\normalfont}%
277 \ccSetProperty{affil-block-format}{%
278 \ccWhenComp{AffilBlock}
279 {\bgroup
280 \ccUseProperty{affil-block-face}%
281 \ccUseComp{AffilBlock}%
282 \egroup
283 \par
284 }}
285 %
286 % Labeled Meta Properties
287 %
288 \ccSetProperty{labeled-meta-format}{%
289 \ccIfProp{labeled-meta-before-\ccCurInfix}
290 {\ccUseProperty{labeled-meta-before-\ccCurInfix}}
291 {\ccUseProperty{labeled-meta-before}}}%
292 \bgroup
293 \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumLabel}\fi
294 \ccIfProp{labeled-meta-\ccCurInfix-face}
295 {\ccUseProperty{labeled-meta-\ccCurInfix-face}}
296 {\ccUseProperty{labeled-meta-face}}}%
297 \ccIfProp{labeled-meta-\ccCurInfix-label-format}
298 {\ccUseProperty{labeled-meta-\ccCurInfix-label-format}}
299 {\ccUseProperty{labeled-meta-label-format}}}%
300 \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumLabel}\fi
301 \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumValue}\fi
302 \ccUseComp{\ccCurComp}%
303 \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumValue}\fi
304 \egroup
305 \ccIfProp{labeled-meta-after-\ccCurInfix}
306 {\ccUseProperty{labeled-meta-after-\ccCurInfix}}
307 {\ccUseProperty{labeled-meta-after}}}%
308 }
309 \ccSetProperty{labeled-meta-label-format}{%
310 \ccWhenComp{\ccCurComp Label}{%
311 \bgroup
312 \ccUseProperty{labeled-meta-before-\ccCurInfix-label}%
313 \ccIfProp{labeled-meta-\ccCurInfix-label-face}
314 {\ccUseProperty{labeled-meta-\ccCurInfix-label-face}}
315 {\ccUseProperty{labeled-meta-label-face}}}%
316 \ccUseComp{\ccCurComp Label}%
317 \ccIfProp{labeled-meta-\ccCurInfix-label-sep}
318 {\ccUseProperty{labeled-meta-\ccCurInfix-label-sep}}
319 {\ccUseProperty{labeled-meta-label-sep}}}%
320 \egroup
321 }}
322 \ccSetProperty{labeled-meta-label-face}{\bfseries}
323 \ccSetProperty{labeled-meta-label-sep}{:\enskip}
324 \ccSetProperty{labeled-meta-face}{}
325 \ccSetProperty{labeled-meta-before}{}
326 \ccSetProperty{labeled-meta-after}{\par}
327 }

```

```

328 %</meta>

```

## **Part II**

# **Document Level Structures**



## Modul 6

# coco-headings.dtx

This module provides handlers for headings like parts, chapters, sections, or inline headings common to all CoCoTeX projects.

```

24 %<*headings>

25 %%
26 %% module for CoCoTeX that extends heading objects.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive >= 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-headings}
34   [2024/03/23 0.4.1 CoCoTeX headings module]
35 \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:

```

36 \RequirePackage{bookmark}%

```

Since we use our own heading levels, we disable all automatically generated bookmarks.

```

37 \hypersetup{bookmarksdepth=-999}%

```

## 1 Facility for declaring heading levels and their layouts

### Container Heading

```

38 \ccDeclareContainer{Heading}{%
39   \ccInherit{Components,Properties}{CommonMeta}%
40   \ccDeclareType{Parent}{}%
41   \ccDeclareType{Components}{%

```

We already have the Author Component inherited from the `CommonMeta` Container. We therefore just need to declare the overrides.

```

42   \cch@provide@authors%

```

The remaining Components are built as usual.

```

43 \cch@provide@comp{Title}%
44 \cch@provide@comp{Subtitle}%
45 \cch@provide@comp{Number}%
46 \cch@provide@comp{LicenceLogo}%
47 \cch@provide@comp{LicenceName}%
48 \ccDeclareComponent{RefLabel}{}{}%
49 \cch@provide@quotes
50 }%
51 \ccDeclareType{Properties}{}%
52 \ccDeclareEnv{\cch@heading}{\cch@end@heading}%
53 }

```

`\ccDeclareHeading` is the user-level macro to declare new headings.

- #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

```

54 \long\def\ccDeclareHeading{\cc@opt@empty\cc@declare@heading}
55 \long\def\cc@declare@heading[#1]#2#3#4{%

```

First, we check if the heading has already been declared.

```

56 \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.

```

57 \ccPackageInfo{Headings}{}{Appending to '#3'}%
58 \ifcsstring{cch@#3@level}{#2}{}{}%
59 \ccPackageError{Headings}
60 {Level Mismatch}
61 {Level of heading '#3' cannot be altered!}
62 {The already existing heading '#3' has toc level '\csname cch@#3@level\endcsname', but
63   your^^}%
64   re-declaration states '#2'.^^}%
65   ^^}%
66   Consider declaring a new heading altogether with '#3' as parent,^^}%
67   or add Properties to '#3' using \string\ccAddToType\string{Properties\string}\string
68   {#3\string}.}%
69 }%

```

we also check the parent.

```

68 \if!#1!\else
69 \ifcsstring{cc@parent@#3}{#1}{}{}%
70 \ccPackageError{Headings}
71 {Parent Mismatch}
72 {Parent of heading '#3' cannot be altered!}
73 {The already existing heading '#3' inherits from '\csname cc@parent@#3\endcsname',^^}%
74   but your re-declaration sets Parent to '#1'.^^}%
75   ^^}%
76   Consider declaring a new heading altogether with '#1' as parent.}%
77 }%
78 \fi

```

and finally pass the new Properties to the existing heading.

```
79 \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.

```
80 \cch@declare@heading{#2}{#3}%
81 }{% 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.

```
82 \ccDeclareContainer{#3}{%
```

`\cch@<3>@level` stores the numeric heading level for the heading

```
83 \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.

```
84 \ifcsdef{cch@#2@unique}{\csgdef{cch@#2@unique}{#3}}%%
```

```
85 \ccPackageInfo{Headings}{\Declaring heading '#3'}%
86 \edef\@argi{#1}%
87 \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.

```
88 \ifx\@argi\@empty
89 \ccInherit{Components,Properties}{Heading}%
90 \else
91 \ccInherit{Components,Properties,Parent}{#1}%
92 \fi
```

The main body of the heading Declaration is a list of Property definitions which we append to the Sub-Container's "Property" Type.

```
93 \ccDeclareType{Properties}{%
94 #4%
95 }%
```

For each heading we declare some common macros like the ToC entry handlers, the heading's counters and its hooks.

```
96 \ccDeclareType{Init}{%
97 \cch@init@hooks{#3}%
98 \let\@cch@cur@cont\cc@cur@cont
99 \def\cc@cur@cont{Heading}%
100 \cc@init@l@{toc}{#2}{#3}%
101 \let\cc@cur@cont\@cch@cur@cont
102 \cch@init@cnt{#3}%
103 }%
```

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.

```
104 \cch@declare@heading{#2}{#3}%
105 }% \ccDeclareContainer{#3}
106 }% \ifcsdef cc@container@#3 fi
```

If CoCoTeX's accessibility features are active, we need to register each new heading with **ltpdfa**'s autoclose mechanism.

```
107 \ccIfAlly{\cch@add@autoclose{#2}{#3}}}% \AtBeginDocument\ccIfAlly
```

Finally, we check and update the counters for the lowest and highest heading levels, resp.

```
108 \ifnum#2<\cch@min@level\relax
109 \global\cch@min@level=\csname cch@#3@level\endcsname\relax
110 \fi
111 \ifnum#2>\cch@max@level\relax
112 \global\cch@max@level=\csname cch@#3@level\endcsname\relax
113 \fi
114 }% \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.

```
115 \newcount\cch@tempcnta \cch@tempcnta\z@
116 \def\cch@add@autoclose#1#2{%
117 \ccAddToHook[document]{cca/at/begin/document}{%}
```

First, we assign the Sectioning tag and the tag for the section's head itself to the **Sect** and **H#** tags, respectively.

```
118 \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.

```
119 \cch@tempcnta=\numexpr\tw@-\cch@highest@level\relax
120 \advance\cch@tempcnta by #1\relax
121 \ccaAddRolemap{#2head}{H\the\cch@tempcnta}%
122 \ifnum\cch@tempcnta>6\relax
123 \ccaAddRolemap{H\the\cch@tempcnta}{H}%
124 \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.

```
125 \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

```
126 \ifnum\cch@tempcnta=\cch@highest@level
127 \edef\x{\noexpand\ccaAddToConfig{autoclose}{document={Type:Sectioning}{Child:\csname cch@#1
128 @unique\endcsname}{Egroup:false}}}\x%
\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.

```

129 \ifnum\cch@tempcnta=\cch@lowest@level\relax
130 \ifcsstring{cch@#1@unique}{#2}
131 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:none}{Egroup:
false}}}\x}
132 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:none}{Egroup:
false}{Alias:\csname cch@#1@unique\endcsname}}}\x}%
133 \else

```

For all higher heading levels, we look for the next lower heading

```

134 \@tempswatrue
135 \loop

```

by incrementing the heading level counter by one

```

136 \advance\cch@tempcnta@ne\relax

```

and checking the variable `repeat` condition:

```

137 \if@tempswa

```

We don't go further when the current loop counter is already larger than the heading level with the highest level counter.

```

138 \ifnum\cch@tempcnta>\cch@lowest@level\relax
139 \@tempswafalse
140 \else

```

If we are below the highest level, we check if a heading with the current level is defined

```

141 \expandafter\ifx\csname cch@\the\cch@tempcnta @unique\endcsname\relax

```

if not, we continue. This is the case, when heading levels are not sequentially numbered. Which does (and did) happen. For reasons unknown. . .

```

142 \@tempswatrue
143 \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.

```

144 \ifcsstring{cch@#1@unique}{#2}
145 {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:\csname cch@
\the\cch@tempcnta @unique\endcsname}{Egroup:false}}}\x}
146 {\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}%
147 \@tempswafalse
148 \fi
149 \fi

```

We repeat this as long as `\@tempswa` is false. This ensures that all heading levels have exactly one child assigned to them.

```

150 \repeat
151 \fi
152 }}

```

`\cch@min@level`, `\cch@max@level`, `\cch@highest@level`, `\cch@lowest@level` store the level numbers of the highest and lowest defined heading levels, respectively. `\begin{document}`.

```

153 \newcount\cch@min@level \cch@min@level=99\relax
154 \newcount\cch@max@level \cch@max@level=-99\relax
155 \ifx\cch@highest@level\@undefined \def\cch@highest@level{99}\fi
156 \ifx\cch@lowest@level\@undefined \def\cch@lowest@level{-99}\fi
157 \AtEndDocument{%
158   \immediate\write\@mainaux{\string\gdef\string\cch@highest@level{\the\cch@min@level}}%
159   \immediate\write\@mainaux{\string\gdef\string\cch@lowest@level{\the\cch@max@level}}%
160 }%
```

`\cch@create@parent` stores the heading level's name and its parent, if it exists.

```

161 \def\cch@create@parent#1#2{%
162   \def\ccCurSecName{#2}%
163   \if!#1!\else
164     \ccCheckParent{#1}{#2}%
165   \fi%
166 }
```

`\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.

```

167 \def\cch@declare@heading#1#2{%
168   \ccEvalType{Parent}%
169   \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.

```

170 \csgdef{ccUseHeading#2}{%
```

Since heading levels don't define their own environments, we make sure that `Heading` is the namespace we are working in.

```

171   \ccSetContainer{Heading}%
172   \@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.

```

173   \def\cchLevel{#1}%
174   \ccEvalType[#2]{Properties}%
```

Processing the author name list (from `coco-meta.sty`).

```

175   \ccm@role@eval{Author}%
176   \ccComposeCollection{Author}{author-contact-block-format}{AuthorContactBlock}%
177   \ccComposeCollection{Affil}{affil-block-item-format}{AffilBlock}%
```

Processing the `Quote` Group Container, if any.

```

178   \ccComposeCollection{Quote}{quote-block-format}{QuoteBlock}%
```

Hyperref related stuff.

```
179 \def\Hy@toclevel{#1}%
```

Call the mechanism to calculate the heading's counter.

```
180 \cch@auto@number{#1}{#2}%
```

Here, the actual construction of the heading begins.

```
181 \ccUseProperty{heading-par}%
182 \cch@use@hook{before-hook}{#2}%
183 \ccUseProperty{before-heading}%
```

Add vertical space before the heading

```
184 \cch@add@before@skip
```

The counters we calculated earlier and the space needed to render them are evaluated

```
185 \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.

```
186 \ccIfProp{after-skip}{\expandafter\global\expandafter\@tempskipa\expandafter=\ccUseProperty{
after-skip}\relax}{\global\@tempskipa=1sp\relax}%
187 \cch@use@hook{before-print-hook}{#2}%
188 \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.

```
189 \ccUseProperty{before-heading-block}%
```

Labels to be used with LaTeX's cross reference mechanism are defined

```
190 \ccCreateLabel{#2}% label facility
191 \leftskip\ccUseProperty{margin-left}%
192 \rightskip\ccUseProperty{margin-right}%
193 \bgroup
194 \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!**

```
195 \ccIfPropVal{no-toc}{true}{}{\cch@make@toc}% ToC entries
196 \ccIfPropVal{no-BM}{true}{}{\cch@make@bookmarks}% Bookmarks
197 \ccUseProperty{toc-hook}%
198 \ccIfProp{extended}{\ccUseProperty{extended-heading}}{}%
199 \egroup%
200 \cch@make@run% Running headers
201 \ccUseProperty{after-heading-block}%
202 }%
```

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.

```

203 \ifdim\@tempskipa <\z@\relax
204 \cch@make@inline%
205 \else
206 \cch@make@block%
207 \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`.

```

208 \aftergroup\next%
209 }%
210 }

```

`\cch@use@hook` recursively includes a hook #1 from the heading #2's parent before expanding its own version.

```

211 \def\cch@use@hook#1#2{%
212 \expandafter\ifx\csname cc@parent@#2\endcsname\relax\else
213 \edef\@cch@parent{#1-\csname cc@parent@#2\endcsname}%
214 \expandafter\cch@use@hook\expandafter{\@cch@parent}%
215 \fi
216 \ccUseHook{#1-\ccCurSecName}%
217 }

```

`\cch@add@before@skip` is a routine that determines the skip that is inserted before a heading.

```

218 \def\cch@add@before@skip{%
219 \setlength\@tempskipa{\ccUseProperty{before-skip}}%
220 \ifdim\@tempskipa<\z@\relax
221 \def\do@skip{\minusvspace{-\@tempskipa}}%
222 \else
223 \def\do@skip{\addvspace{\@tempskipa}}%
224 \fi%
225 \if@nbreak
226 \everypar{}%
227 \do@skip
228 \else
229 \addpenalty\@secpenalty
230 \do@skip
231 \fi}

```

## 1.1 Initializers for New Heading Levels

`\cch@init@hooks` initializes the Hooks for heading level #1.

```

232 \def\cch@init@hooks#1{%
233 \ccDeclareHook{toc-before-hook-#1}% Expanded before the toc entry is printed
234 \ccDeclareHook{toc-after-hook-#1}% Expanded after the toc entry is printed
235 \ccDeclareHook{before-hook-#1}% Expanded before before-heading property is expanded
236 \ccDeclareHook{before-print-hook-#1}% Expanded at the very beginning of the local definition of \
    @svsec
237 }

```

`\cch@init@cnt` initialises a counter with the name #1 for automatic numbering if it doesn't exist, yet.

```

238 \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.

```

239 \def\cch@auto@number#1#2{%
240   \ccIfPropVal{numbering}{auto}
241   {\expandafter\ifx\csname c@#2\endcsname\relax\cch@init@cnt{#2}\fi
242   \ccIfAttrIsSet{Heading}{nonumber}
243   {}
244   {\ccIfComp{Number}
245   {}
246   {\ifnum #1>\c@secnumdepth\relax\else
247     \stepcounter{#2}%
248     \edef\@tempa{\csname the#2\endcsname}%
249     \ccComponentEA{Number}{\@tempa}%
250   \fi}}
251 }{}}

```

## 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.

```

252 \def\cch@set@author@name@list#1{%

```

first, we look if the Override was given in the `Heading` Container. If so, we do nothing.

```

253   \ccUnlessComp{#1AuthorNameList}{%

```

If not, we look whether or not the general `AuthorNameList` override was given in the `Heading` Container.

```

254   \ifx\cc@used@AuthorNameList@override\@empty

```

If yes, then we copy its value to `#1AuthorNameList`.

```

255     \ccComponent{#1AuthorNameList}{\cc@Heading@AuthorNameList}%
256   \else

```

Or else, we re-build the `#1AuthorNameList` from the raw `Author` Subcontainers by using the `author-list-print-format` Property.

```

257     \ifnum\ccAuthorCnt>\z@
258     \ccdefFromCountedComp\cch@tempa{Author}{author-list-print-format}%
259     \ifx\cch@tempa\relax\else
260       \ccComponent{#1AuthorNameList}{\cch@tempa}%
261     \fi
262   \fi
263 \fi
264 }}%

```

## 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.

```

265 \def\cch@make@toc{%
266   \cc@check@empty{Heading}{Title}{Toc}%
267   \cc@check@empty{Heading}{Number}{Toc}%
268   \cc@check@empty{Heading}{Subtitle}{Toc}%
269   \cch@set@author@name@list{Toc}%
270   \ccIfAttrIsSet{Heading}{notoc}{%
271     {\protected@edef\cch@toc@entry{%
272       \ccIfComp{TocTitle}{\string\ccComponent{TocTitle}{\string\ignorespaces\space\expandonce{\
273         cc@Heading@TocTitle}}}{}%
274       \ccIfComp{TocNumber}{\string\ccComponent{TocNumber}{\string\ignorespaces\space\expandonce{\
275         {cc@Heading@TocNumber}}}{}%
276       \ccIfComp{TocAuthorNameList}{\string\ccComponent{TocAuthorNameList}{\string\ignorespaces\
277         space\expandonce{\cc@Heading@TocAuthorNameList}}}{}%
278       \ccIfComp{TocSubtitle}{\string\ccComponent{TocSubtitle}{\string\ignorespaces\space\
279         expandonce{\cc@Heading@TocSubtitle}}}{}%
280     }%
281     \ccIfProp{toc-level}{
282       {\edef\cch@toc@sec@name{\ccUseProperty{toc-level}}}%
283       {\let\cch@toc@sec@name\ccCurSecName}%
284       \protected@write\@auxout
285       {\ccGobble}%
286       {\string\@writefile{toc}{\protect\ccContentsline{\cch@toc@sec@name}{\cch@toc@entry}{\
287         thepage}{\@currentHref}\protected@file@percent}}\relax
288       \ccCreateContentListEntries{Heading}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage}{\
289         @currentHref}%
290       \ccCreateContentListEntries{\cch@toc@sec@name}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage
291         }{\@currentHref}%
292     }%
293   }%
294 }%

```

`\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.

```

286 \def\cc@toc@extract@data#1#2#3#4{%
287   \ccSetContainer{Heading}%
288   \ccEvalType[#2]{Properties}%
289   \ccDeclareComponent{TocPage}{}{}%
290   \ccComponent{TocPage}{\ccUseProperty{toc-page-face}#4}%
291   \ccDeclareComponent{TocTitle}{}{}%
292   \ccDeclareComponent{TocSubtitle}{}{}%
293   \ccDeclareComponent{TocNumber}{}{}%
294   \ccDeclareComponent{TocAuthorNameList}{}{}%
295   \cc@expand@l@contents{#3}{Heading}{Toc}{Title}%
296   \cc@format@number{toc-}{Toc}{#1}%
297 }

```

`\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.

```
298 \def\cc@toc@print@entry#1{%
299   \bgroup
300     \ccUseHook{toc-before-hook-#1}%
301     \ccUseProperty{toc-before-entry}%
302     \ccUseProperty{toc-format}%
303   \egroup
304   \printentry{#1}%
305 }
306
307 \def\cc@toc@print@entry#1{%
308   \bgroup
309     \ccUseHook{toc-before-hook-#1}%
310     \ccUseProperty{toc-before-entry}%
311     \ccUseProperty{toc-format}%
312   \egroup
313   \printentry{#1}%
314 }
315
316 \def\cc@toc@print@entry#1{%
317   \bgroup
318     \ccUseHook{toc-before-hook-#1}%
319     \ccUseProperty{toc-before-entry}%
320     \ccUseProperty{toc-format}%
321   \egroup
322   \printentry{#1}%
323 }
324
325 \def\cc@toc@print@entry#1{%
326   \bgroup
327     \ccUseHook{toc-before-hook-#1}%
328     \ccUseProperty{toc-before-entry}%
329     \ccUseProperty{toc-format}%
330   \egroup
331   \printentry{#1}%
332 }
333
334 \def\cc@toc@print@entry#1{%
335   \bgroup
336     \ccUseHook{toc-before-hook-#1}%
337     \ccUseProperty{toc-before-entry}%
338     \ccUseProperty{toc-format}%
339   \egroup
340   \printentry{#1}%
341 }
342
343 \def\cc@toc@print@entry#1{%
344   \bgroup
345     \ccUseHook{toc-before-hook-#1}%
346     \ccUseProperty{toc-before-entry}%
347     \ccUseProperty{toc-format}%
348   \egroup
349   \printentry{#1}%
350 }
351
352 \def\cc@toc@print@entry#1{%
353   \bgroup
354     \ccUseHook{toc-before-hook-#1}%
355     \ccUseProperty{toc-before-entry}%
356     \ccUseProperty{toc-format}%
357   \egroup
358   \printentry{#1}%
359 }
360
361 \def\cc@toc@print@entry#1{%
362   \bgroup
363     \ccUseHook{toc-before-hook-#1}%
364     \ccUseProperty{toc-before-entry}%
365     \ccUseProperty{toc-format}%
366   \egroup
367   \printentry{#1}%
368 }
369
370 \def\cc@toc@print@entry#1{%
371   \bgroup
372     \ccUseHook{toc-before-hook-#1}%
373     \ccUseProperty{toc-before-entry}%
374     \ccUseProperty{toc-format}%
375   \egroup
376   \printentry{#1}%
377 }
378
379 \def\cc@toc@print@entry#1{%
380   \bgroup
381     \ccUseHook{toc-before-hook-#1}%
382     \ccUseProperty{toc-before-entry}%
383     \ccUseProperty{toc-format}%
384   \egroup
385   \printentry{#1}%
386 }
387
388 \def\cc@toc@print@entry#1{%
389   \bgroup
390     \ccUseHook{toc-before-hook-#1}%
391     \ccUseProperty{toc-before-entry}%
392     \ccUseProperty{toc-format}%
393   \egroup
394   \printentry{#1}%
395 }
396
397 \def\cc@toc@print@entry#1{%
398   \bgroup
399     \ccUseHook{toc-before-hook-#1}%
400     \ccUseProperty{toc-before-entry}%
401     \ccUseProperty{toc-format}%
402   \egroup
403   \printentry{#1}%
404 }
405
406 \def\cc@toc@print@entry#1{%
407   \bgroup
408     \ccUseHook{toc-before-hook-#1}%
409     \ccUseProperty{toc-before-entry}%
410     \ccUseProperty{toc-format}%
411   \egroup
412   \printentry{#1}%
413 }
414
415 \def\cc@toc@print@entry#1{%
416   \bgroup
417     \ccUseHook{toc-before-hook-#1}%
418     \ccUseProperty{toc-before-entry}%
419     \ccUseProperty{toc-format}%
420   \egroup
421   \printentry{#1}%
422 }
423
424 \def\cc@toc@print@entry#1{%
425   \bgroup
426     \ccUseHook{toc-before-hook-#1}%
427     \ccUseProperty{toc-before-entry}%
428     \ccUseProperty{toc-format}%
429   \egroup
430   \printentry{#1}%
431 }
432
433 \def\cc@toc@print@entry#1{%
434   \bgroup
435     \ccUseHook{toc-before-hook-#1}%
436     \ccUseProperty{toc-before-entry}%
437     \ccUseProperty{toc-format}%
438   \egroup
439   \printentry{#1}%
440 }
441
442 \def\cc@toc@print@entry#1{%
443   \bgroup
444     \ccUseHook{toc-before-hook-#1}%
445     \ccUseProperty{toc-before-entry}%
446     \ccUseProperty{toc-format}%
447   \egroup
448   \printentry{#1}%
449 }
450
451 \def\cc@toc@print@entry#1{%
452   \bgroup
453     \ccUseHook{toc-before-hook-#1}%
454     \ccUseProperty{toc-before-entry}%
455     \ccUseProperty{toc-format}%
456   \egroup
457   \printentry{#1}%
458 }
459
460 \def\cc@toc@print@entry#1{%
461   \bgroup
462     \ccUseHook{toc-before-hook-#1}%
463     \ccUseProperty{toc-before-entry}%
464     \ccUseProperty{toc-format}%
465   \egroup
466   \printentry{#1}%
467 }
468
469 \def\cc@toc@print@entry#1{%
470   \bgroup
471     \ccUseHook{toc-before-hook-#1}%
472     \ccUseProperty{toc-before-entry}%
473     \ccUseProperty{toc-format}%
474   \egroup
475   \printentry{#1}%
476 }
477
478 \def\cc@toc@print@entry#1{%
479   \bgroup
480     \ccUseHook{toc-before-hook-#1}%
481     \ccUseProperty{toc-before-entry}%
482     \ccUseProperty{toc-format}%
483   \egroup
484   \printentry{#1}%
485 }
486
487 \def\cc@toc@print@entry#1{%
488   \bgroup
489     \ccUseHook{toc-before-hook-#1}%
490     \ccUseProperty{toc-before-entry}%
491     \ccUseProperty{toc-format}%
492   \egroup
493   \printentry{#1}%
494 }
495
496 \def\cc@toc@print@entry#1{%
497   \bgroup
498     \ccUseHook{toc-before-hook-#1}%
499     \ccUseProperty{toc-before-entry}%
500     \ccUseProperty{toc-format}%
501   \egroup
502   \printentry{#1}%
503 }
504
505 \def\cc@toc@print@entry#1{%
506   \bgroup
507     \ccUseHook{toc-before-hook-#1}%
508     \ccUseProperty{toc-before-entry}%
509     \ccUseProperty{toc-format}%
510   \egroup
511   \printentry{#1}%
512 }
513
514 \def\cc@toc@print@entry#1{%
515   \bgroup
516     \ccUseHook{toc-before-hook-#1}%
517     \ccUseProperty{toc-before-entry}%
518     \ccUseProperty{toc-format}%
519   \egroup
520   \printentry{#1}%
521 }
522
523 \def\cc@toc@print@entry#1{%
524   \bgroup
525     \ccUseHook{toc-before-hook-#1}%
526     \ccUseProperty{toc-before-entry}%
527     \ccUseProperty{toc-format}%
528   \egroup
529   \printentry{#1}%
530 }
531
532 \def\cc@toc@print@entry#1{%
533   \bgroup
534     \ccUseHook{toc-before-hook-#1}%
535     \ccUseProperty{toc-before-entry}%
536     \ccUseProperty{toc-format}%
537   \egroup
538   \printentry{#1}%
539 }
540
541 \def\cc@toc@print@entry#1{%
542   \bgroup
543     \ccUseHook{toc-before-hook-#1}%
544     \ccUseProperty{toc-before-entry}%
545     \ccUseProperty{toc-format}%
546   \egroup
547   \printentry{#1}%
548 }
549
550 \def\cc@toc@print@entry#1{%
551   \bgroup
552     \ccUseHook{toc-before-hook-#1}%
553     \ccUseProperty{toc-before-entry}%
554     \ccUseProperty{toc-format}%
555   \egroup
556   \printentry{#1}%
557 }
558
559 \def\cc@toc@print@entry#1{%
560   \bgroup
561     \ccUseHook{toc-before-hook-#1}%
562     \ccUseProperty{toc-before-entry}%
563     \ccUseProperty{toc-format}%
564   \egroup
565   \printentry{#1}%
566 }
567
568 \def\cc@toc@print@entry#1{%
569   \bgroup
570     \ccUseHook{toc-before-hook-#1}%
571     \ccUseProperty{toc-before-entry}%
572     \ccUseProperty{toc-format}%
573   \egroup
574   \printentry{#1}%
575 }
576
577 \def\cc@toc@print@entry#1{%
578   \bgroup
579     \ccUseHook{toc-before-hook-#1}%
580     \ccUseProperty{toc-before-entry}%
581     \ccUseProperty{toc-format}%
582   \egroup
583   \printentry{#1}%
584 }
585
586 \def\cc@toc@print@entry#1{%
587   \bgroup
588     \ccUseHook{toc-before-hook-#1}%
589     \ccUseProperty{toc-before-entry}%
590     \ccUseProperty{toc-format}%
591   \egroup
592   \printentry{#1}%
593 }
594
595 \def\cc@toc@print@entry#1{%
596   \bgroup
597     \ccUseHook{toc-before-hook-#1}%
598     \ccUseProperty{toc-before-entry}%
599     \ccUseProperty{toc-format}%
600   \egroup
601   \printentry{#1}%
602 }
603
604 \def\cc@toc@print@entry#1{%
605   \bgroup
606     \ccUseHook{toc-before-hook-#1}%
607     \ccUseProperty{toc-before-entry}%
608     \ccUseProperty{toc-format}%
609   \egroup
610   \printentry{#1}%
611 }
612
613 \def\cc@toc@print@entry#1{%
614   \bgroup
615     \ccUseHook{toc-before-hook-#1}%
616     \ccUseProperty{toc-before-entry}%
617     \ccUseProperty{toc-format}%
618   \egroup
619   \printentry{#1}%
620 }
621
622 \def\cc@toc@print@entry#1{%
623   \bgroup
624     \ccUseHook{toc-before-hook-#1}%
625     \ccUseProperty{toc-before-entry}%
626     \ccUseProperty{toc-format}%
627   \egroup
628   \printentry{#1}%
629 }
630
631 \def\cc@toc@print@entry#1{%
632   \bgroup
633     \ccUseHook{toc-before-hook-#1}%
634     \ccUseProperty{toc-before-entry}%
635     \ccUseProperty{toc-format}%
636   \egroup
637   \printentry{#1}%
638 }
639
640 \def\cc@toc@print@entry#1{%
641   \bgroup
642     \ccUseHook{toc-before-hook-#1}%
643     \ccUseProperty{toc-before-entry}%
644     \ccUseProperty{toc-format}%
645   \egroup
646   \printentry{#1}%
647 }
648
649 \def\cc@toc@print@entry#1{%
650   \bgroup
651     \ccUseHook{toc-before-hook-#1}%
652     \ccUseProperty{toc-before-entry}%
653     \ccUseProperty{toc-format}%
654   \egroup
655   \printentry{#1}%
656 }
657
658 \def\cc@toc@print@entry#1{%
659   \bgroup
660     \ccUseHook{toc-before-hook-#1}%
661     \ccUseProperty{toc-before-entry}%
662     \ccUseProperty{toc-format}%
663   \egroup
664   \printentry{#1}%
665 }
666
667 \def\cc@toc@print@entry#1{%
668   \bgroup
669     \ccUseHook{toc-before-hook-#1}%
670     \ccUseProperty{toc-before-entry}%
671     \ccUseProperty{toc-format}%
672   \egroup
```

```

303 \ccUseHook{toc-after-hook-#1}%
304 \ccUseProperty{toc-after-entry}%
305 \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.

```

306 \def\cch@make@run{%
307   \cc@check@empty{Heading}{Title}{Run}%
308   \cc@check@empty{Heading}{Number}{Run}%
309   \cc@check@empty{Heading}{Subtitle}{Run}%
310   \cch@set@author@name@list{Run}%
311   \ccUseProperty{running-extra}%
312   \ccIfProp{running-level}
313     {\letcs\cch@mark@name{\ccUseProperty{running-level}mark}}
314     {\letcs\cch@mark@name{\ccCurSecName mark}}%
315   \letcs\cch@parent{cc@parent@\ccCurSecName}%
316   \ifx\cch@mark@name\undefined
317     \ifx\cch@parent\relax\else
318       \letcs\cch@mark@name{\cch@parent mark}%
319     \fi
320   \fi
321   \ifx\cch@mark@name\undefined\else
322     \begingroup
323       \ccGobble
324       \protected@edef\@tempa{\csname cc@Heading@running-heading\endcsname}%
325       \expandafter\cch@mark@name\expandafter{\@tempa}%
326     \endgroup
327   \fi
328 }

```

## 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.

```

329 \def\cch@make@bookmarks{%
330   \cc@check@empty{Toc}{Heading}{Title}{BM}%
331   \cc@check@empty{Toc}{Heading}{Number}{BM}%
332   \cc@check@empty{Toc}{Heading}{AuthorNameList}{BM}%
333   \cc@check@empty{Toc}{Heading}{Subtitle}{BM}%
334   \ccIfAttrIsSet{Heading}{noBM}
335     {}
336     {\ccIfProp{bookmark-level}{\edef\Hy@toclevel{\ccUseProperty{bookmark-level}}}{}}%
337   \begingroup
338     \ccGobble
339     \protected@edef\@tempa{\csname cc@Heading@bookmark\endcsname}%
340     \bookmark[level=\Hy@toclevel,dest=\@currentHref]{\expandonce{\@tempa}}%
341   \endgroup
342 }

```

## 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.

```

343 \newbox\cch@inline@sec@box
344 \def\cch@make@inline{%
345   \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
346   \ccIfProp{interline-para}
347   {\global\setbox\cch@inline@sec@box\hbox{\ifvoid\cch@inline@sec@box\else\unhbox\
      cch@inline@sec@box\ccUseProperty{interline-para-sep}\fi\@svsec}}%
348   {\global\setbox\cch@inline@sec@box\hbox{\@svsec}}
349   \@nobreakfalse
350   \global\@noskipsectrue
351   \gdef\next{%
352     \global\everypar{%
353       \if@noskipsec
354         \global\@noskipsecfalse
355         {\setbox\z@\lastbox}%
356         \clubpenalty\@M
357         \begingroup
358           \unhbox\cch@inline@sec@box
359         \endgroup
360         \unskip
361         \hskip -\@tempskipa
362       \else
363         \clubpenalty \@clubpenalty
364         \global\setbox\cch@inline@sec@box\box\voidb@x
365         \everypar{}}%
366       \fi}%
367   \ignorespaces}}

```

### 3.2 Block Headings

`\cch@make@block` is used to print block headings.

```

368 \def\cch@make@block{%
369   \@svsec
370   \ccUseProperty{after-heading-par}%
371   \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
372   \gdef\next{%
373     \ifdim\parskip>\z@\relax\advance\@tempskipa-\parskip\relax\fi
374     \vskip \@tempskipa
375     \@afterheading
376     \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.



```

377 \def\cch@heading{\cc@opt@empty\@cch@heading}%
378 \def\@cch@heading[#1]#2{%

```

Adding start tags for the contents that “belong” to a heading. **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 older and more modern 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.

```

379 \ccIfAlly
380   {\global\let\cch@currenvir\@currenvir
381   \endgroup
382   \ccaVstructStart{#2}%
383   \ifnum\luatexversion>111\relax\UseHook{env/\ccPrefix Heading/before}\fi
384   \@ignorefalse
385   \begingroup
386   \@endpfalse
387   \let\@currenvir\cch@currenvir
388   \edef\@currenvline{\on@line}%
389   \ifnum\luatexversion>111\relax
390     \@execute@begin@hook{\ccPrefix Heading}%
391   \fi
392   }{}%

```

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.

```

393 \cch@reserve

```

Handling of the optional argument

```

394 \ccParseAttributes{Heading}{#1}%

```

and treatment of heading-level specific style classes.

```

395 \ccWhenAttr{Heading}{class}
396   {\global\let\cch@current@class\cc@Heading@attr@class% TODO: check if still needed!
397   \expandafter\ccUseStyleClass\expandafter{\cc@Heading@attr@class}{Heading}}%

```

`\ccCurSecName` stores the name of the current heading level.

```

398 \edef\ccCurSecName{#2}%

```

The cascaded Properties of the heading level are expanded. This is excluded into its own macro to simplify re-definition if necessary.

```

399 \ccEvalType{#2}{Components}%
400 }

```

`\cch@end@heading` is stuff that happens at the end of the `Heading` environment.

```

401 \def\cch@end@heading{%
402   \expandafter\ifx\csname ccUseHeading\ccCurSecName\endcsname\relax
403     \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.}%

```

```

404 \else
405   \csname ccUseHeading\ccCurSecName\endcsname%
406 \fi
407 \cch@reset
408 }

```

## 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:

```

409 \def\cch@reserve{%
410   \ccSetContainer{Heading}%
411   \let\cch@ltx@dbl@backslash\
412   \letcs\{\ccPrefix Break}
413   \let\cc@ltx@label\label
414   \def\ccAuthorCnt{\z@}%
415   \def\ccAffilCnt{\z@}%
416   \cc@reset@components{\cc@cur@cont}%
417 }

```

`\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).

```

418 \def\cch@reset{%
419   \let\cc@cur@cont\relax
420   \let\cch@ltx@dbl@backslash
421   \let\label\cc@ltx@label
422   \let\ccCurSecName\relax
423 }

```

`\cch@provide@quotes` covers multiple quotation blocks associated with a heading.

```

424 \def\cch@provide@quotes{%
425   \ccDeclareComponent{QuoteBlock}{}{}%
426   \ccDeclareComponentGroup{Quote}{}%
427   \ccDeclareCountedComponent{QuoteText}%
428   \ccDeclareCountedComponent{QuoteSource}%
429 }%
430 }

```

`\cch@provide@authors` sets up the additional Components for the Author Role specific to headings.

```

431 \def\cch@provide@authors{%
432   \ccAddToRole{Author}{}%
433   \ccDeclareCountedComponent{AuthorContact}%
434 }%
435 \ccDeclareRoleBlock{Author}{ContactBlock}{author-contact-block-format}%
436 \ccDeclareGroupHandler{Author}{}%
437   \ccIfComp{AuthorContact}{}{\ccComponent{AuthorContact}{\ccUseProperty{author-contact-format}}}%
438 }%
439 \cc@provide@overrides{AuthorNameList}%
440 }

```

`\cch@provide@comp` is a wrapper that creates the user-level macros for the Component itself and its overrides. #1 is the Component name.

```
441 \def\cch@provide@comp#1{%
442   \ccDeclareComponent{#1}{}{}%
443   \cc@provide@overrides{#1}%
444 }
```

`\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 (`\ccPrefix#1`), *ii* data sent to toc (`\ccPrefix Toc#1`), (iii) data sent to the page styles (`\ccPrefix Run#1`), and (iv) the data sent to the PDF bookmarks (`\ccPrefix BM#1`).

```
445 \def\cc@provide@overrides#1{%
446   \ccDeclareComponent{Toc#1}{}{}% toc overrides
447   \ccDeclareComponent{Run#1}{}{}% running overrides
448   \ccDeclareComponent{BM#1}{}{}% bookmark overrides
449 }
```

## 5 Defaults

```
450 \ccAddToProperties{Heading}{%
451   \ccSetProperty{interline-para}{}%
452   \ccSetProperty{interline-para-sep}{\space}
453   \ccSetProperty{heading-para}{%
454     \ccIfProp{interline-para}{\if@noskipsec \leavevmode \fi}{}%
455     \par
456     \global\@afterindenttrue
457   }%
458   \ccSetProperty{after-heading-para}{\par \nobreak}% par commands at the end of non-inline headings
459   \ccSetProperty{before-heading}{}%
460   \ccSetProperty{title-face}{\bfseries}%
461   \ccSetProperty{subtitle-face}{\normalfont}%
462   \ccSetProperty{author-face}{\normalfont}%
463   \ccSetProperty{quote-face}{\raggedleft}%
464   \ccSetProperty{quote-source-face}{}%
465   \ccSetProperty{quote-block-format}{}%
466   \bgroup
467     \ccUseProperty{quote-face}%
468     \ccUseComp{QuoteText}\par
469     \ccIfComp{QuoteSource}{\ccUseProperty{quote-source-face}--\space\ccUseComp{QuoteSource}}\
470     \par{}{}%
471   \egroup}
472 \ccSetProperty{heading-block}
473   {\ccUseProperty{main-title-format}%
474     \ccIfComp{Subtitle}{\ccUseProperty{subtitle-face}\ccUseComp{Subtitle}}\par{}{}%
475     \ccIfComp{AuthorNameList}{\ccUseProperty{author-face}\ccUseComp{AuthorNameList}}\par{}{}%
476     \ccIfComp{QuoteBlock}{\ccUseComp{QuoteBlock}}{}%
477     \ccIfComp{AffilBlock}{\ccUseProperty{affil-block-face}\ccUseComp{AffilBlock}}\par{}{}%
478   }%
479 \ccSetProperty{main-title-format}{%
480   \ccUseProperty{title-face}%
481   \ccaVstructStart{\ccCurSecName head}%
482   \ccIfComp{Number}%
483   {\ccUseProperty{hang-number}}%
484   {\leftskip0pt}%
```

```

484 \ccUseComp{Title}
485 \ccaVstructEnd{\ccCurSecName head}%
486 \par
487 }
488 \ccSetProperty{extended-heading}{%
489 \ccIfComp{Abstract}
490 {\par\vskip\baselineskip
491 {\bfseries\ccIfComp{AbstractLabel}}{\ccUseComp{AbstractLabel}}{Abstract}}\par
492 {\itshape\small\ccUseComp{Abstract}}\par}
493 {}%
494 \ccIfComp{Keywords}
495 {\par\vskip\baselineskip
496 {\bfseries\ccIfComp{KeywordsLabel}}{\ccUseComp{KeywordsLabel}}{Keywords}}\par
497 {\itshape\small\ccUseComp{Keywords}}\par}
498 {}%
499 }%
500 \ccSetProperty{before-skip}{\z@skip}% TODOC: values < 0pt use \minusvspace, else \addvspace. LaTeX's
    default behaviour of @afterindent is relocated to the after-indent property.
501 \ccSetProperty{after-heading-block}{}%
502 \ccSetProperty{before-heading-block}{\parindent\z@ \parskip\z@}%
503 \ccSetProperty{toc-hook}{}% Called, after ToC and BM entries have been written to the .aux file
504 \ccSetProperty{after-indent}{}%
505 \ccSetProperty{margin-left}{}%
506 \ccSetProperty{margin-right}{\@flushglue}%
507 \ccSetProperty{after-skip}{1sp}%
508 \ccSetProperty{indent}{auto}%
509 \ccSetProperty{number-width}{}%
510 \ccSetProperty{number-sep}{\space}%
511 \ccSetProperty{number-align}{left}%
512 \ccSetProperty{number-format}{}%
513 \bgroup
514 \ccUseProperty{title-face}%
515 \ccUseProperty{number-face}%
516 \ccUseComp{Number}%
517 \ccUseProperty{number-sep}%
518 \egroup
519 \ccSetProperty{numbering}{auto}%
520 %% running header
521 \ccSetProperty{running-level}{}% override level for running title, name
522 \ccSetProperty{running-heading}{}%
523 \ccIfComp{RunAuthorNameList}{\ccUseComp{RunAuthorNameList}:\space}{}%
524 \ccUseComp{RunTitle}%
525 }%
526 %% ToC
527 \ccSetProperty{no-toc}{false}% toc entries are generally disabled iff true
528 \ccSetProperty{no-BM}{false}% bookmark entries are generally disabled, iff true
529 \ccSetProperty{toc-margin-top}{\z@}% left indent of the whole entry
530 \ccSetProperty{toc-margin-bottom}{\z@}% bottom margin of the whole entry
531 \ccSetProperty{toc-margin-left}{auto}% left indent of the whole entry
532 \ccSetProperty{toc-margin-right}{\@pnumwidth}% right margin of the whole entry
533 \ccSetProperty{toc-title-face}{}% appearance of title
534 \ccSetProperty{toc-indent}{auto}% offset of the first line of the entry. auto: hang indent by max-
    number-width for the level
535 \ccSetProperty{toc-number-width}{}% current width of the TocNumber
536 \ccSetProperty{toc-number-align}{left}% alignment of TocNumber within the hbox when hanging
537 \ccPropertyLet{toc-number-face}{toc-title-face}% appearance of the TocNumber
538 \ccSetProperty{toc-number-sep}{\enskip}% thing between TocNumber and TocTitle
539 \ccSetProperty{toc-number-format}{}% Format of the TocNumber
540 \bgroup
541 \ccUseProperty{toc-number-face}%
542 \ccUseComp{TocNumber}%

```

```

543 \ccUseProperty{toc-number-sep}%
544 \egroup}
545 \ccSetProperty{toc-page-sep}{\dotfill}% between TocTitle and the page counter
546 \ccSetProperty{toc-page-face}{}% appearance of the page value
547 \ccSetProperty{toc-page-format}{}% format of the page value
548 \ccUseProperty{toc-page-sep}%
549 \bgroup
550 \ccUseProperty{toc-page-face}%
551 \ccUseComp{TocPage}%
552 \egroup}%
553 \ccSetProperty{toc-link}{none}% should toc entries be linked? values: none,title,page,all
554 \ccSetProperty{toc-level}{}% override heading level for ToC, name!
555 \ccSetProperty{toc-before-entry}{}% stuff before anything is output; used to setup margins, alignment,
    line-breaking rules, etc.
556 \addvspace{\ccUseProperty{toc-margin-top}}%
557 \parindent \z@
558 \let\\\@centercr
559 \hyphenpenalty=\@M
560 \rightskip \ccUseProperty{toc-margin-right} \@plus 1fil\relax
561 \parfillskip -\rightskip
562 \leftskip\ccUseProperty{toc-margin-left}%
563 }%
564 \ccSetProperty{toc-after-entry}{\par\addvspace{\ccUseProperty{toc-margin-bottom}}}% Thing at the
    end of the entry, after the page number
565 \ccSetProperty{toc-format}{}% Order and formatting of the entry itself
566 \ccUseProperty{toc-title-face}%
567 \ccaStructStart{TOCI}%
568 \ccIfComp{TocNumber}
569 {\ccaStructStart{P}\ccaStructStart{Reference}\ccaStructStart{Lbl}\ccUseProperty{toc-hang-
    number}\ccaStructEnd{Lbl}}
570 {\leftskip0pt\leavevmode}%
571 \ccaVstructStart{Span}%
572 \ccTocLink{%
573 \ccWhenComp{TocAuthorNameList}{\ccUseComp{TocAuthorNameList}:\space}%
574 \ccUseComp{TocTitle}%
575 \ccUseProperty{toc-page-format}%
576 }%
577 \ccaVstructEnd{Span}%
578 \ccWhenComp{TocNumber}{\ccaStructEnd{Reference}\ccaStructEnd{P}}%
579 \ccaStructEnd{TOCI}%
580 }%
581 %% PDF-Bookmarks
582 \ccSetProperty{bookmark-level}{}% override heading level for PDF bookmarks, numeric!
583 \ccSetProperty{bookmark}{}%
584 \ccIfComp{BMNumber}{\ccUseComp{BMNumber}\space}%
585 \ccUseComp{BMTitle}%
586 }%
587 \ccSetProperty{orcid-link}{}% how the ORC-ID is rendered
588 \ccIfComp{ORCID}{\ccCompLink{ORCID}{\includegraphics[height=1em]{logos/ORCID.pdf}}}%
589 }%
590 %% a single Author's contact information block
591 \ccSetProperty{author-contact-format}{}%Format of a single author's contact information
592 \ccUseComp{FullName}\ccWhenComp{RefAffil}{\textsuperscript{\ccUseComp{AffilRef}}}%
593 \ccUseProperty{orcid-link}%
594 }%
595 \ccPropertyLet{author-list-format}{author-list-print-format}%
596 \ccSetProperty{author-contact-block-format}{}% Format of the whole contact information block
597 \ccUseComp{AuthorContact}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi
598 }%
599 }

```

## 6 Miscellaneous

### 6.1 Alternative paragraph separation

`\ccPrefix NewPar` 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 0pt in the preamble. This macro is intended to be used at the end of the first of the paragraphs.

```

600 \newdimen\ccnewparskip \AtBeginDocument{\ifdim\ccnewparskip=\z@\relax \ccnewparskip=1\
    baselineskip\relax\fi}
601 \csdef{\ccPrefix NewPar}{\@ifnextchar[{\cc@newpar}{\cc@newpar[\the\ccnewparskip]}}%
602 \def\cc@newpar[#1]{%
603   \ifhmode\par\fi
604   \vskip#1\relax
605   \@afterheading
606 }

```

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

`\TitleBreak`

```

607 \letcs\TitleBreak{\ccPrefix Break}

```

```

608 %</headings>

```

## Modul 7

# coco-notes.dtx

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.

```

24 %<*endnotes>

25 %%
26 %% module for CoCoTeX that handles footnote/endnote switching.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-notes}
34 [2024/03/23 0.4.1 le-tex coco notes module]

```

internal switch for endnotes (`\ccn@use@enttrue`) or footnotes (`\ccn@use@enfalse`, default).

```

35 \newif\if@ccn@use@en \ccn@use@enfalse
36 \newif\if@ccn@en@links \ccn@en@linksfalse

```

package options:

- `endnotes` activates endnotes.
- `ennotoc` prevents chapter headings in the Notes section from creating toc entries.
- `resetnotesperchapter` resets foot- and endnotes at the start of each chapter level heading. If omitted (default) foot- or endnotes are numbered throughout the whole document
- `endnotesperchapter` implies `endnotes` and allows the output of all collected endnotes at the end of each chapter. It also sets the note's heading to section level (otherwise it is chapter level).

```

37 \DeclareOption{endnotes}{\global\@ccn@use@enttrue}
38 \DeclareOption{ennotoc}{\global\let\ccn@en@no@toc\relax}
39 \DeclareOption{resetnotesperchapter}{\global\let\ccn@reset@notes@per@chapter\relax}
40 \DeclareOption{endnoteswithchapters}{\global\@ccn@use@enttrue\global\let\ccn@en@with@chapters\relax}
41 \DeclareOption{endnotelinks}{\global\@ccn@en@linkstrue}
42 \ProcessOptions

```

footnote package is mandatory since it provides the `\savenotes` and `\spewnotes` macros:

```

43 \RequirePackage{footnote}

```

Handling of endnotes:

```

44 \newif\if@enotesopen
45 \AtBeginDocument{\edef\ccn@parindent{\the\parindent}}
46 \if@ccn@use@en
47 \RequirePackage{endnotes}
48 \@ifpackageloaded{coco-headings}{\let\ccn@use@TeX@heading\relax}{}

```

```

49 % Allow linking endnotes to their respective occurrence in the document.
50 \if@ccn@en@links
51   \global\newcount\endnoteLinkCnt \global\endnoteLinkCnt\z@
52   \def\@endnotemark{%
53     \leavevmode
54     \ifhmode\edef\@x@sff{\the\spacefactor}\nobreak\fi
55     \phantomsection%
56     \label{endnote-\the\endnoteLinkCnt}%
57     \hyperref[endnotetext-\the\endnoteLinkCnt]{\makeenmark}%
58     \ifhmode\spacefactor\@x@sff\fi%
59     \relax%
60   }
61 \fi
62 \let\footnote=\endnote
63 \def\enotesize{\normalsize}%
64 \def\enoteformat{%
65   % Create the label right at the start of the endnote text to prevent erroneous pointing to the next page
66   .
67   \if@ccn@en@links%
68     \phantomsection%
69     \label{endnotetext-\currentEndnote}%
70   \fi
71   \noindent
72   \leavevmode
73   \hskip-2em\hb@xt@2em{%
74     \if@ccn@en@links
75       \hyperref[endnote-\currentEndnote]{\@theenmark}\hss%
76     \else
77       \@theenmark\hss%
78     \fi%
79   }\expandafter\parindent\ccn@parindent\relax\expandafter%
80 }%
81 \gdef\enoteheading{%
82   \leftskip2em
83 }%
84 \def\printnotes{%
85   \ifx\ccn@en@with@chapters\relax
86     \ifnum\c@endnote>\z@
87       \expandafter\global\expandafter\let\csname enotes@in@\the\realchap\endcsname\@empty
88     \fi
89   \fi
90   \if@enotesopen
91     \global\c@endnote\z@%
92     \bgroup
93     %\parindent\z@
94     \parskip\z@
95     \theendnotes
96     \egroup
97   \fi}
98 \else
99   \newcount\c@endnote \c@endnote\z@
100   \let\printnotes\relax
101 \fi
102 \newcount\realchap \realchap\z@
103 \ifx\ccn@en@with@chapters\relax
104   \AtBeginDocument{%
105     \ccAddToHook[heading]{before-hook-chapter}{%
106       \ifnum\c@endnote>\z@\relax
107         \expandafter\global\expandafter\let\csname enotes@in@\the\realchap\endcsname\@empty
108       \fi

```



```

109 \global\advance\realchap\@ne
110 \global\c@endnote\z@
111 \def\ccn@par@title{\ccIfComp{TocTitle}{\ccUseComp{TocTitle}}{\ccUseComp{Title}}}%
112 \def\ccn@par@runtile{\ccIfComp{RunTitle}{\ccUseComp{RunTitle}}{\ccUseComp{Title}}}%
113 \addtoendnotes{%
114 \noexpand\expandafter\noexpand\ifx\noexpand\csname enotes@in@\the\realchap\noexpand\
    endcsname\noexpand\@empty
115 \bgroup
116 \noexpand\leftskip\noexpand\z@
117 \noexpand\begin{heading}\ifx\ccn@en@no@toc\relax[notoc]\fi{section}%
118 \noexpand\ccComponent{Title}{\ccn@par@title}%
119 \noexpand\ccComponent{RunTitle}{\ccn@par@runtile}%
120 \noexpand\end{heading}%
121 \egroup
122 \noexpand\fi}%
123 }%
124 }
125 \fi
126 \ifx\ccn@reset@notes@per@chapter\relax
127 \AtBeginDocument{%
128 \ccAddToHook[heading]{before-hook-chapter}{%
129 \global\c@footnote\z@
130 \global\c@endnote\z@
131 }%
132 }%
133 \fi

```

Here 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.

```

134 \long\def\fn@fntext#1{%
135 \ifx\ifmeasuring@\@undefined%
136 \expandafter\@secondoftwo\else\expandafter\@iden%
137 \fi%
138 {\ifmeasuring@\expandafter\@gobble\else\expandafter\@iden\fi}%
139 {%
140 \global\setbox\fn@notes\vbox{%
141 \unvbox\fn@notes%
142 \fn@startnote%
143 \@makefntext{%
144 \rule\z@\footnotesep%
145 \ignorespaces%
146 #1%
147 \@finalstrut\strutbox%
148 }%
149 \fn@endnote%
150 }%
151 }%
152 }

```

Adding artifact tagging to the footnoterule:

```

153 \pretocmd\footnoterule{\ccaVstructStart[document]{footnoterule}}{}{}
154 \apptocmd\footnoterule{\ccaVstructEnd{footnoterule}}{}{}

```

Re-definition of `footnote` package's footnote mark retriever to allow non-numeric values in the optional argument of `\footnote`.

```

155 \def\fn@getmark@i#1[#2]{%
156 \sbox\z@{\@tempcnta#2\relax}%
157 \ifdim\wd\z@>0\p@\relax

```

```

158 \def\thempfn{#2}%
159 \fn@getmark@iii%
160 \else
161 \csname c@\mpfn\endcsname#2%
162 \fn@getmark@ii%
163 \fi
164 }
165 \def\fn@getmark@iii#1{%
166 \unrestored@protected@xdef\@thefnmark{\thempfn}%
167 \endgroup%
168 #1%
169 }

```

And the same for plain L<sup>A</sup>T<sub>E</sub>X:

```

170 \def\@xfootnote[#1]{%
171 \begingroup
172 \sbox\z@{\@tempcnta0#1\relax}%
173 \ifdim\wd\z@>0p@\relax
174 \unrestored@protected@xdef\@thefnmark{#1}%
175 \else
176 \csname c@\mpfn\endcsname #1\relax
177 \unrestored@protected@xdef\@thefnmark{\thempfn}%
178 \fi
179 \endgroup
180 \@footnotemark\@footnotetext%
181 }

```

patching \@footnotemark

```

182 \pretocmd\@footnotemark{%
183 \ccaStructStart{Span}\protected@xdef\@lt@fn@parent{\ccaGetCurStruct{idx}}}%
184 \ccaStructStart{footnotemark}%\addAltText{\@thefnmark}
185 }{}{}
186 \apptocmd\@footnotemark{%
187 \ccaStructEnd{footnotemark}\ccaStructEnd{Span}%
188 }{}{}

```

patching \@makefnintext

```

189 \pretocmd\@makefnintext{%
190 \ccaStructStart{footnotetext}%
191 \ifx\@lt@fn@parent\@empty\relax\else\addToStruct{\@lt@fn@parent}\fi%
192 }{}{}
193 \apptocmd\@makefnintext{%
194 \ccaAddID{auto}\ccaStructEnd{footnotetext}%
195 }{}{}

```

Adding footnotemark and footnotetext PDF tags to the rolemap

```

196 \ccaAddRolemap{footnotemark}{Reference}
197 \ccaAddRolemap{footnotetext}{Note}

```

Linking endnotes requires overwriting the endnotetext macro to save a global counter to the \*.ent file.

```

198 \global\newif\if@haveenotes
199 \long\def\@endnotetext#1{%
200 \global\@haveenotesttrue
201 \if@enotesopen \else \@openenotes \fi
202 \immediate\write\@enotes{%
203 \if@ccn@en@links

```

```
204 \string\def\string\currentEndnote{\the\endnoteLinkCnt}%  
205 \fi%  
206 \@doanenote{\@theenmark}%  
207 }%  
208 \begingroup  
209 \def\next{#1}%  
210 \newlinechar='40  
211 \immediate\write\@enotes{\meaning\next}%  
212 \endgroup  
213 \immediate\write\@enotes{\@endanenote}%  
214 \if@ccn@en@links  
215 \global\advance\endnoteLinkCnt\@ne%  
216 \fi%  
217 }
```

```
218 %</endnotes>
```



## Modul 8

# coco-script.dtx

This package is used to handle non-latin based script systems like Japanese, Chinese, Armenian and the like.

```

24 %<script>

25 %% module for CoCoTeX that handles script switching.
26 %%
27 %% Maintainer: p.schulz@le-tex.de
28 %%
29 %% lualatex - texlive > 2019
30 %%
31 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
32 \ProvidesPackage{coco-script}
33   [2024/03/23 0.4.1 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.

```

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   %
49   \expandafter\parse@script\usescript,,\relax
50 \fi
51 \message{^^} [coco-script Fonts loaded: \meaning\usescript^^]

```

## 1 Default fallback font

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\textssffallback{\sffallbackfont}

```

## 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}`

```

65 \def\ccDeclareBabelFont{\ccopt@empty\ccs@declare@babel@font}%
66 \def\ccs@declare@babel@font[#1]#2#3#4{%
67   \expandafter\ifx\csname use@script@#2\endcsname\@empty
68     \babelprovide[#1]{#2}%
69     \message{^^J [coco-script Loaded Script: #2]^^J}%
70     %%
71     \expandafter\gdef\csname ccs@babel@rm@font@#2\endcsname{#3}%
72     \expandafter\gdef\csname ccs@babel@sf@font@#2\endcsname{#4}%
73     \if!#2!\else
74       \def\ccs@tempa{\babelfont[#2]{rm}}%
75       \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@rm@font@#2\endcsname
76       \fi
77     \if!#3!\else
78       \def\ccs@tempa{\babelfont[#2]{sf}}%
79       \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@sf@font@#2\endcsname
80       \fi
81     \fi
82 }

```

Top level macro to declare a font alias.

- #1 font family alias
- #2 font family fallback

```

83 \def\ccBabelAlias#1#2{%
84   \ifx\usescript\relax\else
85     \def\ccs@callback##1{%
86       \expandafter\ifx\csname ccs@no@fallback@##1\endcsname\relax
87         \expandafter\ifx\csname ccs@babel@#2@font@##1\endcsname\relax
88           \PackageError
89             {coco-script.sty}
90             {\expandafter\string\csname #2family\endcsname\space for Language ‘##1’ was not
              declared!}
91             {You attempted to declare an alias towards a font family that has not been declared
              for the language ‘##1’, yet.}%
92         \else
93           \def\ccs@tempa{\babelfont[##1]{#1}}%
94           \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@#2@font@##1\endcsname
95           \fi
96         \else

```

```

97 \PackageInfo{coco-script.sty}{^^J\space\space\space\space No fallback for ‘##1’;^^J\space
    \space\space\space Skipping font family ‘#1’->‘#2’}%
98 \fi}%
99 \expandafter\parse@script\usescript,,\relax
100 \fi}

```

## 3 Predefined script systems

### 3.1 Support for Armenian script

```

101 \ifx\use@script@armenian\@empty
102 \message{^^J [coco-script Loaded Script: Armenian]^^J}
103 \def\NotoArmenianPath{./fonts/Noto/Armenian/}
104 \newfontfamily\fallbackfont@armenian{NotoSansArmenian-Regular.ttf}%
105 [BoldFont = NotoSansArmenian-Bold.ttf,%
106 Path = \NotoArmenianPath,%
107 WordSpace = 1.25]
108 \DeclareTextFontCommand\armenian{\fallbackfont@armenian}
109 \let\ccs@no@fallback@armenian\@empty%
110 \fi

```

### 3.2 Support for Chinese script

```

111 \ccDeclareBabelFont{chinese}{[%
112 Path=./fonts/Noto/Chinese/,
113 BoldFont = NotoSerifSC-Bold.otf,%
114 WordSpace = 1.25]{NotoSerifSC-Regular.otf}}
115 {[%
116 Path=./fonts/Noto/Chinese/,
117 BoldFont = NotoSansSC-Bold.otf,%
118 WordSpace = 1.25]{NotoSansSC-Regular.otf}%
119 }

```

### 3.3 Support for Japanese script

```

120 \ccDeclareBabelFont{japanese}{[%
121 Path=./fonts/Noto/Japanese/,
122 BoldFont = NotoSerifJP-Bold.otf,%
123 WordSpace = 1.25]{NotoSerifJP-Regular.otf}
124 }{[%
125 Path=./fonts/Noto/Japanese/,
126 BoldFont = NotoSansJP-Bold.otf,%
127 WordSpace = 1.25]{NotoSansJP-Regular.otf}
128 }

```

### 3.4 Support for Hebrew script

```

129 \ccDeclareBabelFont{hebrew}{[%
130 Scale=MatchUppercase,%

```

```

131 Path=./fonts/Noto/Hebrew/,%
132 Ligatures=TeX,%
133 BoldFont = NotoSerifHebrew-Bold.ttf]{NotoSerifHebrew-Regular.ttf}%
134 }{[%
135 Scale=MatchUppercase,%
136 Path=./fonts/Noto/Hebrew/,%
137 Ligatures=TeX,%
138 BoldFont = NotoSansHebrew-Bold.ttf]{NotoSansHebrew-Regular.ttf}%
139 }

```

### 3.5 Support for Arabic script

```

140 \ccDeclareBabelFont{arabic}{[%
141   BoldFont = NotoNaskhArabic-Bold.ttf,%
142   Path = ./fonts/Noto/Arabic/%
143   ]{NotoNaskhArabic-Regular.ttf}}
144 {[%
145   BoldFont = NotoSansArabic-Bold.ttf,%
146   Path = ./fonts/Noto/Arabic/%
147   ]{NotoSansArabic-Regular.ttf}%
148 }

```

### 3.6 Support for Greek script

```

149 \ccDeclareBabelFont{greek}{[%
150   BoldFont = NotoSerif-Bold.ttf,%
151   ItalicFont = NotoSerif-Italic.ttf,%
152   BoldItalicFont = NotoSerif-BoldItalic.ttf,%
153   Path = ./fonts/Noto/Serif/,%
154   WordSpace = 1.25
155   ]{NotoSerif-Regular.ttf}}
156 {[%BoldFont = NotoSans-Bold.ttf,%
157   ItalicFont = NotoSans-Italic.ttf,%
158   BoldItalicFont = NotoSans-BoldItalic.ttf,%
159   Path = ./fonts/Noto/Sans/,%
160   WordSpace = 1.25%
161   ]{NotoSans-Regular.ttf}%
162 }

```

### 3.7 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.

```

163 \expandafter\ifx\csname use@script@syriac\endcsname\@empty%
164 \RequirePackage{filecontents}
165 \begin{filecontents*}{babel-syriac.ini}
166 [identification]
167 charset = utf8
168 version = 0.1

```



```

169 date = 2019-08-25
170 name.local = xxxxxxxxxx
171 name.english = Classical Syriac
172 name.babel = classicalsyrac
173 tag.bcp47 = syc
174 tag.opentype = SYR
175 script.name = Syriac
176 script.tag.bcp47 = Syrc
177 script.tag.opentype = syrc
178 level = 1
179 encodings =
180 derivate = no
181 [captions]
182 [date.gregorian]
183 [date.islamic]
184 [time.gregorian]
185 [typography]
186 [characters]
187 [numbers]
188 [counters]
189 \end{filecontents*}
190 \fi

```

Now, we can create the fallback font and import the newly created ini file:

```

191 \ccDeclareBabelFont[import=syriac]{syriac}{[%
192   BoldFont = NotoSansSyriac-Black.ttf,%
193   ItalicFont = NotoSansSyriac-Regular.ttf,%
194   BoldItalicFont = NotoSansSyriac-Black.ttf,%
195   Path = ./fonts/Noto/Syriac/,%
196   WordSpace = 1.25
197   ]{NotoSansSyriac-Regular.ttf}}
198 {[BoldFont = NotoSansSyriac-Black.ttf,%
199   ItalicFont = NotoSansSyriac-Regular.ttf,%
200   BoldItalicFont = NotoSansSyriac-Black.ttf,%
201   Path = ./fonts/Noto/Syriac/,%
202   WordSpace = 1.25%
203   ]{NotoSansSyriac-Regular.ttf}%
204 }

```

### 3.8 Support for medieval scripts and special characters

only **rm**!

```

205 \babelfont{mdv}{%
206 Path=fonts/Junicode/,%
207 ItalicFont = Junicode-Italic.ttf,%
208 BoldFont = Junicode-Bold.ttf,%
209 BoldItalicFont = Junicode-BoldItalic.ttf,%
210 ]{Junicode.ttf}
211 \def\mdvfont#1{{\mdvfamily#1}}

```

```

212 %</script>

```



## Modul 9

# coco-title.dtx

This file provides macros and facilities for title pages.

```

24 %<*title>

25 %%
26 %% module for CoCoTeX for maketitle.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-title}
34   [2024/03/23 0.4.1 CoCoTeX title module]
35 \RequirePackage{coco-meta}

```

## 1 Top-Level Interface

Container **titlepage** is the main Container for the document's locally defined meta data.

```

36 \ccDeclareContainer{titlepage}{%
37   \ccInherit {Components,Properties}{CommonMeta}%
38   \ifarticle\ccInherit{Components}{article-meta}\fi
39   \ccDeclareType{Components}{%
40     \cct@simple@comps

```

The following macro provides some meta data Components defined in the **coco-meta** module. They are:

- **Abstract** and **AbstractTitle**,
- **Keywords** and **KeywordsTitle**,
- **DOI** and **DOITitle**, and
- **TitleEn** and **TitleEnTitle**, intended for foreign language publications where the title is translated into English.

```

41   \cct@fundings@comp
42   \cct@role@handlers{author}{Author}%
43   \cct@declare@role{editor}{Editor}%
44   \cct@declare@role{series-editor}{SeriesEditor}%
45 }%
46 \ccDeclareType{Properties}{}%
47 \ccDeclareEnv[Meta]{\cct@meta}{\endcct@meta}%
48 }

```

**\cct@declare@role** declares the roles for editors and series editors and initializes the biography meta block for both.

```

49 \def\cct@declare@role#1#2{%
50   \ccDeclareRole[#1]{#2}%
51   \cct@role@handlers{#1}{#2}%
52 }

```

**\cct@role@handlers** adds title page specific Components and Handlers to the Author, Editor and Series-Editor Roles.

```

53 \def\cct@role@handlers#1#2{%
54   \ccAddToRole{#2}{%
55     \ccDeclareCountedComponent{Bio}%
56     \ccDeclareCountedComponent{Biography}%
57   \ccDeclareGroupHandler{#2}{%
58     \ccIfComp{Biography}{\ccIfComp{Bio}{\ccComponent{Biography}{\ccUseProperty{#1-biography-
59       format}}}}}%
60   \ccDeclareRoleBlock[apply]{#2}{BioBlock}{#1-bio-block-format}%
61 }

```

**\ccDeclareTitlepage** is the default titlepage declarator with the next token being added the titlepage's Property list.

```

62 \def\ccDeclareTitlepage{\ccAddToType{Properties}{titlepage}}

```

**\cct@meta** is the code executed at the beginning of the **\ccPrefix Meta** Container

```

63 \def\cct@meta{%
64   \ccEvalType{Components}%
65 }

```

**\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.

```

66 \def\ccAddTitleRole#1#2{%
67   \ccAddToType{Components}{titlepage}{\cct@declare@role{#1}{#2}}%
68   \ccAddTitleEval{\cct@eds@eval{#2}}%
69 }

```

**\ccAddTitleEval** is a User-level macro to add additional Material titlepage evaluators (the next token).

```

70 \def\ccAddTitleEval{\csgappto{\cct@add@eval}}

```

**\cct@add@eval** is a hook for additional titlepage evaluators

```

71 \def\cct@add@eval{}

```

**\endcct@meta** is the code executed at the end of the **Meta** Container

```

72 \def\endcct@meta{%
73   \ccSetContainer{titlepage}%
74   \ccEvalType{Properties}%
75   \cct@maketitle
76   \ccm@role@eval{Author}%
77   \ccApplyCollection{Affil}{affil-block-item-format}{AffilBlock}%
78   \cct@eds@eval{Editor}%
79   \cct@eds@eval{SeriesEditor}%

```

```

80 \ccm@generic@eval
81 \cct@fundings@eval
82 \cct@add@eval
83 \cc@if@preamble\cct@set@pdfmeta\relax
84 \ccUseHook{document-meta-hook}%
85 \let\cct@cur@cont\@empty
86 }

```

## 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.

```

87 \def\cct@set@pdfmeta{%

```

`\cct@write@pdf@meta` is used to transfer the DocumentInfo meta data to the pdf writer.

```

88 \def\cct@write@pdf@meta##1##2##3{%
89 \let\cct@cur@data\@empty

```

First, we check, whether `coco-accessibility.sty` is used. If so, we check if the User has provided an `xmp` file by reading the required meta data field given in `##2` from that `xmp` file. If there is an `xmp` file and the data field is non-empty, we do nothing, because in this case, the PDF DocInfo is auto-generated from the data in the `xmp` file by the `ltpdfa` package.

```

90 \ccIfAlly{\edef\cct@cur@data{\expandonce{\directlua{tex.print(cocotex.ally.meta.##2)}}}}{%
91 \ifx\cct@cur@data\@empty

```

If the temporary storage `\cct@cur@data` is still empty, we take the value given in `\#\#3` and store its plain text in `\cct@cur@data`. Data conversion is done with `hyperref`'s `\pdfstringdef` macro.

```

92 \pdfstringdef\cct@cur@data{##3}%

```

If the storage is still empty (i.e. the field is also missing in the `Meta` environment), we do nothing.

```

93 \ifx\cct@cur@data\@empty\else

```

If the user has provided the data Component in the `Meta` environment, we pass it either to `hyperref`'s `hypersetup` variable given in `\#\#1` (when `coco-accessibility.sty` is *not* used), or we pass it to `ltpdfa.setDocInfo` using the data field given in `\#\#2`. In this case, the `ltpdfa` automatically creates a `\jobname.xmp` from which the DocInfo will be generated during subsequent  $\text{\LaTeX}$  run(s).

```

94 \ccIfAlly

```

If we use `coco-accessibility`, we invoke `\ccaSetDocinfo{\#\#2}{\#\#3}`,

```

95 {\edef\x{\noexpand\ccaSetDocinfo{##2}}%
96 \expandafter\x\expandafter{\cct@cur@data}}%

```

or `hyperref`'s `\hypersetup{\#\#1=\#\#3}`, if not. Note that we need to feed `\#\#3` directly into `hypersetup` since it passes the values of pdf meta data keys through `pdfstringdef`. If we were to pass `\cct@cur@data`, which already went through `pdfstringdef`, the octal byte sequences from the first run are interpreted a second time, which leads to weird glyphs in the final PDF'S DocInfo. Therefore, we stick with the original input.

```

97     {\protected@edef\x{\noexpand\hypersetup{##1={\expandonce{##3}}}\x}%
98     \fi
99     \fi
100 }%
```

After we decided how we want to process the PDF meta data, we now start to collect the necessary data points:

```

101 \cct@title@insert@xmp
102 \cct@title@process@bk
103 \cct@title@process@bkt
104 \cct@title@process@bka
105 }
```

## 2.1 Processing of the Document's Title

`\cct@title@process@bkt` processes the document's main title

```

106 \def\cct@title@process@bkt{%
107   \cslet{\ccPrefix Break}\space
108   \pdfstringdef\@title{\ccUseComp{Title}}%
109   \cct@write@pdf@meta{pdftitle}{Title}{\ccUseComp{Title}}%
110   \ccpgdefFromProperty{RunBookTitle}{run-book-title}%
111 }
```

## 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.

```

112 \def\cct@title@process@bka{%
113   \@tempswatrue
114   \begingroup
115     \ccGobble
116     \renewcommand\foreignlanguage[2]{\##2}%
117     \ccIfComp{AuthorPDFInfo}
118     {\ccpgdefFromProperty{RunBookName}{AuthorPDFInfo}}
119     {\ccIfComp{EditorPDFInfo}
120      {\ccpgdefFromProperty{RunBookName}{EditorPDFInfo}}
121      {\ifnum\ccAuthorCnt>\z@
122       \setpar{\@par}%
123       \ccgdefFromCountedComp{RunBookName}{Author}{author-list-pdfinfo-format}%
124       \else
125       \ifnum\ccEditorCnt>\z@
126       \ccpgdefFromCountedComp{RunBookName}{Editor}{editor-list-pdfinfo-format}%
127       \else
128       \ccPackageWarning{transcript-title}{Meta Data}{No author or editor given!}%
129       \@tempswafalse
130       \fi
131       \fi}}%
132   \if@tempswa
133     \pdfstringdef\@author{\csname\ccPrefix RunBookName\endcsname}%
134     \cct@write@pdf@meta{pdfauthor}{Author}{\csname\ccPrefix RunBookName\endcsname}%
135     \fi
136   \endgroup
137 }
```

## 2.3 Processing of the PDF's Creator, Producer, and Keywords Meta Data

`\cct@title@process@bkc` processes the metadata for the pdf creator

```

138 \def\cct@title@process@bkc{%
139   \cct@write@pdf@meta{pdfcreator}{Creator}{\ccIfComp{PDFCreator}{\ccUseComp{PDFCreator}}{\
      ccUseComp{Publisher}\ccIfComp{PubPlace}{, \ccUseComp{PubPlace}}{}}}%
140   \cct@write@pdf@meta{pdfproducer}{Producer}{\ccUseComp{PDFProducer}}%
141   \cct@write@pdf@meta{pdfkeywords}{Keywords}{\ccUseComp{Keywords}}%
142 }

```

## 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.

```

143 \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.

```

144 \def\cct@title@insert@xmp@direct{%
145   \edef\include@xmp{\noexpand\include@xmp{\ccUseComp{XmpFile}.xmp}}%
146   \def\include@xmp##1{\IfFileExists{##1}{\@@include@xmp{##1}}{}}%
147   \def\@@include@xmp##1{%
148     \begingroup
149       \immediate\pdfobj stream attr {/Type /Metadata /Subtype /XML}
150       file{##1}
151       \pdfcatalog{/Metadata \the\pdflastobj\space 0 R}
152     \endgroup}%
153   \include@xmp
154 }

```

`\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.

```

155 \def\cct@title@insert@xmp@ltpdfa{%
156   \edef\cca@xmp@file@name{\ccUseComponentFrom{titlepage}{XmpFile}.xmp}%
157   \IfFileExists{\cca@xmp@file@name}
158     {\ccaAddToConfig{metadata}{xmpfile=\cca@xmp@file@name}%
159      \directlua{ally.meta.extract()}}
160     {\ccPackageWarning{A1ly}{File}{%
161       \cca@xmp@file@name\space not found.^}
162      Note that the ltpdfa package will create one^^
163      from the Components given in the Meta Container.}}

```

## 3 Intermediate Level Interfaces

`before-maketitle-hook` Hook that is expanded right before the titlepage is printed.

```

164 \ccDeclareHook[titlepage]{before-maketitle-hook}
165 \ccDeclareHook[titlepage]{document-meta-hook}

```

`\cct@article@titlepage` is the prototype for article title pages.

```
166 \def\cct@article@titlepage{%
167   \ccUseProperty{article-title}%
168 }
```

`\cct@journal@titlepage` is the prototype for journal title pages.

```
169 \def\cct@journal@titlepage{%
170   \ccUseProperty{before-titlepage}%
171   \ccUseProperty{coverpage}%Cover ist kein Bild, wird von uns gebaut
172   \ccUseProperty{before-titlepage-roman}%
173   \ccUseProperty{titlepage-roman}%
174   \ccUseProperty{after-titlepage}%
175 }
```

`\cct@book@titlepage` is the prototype for book (monographs and collections) title pages.

```
176 \def\cct@book@titlepage{%
177   \ccUseProperty{before-titlepage}%
178   \ccWhenComp{Cover}{\ccUseProperty{coverpage}}%
179   \ccUseProperty{before-titlepage-roman}%
180   \ccUseProperty{titlepage-roman}%
181   \ccUseProperty{after-titlepage}%
182 }
```

`\cct@maketitle` assigns one of the above definitions to the `\ccPrefix Maketitle` macro.

```
183 \def\cct@maketitle{%
184   \expandafter\gdef\csname\ccPrefix Maketitle\endcsname{%
185     \let\cc@cnt@grp\@empty
186     \ccUseHook[tITLEpage]{before-maketitle-hook}%
187     \bgroup
188     \ccSetContainer{titlepage}%
189     \ccEvalType{Properties}%
190     \ifarticle
191       \cct@article@titlepage
192     \else
193       \ifjournal
194         \cct@journal@titlepage
195       \else
196         \cct@book@titlepage
197       \fi
198     \fi
199     \egroup
200     \ccUseHook[tITLEpage]{after-maketitle-hook}%
201   }%
202 }
```

### 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

```
203 \def\cct@fundings@comp{%
204   \ccDeclareComponent{FundingBlock}{\expandafter\global}{}}%
```



```

205 \ccDeclareComponentGroup{Funding}{%
206   \ccDeclareCountedComponent{FundName}%
207   \ccDeclareCountedComponent{FundLogo}%
208   \ccDeclareCountedComponent{FundID}%
209 }{}%
210 }

```

**\cct@fundings@eval** Evaluator for the funding

```

211 \def\cct@fundings@eval{%
212   \def\cc@cur@cont{titlepage}%
213   \ccComposeCollection{Funding}{fund-format}{FundingBlock}%
214 }%

```

**\cct@eds@eval** evaluator for the editors

```

215 \def\cct@eds@eval#1{%
216   \ccm@role@eval{#1}%
217   \cct@create@editor@string{#1}%

```

**\cct@create@editor@string** evaluates the editor string and adds a suffix.

```

218 \def\cct@create@editor@string#1{%
219   \expandafter\ifx\csname cc@\cc@cur@cont @#1NameList\endcsname\relax\else
220     \csgappto{cc@\cc@cur@cont @#1NameList}{\letcs\ccTotalCount{cc#1Cnt}\ccUseProperty{editor-
221       suffix}}}%
222 }%

```

## 3.2 Simple Component Declarations

**\cct@simple@comps** wrapper for the Titlepage's simple Components.

```

223 \def\cct@simple@comps{%
224   \ccDeclareGlobalComponent[\jobname]{XmpFile} % File basename of the XMP file ('.xmp' is added
225     automatically)
226   %% Cover
227   \ccDeclareGlobalComponent{Cover} % Path to Cover Image(!)
228   %% Titles
229   \ccDeclareGlobalComponent{Title} % Main Title
230   \ccDeclareGlobalComponent{ShortTitle} % Shortened main title
231   \ccDeclareGlobalComponent{RunTitle} % Shortened main title override for headers
232   \ccDeclareGlobalComponent{AltTitle} % Alternative main title (e.g. for bastard title page)
233   \ccDeclareGlobalComponent{Subtitle} % Sub Title
234   \ccDeclareGlobalComponent{TitleNote} % Additional Title Information (contributor list)
235   \ccDeclareGlobalComponent{RunNames} % Shortened list of names (authors and/or publishers)
236   \ccDeclareGlobalComponent{AltNames} % Alternative list of names (e.g. for bastard title page)
237   %% Series
238   \ccDeclareGlobalComponent{Series} % Series Title
239   \ccDeclareGlobalComponent{SubSeries} % Series Subtitle
240   \ccDeclareGlobalComponent{SeriesNote} % Series Notes
241   \ccDeclareGlobalComponent{Volume} % Series Volume
242   \ccDeclareGlobalComponent{Number} % Series Number
243   \ccDeclareGlobalComponent{EditorNameList} % Editor Text Line
244   \ccDeclareGlobalComponent{SeriesEditorNameList} % Series Editor Text Line
245   %% Publisher
246   \ccDeclareGlobalComponent{Publisher} % Publisher Name

```

```

246 \ccDeclareGlobalComponent{PubDivision} % Publishing Division
247 \ccDeclareGlobalComponent{PubDivInfo} % Publishing Division Info
248 \ccDeclareGlobalComponent{PubPlace} % Publisher Location
249 \ccDeclareGlobalComponent{PubLogo} % Publisher Logo
250 \ccDeclareGlobalComponent{PubNote} % Additional publisher notes
251 \ccDeclareGlobalComponent{PubWeb} % Publisher URL
252 %% Pubication Meta
253 \ccDeclareGlobalComponent{PDFCreator} % Creator for pdf metadata
254 \ccDeclareGlobalComponent[le-tex xerif with CoCoTeX v.0.4.1]{PDFProducer} % PDF producer for pdf
    metadata
255 \ccDeclareGlobalComponent{Dedication} % Dedication
256 \ccDeclareGlobalComponent{Acknowledgements} % Acknowledgements
257 \ccDeclareGlobalComponent{Statement} % Acknowledgements
258 \ccDeclareGlobalComponent{EditionNote} % Edition Note
259 \ccDeclareGlobalComponent{Editorial} % Editorial
260 \ccDeclareGlobalComponent{Edition} % Edition
261 \ccDeclareGlobalComponent{Year} % Publication Year
262 \ccDeclareGlobalComponent{ISBNPreText} % Text before ISBN block
263 \ccDeclareGlobalComponent{ISBN} % ISBN
264 \ccDeclareGlobalComponent{ISSN} % ISSN
265 \ccDeclareGlobalComponent{EISSN} % Ebook-ISSN
266 \ccDeclareGlobalComponent{EpubPreText} % Text between ISBN and eISBN
267 \ccDeclareGlobalComponent{EISBN} % Ebook-ISBN
268 \ccDeclareGlobalComponent{EpubISBN} % Epub-ISBN
269 \ccDeclareGlobalComponent{ElibPDF} % ???
270 \ccDeclareGlobalComponent{BiblISSN} % Bibl-ISBN
271 \ccDeclareGlobalComponent{BibleISSN} % Bible-ISBN
272 %% Funding
273 \ccDeclareGlobalComponent{FundingPreText} % Text before the Funding list
274 \ccDeclareGlobalComponent{FundingPostText} % Text after the Funding list
275 %% Imprint Meta
276 \ccDeclareGlobalComponent{Biblio} % Bibliographical Information
277 \ccDeclareGlobalComponent{BiblioTitle} % Heading Bibliographical Information
278 \ccDeclareGlobalComponent{Print} % Printer
279 \ccDeclareGlobalComponent{PrintNote} % Print Note
280 \ccDeclareGlobalComponent{Lectorate} % Lector
281 \ccDeclareGlobalComponent{Translator} % Translator
282 \ccDeclareGlobalComponent{CoverConcept} % Cover Concept
283 \ccDeclareGlobalComponent{CoverDesign} % Cover Designer
284 \ccDeclareGlobalComponent{CoverImage} % Cover Image Creator
285 \ccDeclareGlobalComponent{Typesetter} % Typesetting company
286 \ccDeclareGlobalComponent{QA} % Quality Assurance
287 \ccDeclareGlobalComponent{UsedFont} % Used Font(s)
288 \ccDeclareGlobalComponent{Conversion} % Data Converison
289 \ccDeclareGlobalComponent{EnvDisclaimer} % Environmental Disclaimer
290 \ccDeclareGlobalComponent{Advertise} % Advertisements
291 %% Licencing
292 \ccDeclareGlobalComponent{LicenceText} % License Description
293 \ccDeclareGlobalComponent{LicenceLogo} % License Logo
294 \ccDeclareGlobalComponent{LicenceLink} % License Link
295 \ccDeclareGlobalComponent{LicenceName} % License Name
296 \ccDeclareGlobalComponent{CopyrightDisclaimer} % Copyright Disclaimer
297 %% for journals
298 \ccDeclareGlobalComponent{JournalName} % Full name of the journal
299 \ccDeclareGlobalComponent{JournalAbbrev} % Short name of the journal
300 \ccDeclareGlobalComponent{Issue} % Issue of the journal
301 \ccDeclareGlobalComponent{PubCycle} % Publication cycle
302 \ccDeclareGlobalComponent{Prices} % Prices of the journal issues or subscription models
303 \ccDeclareGlobalComponent{MemberList} % In case of publishing organizations, this macro may hold a
    list of members.
304 %% Generic additional information

```

```

305 \ccDeclareGlobalComponent{AddNoteI} % Additional information, title page I
306 \ccDeclareGlobalComponent{AddNoteII} % Additional information, title page II
307 \ccDeclareGlobalComponent{AddNoteIII} % Additional information, title page III
308 \ccDeclareGlobalComponent{AddNoteIV} % Additional information, title page IV
309 }

```

## 4 Default Settings

```

310 \ccAddToProperties{titlepage}{%
311   \ccSetProperty{article-title}{}%
312   % Title page hooks
313   % Before \ccPrefix Maketitle and outside the group
314   \ccSetProperty{before-titlepage}{%
315     \pagestyle{empty}%
316     \parindent\z@
317     \parskip\z@
318   }%
319   \ccSetProperty{after-titlepage}{\pagestyle{headings}}%
320   % Pages of title
321   %% Cover page
322   \ccSetProperty{coverpage}{%
323     \bgroup
324     \def\thepage{\@alph\c@page}%
325     \smash{\rlap{%
326       \raise\dimexpr\headheight+\headsep+\topmargin+\topskip-\paperheight\relax
327       \vtop{%
328         \hskip-\oddsidemargin
329         \includegraphics[width=\paperwidth,height=\paperheight]{\ccUseComp{Cover}}%
330       }}}%
331     \ccUseProperty{after-coverpage}%
332     \egroup
333   }%
334   \ccSetProperty{after-coverpage}{\cleardoublepage}%
335   \ccSetProperty{titlepage-roman}{%
336     \ccUsePropertyEnv{titlepage-i}%
337     \clearpage
338     \ccUsePropertyEnv{titlepage-ii}%
339     \clearpage
340     \ccUsePropertyEnv{titlepage-iii}%
341     \clearpage
342     \ccUsePropertyEnv{titlepage-iv}%
343     \clearpage
344   }%
345   %% Generic meta blocks
346   \ccSetProperty{generic-meta-heading-face}{\large}% format of the heading of a generic meta block
347   \ccSetProperty{generic-meta-format}{% Format of a single generic meta-block
348     \ccIfComp{Heading}{\ccUseProperty{generic-meta-heading-face}\ccUseComp{Heading}\par}\vskip\baselineskip}%
349     \ccUseComp{Content}%
350     \par%
351   }%
352   %% Funding
353   \ccSetProperty{funding-columns}{2}
354   \ccSetProperty{funding-format}{}%

```

Fallback for the width in case someone sets up a fixed value for a fund's width.

```

355 \ccSetProperty{fund-width}{.5\textwidth}
356 \ccSetProperty{fund-vertical-sep}{\baselineskip}%
357 \ccSetProperty{fund-sep}{%
358   \expandafter\@tempcnta\CalcModulo{\ccCurCount}{\ccUseProperty{funding-columns}}}%
359   \ifnum\@tempcnta=\z@
360     \par
361     \ifnum\ccCurCount<\ccTotalCount\relax
362       \vskip\ccUseProperty{fund-vertical-sep}%
363     \fi
364   \else
365     \hfill
366   \fi}
367 \ccSetProperty{fund-format}{% Format of a single fund/grant/sponsor
368   \strut\vtop{%
369     \hsize\ccUseProperty{fund-width}%
370     \ccIfComp{FundName}{\ccUseComp{FundName}}{\[1ex]}{}%
371     \includegraphics[width=\ccUseProperty{fund-width}]{\ccUseComp{FundLogo}}}%
372   \ccUseProperty{fund-sep}%
373 }%
374 \ccSetProperty{funding-sep}{4mm}%
375 \ccSetProperty{funding-block}{%
376   \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.

```

377   \ccSetPropertyX{fund-width}{\dimexpr(\textwidth/\ccUseProperty{funding-columns})-(\
378     \ccUseProperty{funding-sep}/\ccUseProperty{funding-columns})\relax}
379   \ccUseProperty{funding-format}%
380   \ccGetComp{FundingPreText}%
381   \ccGetComp{FundingBlock}%
382   \ccGetComp{FundingPostText}%
383   \par
384   \egroup
385 }
386 %% before the roman part of the title pages but after cover page
387 \ccSetProperty{before-titlepage-roman}{%
388   \setcounter{page}{1}%
389   \def\thepage{\roman{page}}%
390 }%
391 \ccSetProperty{titlepage-i}{%
392   \ifmonograph
393     \ccUseComp{AuthorNameList}%
394   \else
395     \ccUseProperty{EditorNameList}%
396   \fi%
397   \vskip\baselineskip
398   \bgroup
399   \ccUseProperty{title-face}\ccUseComp{Title}%
400   \egroup
401 }%
402 \ccSetProperty{titlepage-ii}{%
403   \ccGetComp{Editorial}%
404   \ccGetComp{SeriesNote}%
405   \ccGetComp{GenericMetaBlock}%
406   \vfill
407   \ccUseProperty{bio-output}%
408 }%
409 \ccSetProperty{titlepage-iii}{%
410   \ifmonograph

```

```

410 \ccUseComp{AuthorNameList}%
411 \else
412 \ccUseProperty{EditorNameList}%
413 \fi%
414 \par
415 \ccUseProperty{title-format}%
416 \ccGetComp{Edition}%
417 \ccGetComp{EditionNote}%
418 \vfill
419 \clearpage
420 }%
421 \ccSetProperty{titlepage-iv}{%
422 \ccGetComp{Dedication}% Dedication
423 \ccGetComp{Acknowledgements}% Dedication
424 \ccUseProperty{imprint-format}%
425 \ccUseProperty{funding-block}%
426 \vfill
427 \bgroup
428 \ccUseProperty{imprint-face}%
429 \ccIfComp{Biblio}{\bfseries\ccGetComp{BiblioTitle}}\ccGetComp{Biblio}{}%
430 \ccUseProperty{imprint-sep}%
431 \ccUseProperty{imprint}%
432 \egroup
433 \clearpage
434 }%
435 %% predefined face and format Properties
436 \ccSetProperty{title-face}{\Huge\sffamily\bfseries}%
437 \ccSetProperty{title-format}{%
438 \bgroup
439 \ccVstructStart{Title}% PDF 2.0
440 \ccUseProperty{title-face}%
441 \ccUseComp{Title}\par
442 \ccVstructEnd{Title}% PDF 2.0
443 \egroup
444 \ccWhenComp{Subtitle}{\ccUseProperty{subtitle-format}}%
445 \ccWhenComp{TitleNote}{\ccUseProperty{title-note-format}}%
446 \ccGetComp{Statement}%
447 \vskip\baselineskip
448 }%
449 \ccSetProperty{title-note-face}{\large\sffamily}%
450 \ccSetProperty{title-note-format}{%
451 \bgroup
452 \ccUseProperty{title-note-face}%
453 \ccUseComp{TitleNote}%
454 \egroup
455 \par
456 }%
457 \ccSetProperty{subtitle-face}{\Large\sffamily\bfseries}%
458 \ccSetProperty{subtitle-format}{%
459 \bgroup
460 \ccUseProperty{subtitle-face}%
461 \ccUseComp{Subtitle}%
462 \egroup
463 \par
464 }%
465 %% Imprint
466 \ccSetProperty{imprint-face}{\footnotesize}%
467 \ccSetProperty{imprint-sep}{\ifhmode\par\fi\addvspace{\baselineskip}}%
468 \ccSetProperty{imprint}{%
469 \ccUseProperty{publisher}%
470 \ccGetComp{Qualification}%

```

```

471 \ccGetComp{Conversion}%%
472 \ccGetComp{CoverDesign}%%
473 \ccGetComp{CoverImage}%%
474 \ccGetComp{Lectorate}%%
475 \ccGetComp{QA}%%
476 \ccGetComp{Translator}%%
477 \ccGetComp{Appraiser}%%
478 \ccGetComp{Discussion}%%
479 \ccGetComp{Typesetter}%%
480 \ccGetComp{Print}%%
481 \ccGetComp{UsedFont}%%
482 \ccGetComp{DOI}%%
483 \ccGetComp{Keywords}%%
484 \ccUseProperty{imprint-sep}%
485 \ccGetComp{ISBNPreText}%
486 \ccGetComp{ISBN}%
487 \ccGetComp{EpubPreText}%
488 \ccGetComp{EISBN}%
489 \ccGetComp{EpubISBN}%
490 \ccUseProperty{imprint-sep}%
491 \ccGetComp{EnvDisclaimer}%
492 }%
493 \ccSetProperty{journal-meta}{%
494   \ccUseLabeledComp{Submitted}%
495   \ccUseLabeledComp{Received}%
496   \ccUseLabeledComp{Revised}%
497   \ccUseLabeledComp{Accepted}%
498   \ccUseLabeledComp{Published}%
499   \ccUseLabeledComp{Copyright}%
500   \ccUseLabeledComp{COIStatement}%
501   \ccUseLabeledComp{Keywords}%
502 }%
503 \ccSetProperty{licence}{%
504   \ccIfComp{LicenceLogo}{\includegraphics{\ccUseComp{LicenceLogo}}\par}{}%
505   \ccGetComp{LicenceText}%
506 }%
507 \ccSetProperty{copyright}{%
508   \ccIfComp{Copyright}
509     {\ccUseComp{Copyright}\par}
510     {\textcopyright\space\ccUseComp{Year}\space\ccUseComp{Publisher},\space\ccUseComp{PubPlace}
511       }\par}%
512 }%
513 \ccSetProperty{publisher}{%
514   \ccGetComp{PubDivInfo}%
515   \ccUseProperty{copyright}%
516   \ccGetComp{PubNote}%
517   \ccGetComp{PubWeb}%
518 }%
519 % Name Formats
520 \ccSetProperty{counted-meta-sep}{\ifnum\ccCurCount<\ccTotalCount\relax\vskip\baselineskip\fi}%
521   separator between multiple instances of the same meta datum
522 \ccSetProperty{counted-name-sep}{% Separator between multiple names; titlepage-specific override of
523   the same Property in coco-meta!
524   \ifnum\ccTotalCount>1\relax
525     \ifnum\ccCurCount<\ccTotalCount\relax
526       \ifnum\ccCurCount<\numexpr\ccTotalCount-1\relax
527         \ccUseProperty{name-sep}%
528       \else
529         \ccUseProperty{name-and}%
530       \fi
531     \fi
532   }%

```

```

529 \fi
530 }%
531 % Aliasses for different Roles, see coco-meta.sty for the actual Property values:
532 %% editors:
533 \ccPropertyLet{editor-cite-name-format} {role-cite-name-format}%
534 \ccPropertyLet{editor-short-cite-name-format} {role-short-cite-name-format}%
535 \ccPropertyLet{editor-full-name-format} {role-full-name-format}%
536 \ccPropertyLet{editor-pdfinfo-name-format} {role-pdfinfo-name-format}%
537 \ccPropertyLet{editor-correspondence-as-format} {role-correspondence-string-format}%
538 %
539 \ccPropertyLet{editor-list-print-format} {role-block-print-format}%
540 \ccPropertyLet{editor-list-cite-format} {role-block-cite-format}%
541 \ccPropertyLet{editor-list-short-cite-format} {role-block-short-cite-format}%
542 \ccPropertyLet{editor-list-pdfinfo-format} {role-block-pdfinfo-format}%
543 \ccPropertyLet{editor-list-correspondence-format} {role-block-correspondence-format}%
544 %% series-editors:
545 \ccPropertyLet{series-editor-cite-name-format} {role-cite-name-format}%
546 \ccPropertyLet{series-editor-short-cite-name-format} {role-short-cite-name-format}%
547 \ccPropertyLet{series-editor-full-name-format} {role-full-name-format}%
548 \ccPropertyLet{series-editor-pdfinfo-name-format} {role-pdfinfo-name-format}%
549 \ccPropertyLet{series-editor-correspondence-as-format} {role-correspondence-as-format}%
550 %
551 \ccPropertyLet{series-editor-list-print-format} {role-block-print-format}%
552 \ccPropertyLet{series-editor-list-cite-format} {role-block-cite-format}%
553 \ccPropertyLet{series-editor-list-short-cite-format} {role-block-short-cite-format}%
554 \ccPropertyLet{series-editor-list-pdfinfo-format} {role-block-pdfinfo-format}%
555 \ccPropertyLet{series-editor-list-correspondence-format} {role-block-correspondence-format}%
556 %% name Separators
557 \ccSetProperty{editor-suffix-sgl}{(Ed.)}%
558 \ccSetProperty{editor-suffix-pl}{(Eds.)}%
559 \ccSetProperty{editor-suffix}{%
560 \space
561 \ifnum\ccTotalCount=\@ne\relax
562 \ccUseProperty{editor-suffix-sgl}%
563 \else
564 \ccUseProperty{editor-suffix-pl}%
565 \fi
566 }%
567 % Biography
568 % those Properties control how (Role specific) Biography Blocks are formatted, i.e. the list of all
569 %   Biographies of a specific Role:
569 \ccSetProperty{role-bio-block-face}{}% face for the entire, role-specific, Biography Block
570 \ccSetProperty{role-bio-block-format}{{\ccUseProperty{role-bio-block-face}\ccUseComp{Biography
571 }}\par}% Format of the whole, Role specific, Biography Block
571 \ccPropertyLet{author-bio-block-format} {role-bio-block-format}% Override for single author meta
572 \ccPropertyLet{editor-bio-block-format} {role-bio-block-format}% Override for single editor meta
573 \ccPropertyLet{series-editor-bio-block-format} {role-bio-block-format}% Override for single
574 %   series editor meta info
574 % those Properties control how a (Role specific) Biography is formatted:
575 \ccSetProperty{role-biography-format}{{\bfseries\ccUseComp{FullName}:}\space\ccUseComp{Bio}\
576 \par}% Format of a single entry in the Role specific Biography
576 \ccPropertyLet{author-biography-format} {role-biography-format}% Override for single author meta
577 \ccPropertyLet{editor-biography-format} {role-biography-format}% Override for single editor meta
578 \ccPropertyLet{series-editor-biography-format} {role-biography-format}% Override for single
579 %   series editor meta info
579 \ccSetProperty{bio-output-format}{}%
580 \ccGetComp{AuthorBioBlock}%

```



```

581 \ccGetComp{EditorBioBlock}%
582 \ccGetComp{SeriesEditorBioBlock}%
583 }%
584 % Running headers
585 \ccSetProperty{run-book-title}{%
586 \ccIfComp{RunTitle}
587 {\ccUseComp{RunTitle}}
588 {\ccIfComp{ShortTitle}
589 {\ccUseComp{ShortTitle}}
590 {\ccIfComp{Title}{\ccUseComp{Title}}{No title given!}}}%
591 }%
592 \ccSetProperty{run-book-name}{%
593 \ccIfComp{RunNames}
594 {\ccUseComp{RunNames}}
595 {\ifmonograph
596 \ccIfComp{AuthorNameList}
597 {\ccUseComp{AuthorNameList}}
598 {no author defined!}%
599 \else
600 \ccIfComp{EditorNameList}
601 {\ccUseComp{EditorNameList}}
602 {no editor defined!}%
603 \fi}%
604 }%
605 }

```

## 5 Accessibility Features

### 5.1 Output Intent and ICC Profiles

```
606 \ccWhenAlly{%
```

First, we declare some Components that represent the three necessary parameters for the output intent:

```
607 \ccAddToType{Components}{titlepage}{%
```

**Component titlepage::IccProfileFile** holds the path (relative to the main tex file) and name of the .icc file.

```
608 \ccDeclareGlobalComponent{IccProfileFile}
```

**Component titlepage::IccComponents** holds the number of components in the color profile

```
609 \ccDeclareGlobalComponent{IccComponents}
```

**Component titlepage::IccIdentifier** holds the identifier of the color profile

```
610 \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):

```

611 \ifdefstring\cc@color@enc{cmyk}
612 {\def\cca@default@icc@comp{4}}
613 {\def\cca@default@icc@comp{3}}

```



```

614 \ifdefstring\cc@color@enc{cmyk}
615   {\def\cca@default@icc@iden{Coated FOGRA39}}
616   {\def\cca@default@icc@iden{sRGB IEC61966-2.1}}
617 \ccAddToType{Properties}{titlepage}{%

```

**Property titlepage::output-intent** 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,

**components** 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.)

```

618 \ccSetProperty{output-intent}{%
619   profile=\ccIfComp{IccProfileFile}{\ccUseComp{IccProfileFile}}{suppl/\cc@color@enc.icc};%
620   components=\ccIfComp{IccComponents}{\ccUseComp{IccComponents}}{\cca@default@icc@comp};%
621   identifier=\ccIfComp{IccIdentifier}{\ccUseComp{IccIdentifier}}{\cca@default@icc@iden}%
622 }%

```

The Component Handler which links the new Components to that Property is added to titlepage's **document-meta-hook**:

```

623 \ccAddToHook[titlepage]{document-meta-hook}{\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:

```

624 \ccAddToType{Components}{titlepage}{%

```

**Component titlepage::PDFaID** defines the PDF/A ID (Default: 2, meaning: PDF/A-2)

```

625 \ccDeclareGlobalComponent[2]{PDFaID}%

```

**Component titlepage::PDFaLevel** defines the PDF/A Level (Default: A, meaning PDF/A-2A)

```

626 \ccDeclareGlobalComponent[A]{PDFaLevel}%

```

**Component titlepage::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.

```

627 \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 **document-meta-hook**. The **conformance** DocumentInfo nodes are only written, if **neither PDFaID, nor PDFaLevel** is empty.

```

628 \ccAddToHook[titlepage]{document-meta-hook}{%
629   \ccIfCompEmpty{PDFaID}{\ccIfCompEmpty{PDFaLevel}}{}%
630   \edef\x{\noexpand\ccaSetDocinfo{conformance}}{%
631     pdfaid=\ccUseComp{PDFaID};%
632     level=\ccUseComp{PDFaLevel}%

```

```
633 \ccIfCompEmpty{PDFUAID}{\pdfuaid=\ccUseComp{PDFUAID}}}%  
634 \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.

```
635 \ccaAddRolemap{Title}{H1}  
636 \ccaAddRolemap{Titlepage}{Div}
```

```
637 }%ccWhenAlly
```

```
638 %</title>
```

## Modul 10

# coco-floats.dtx

This module provides handlers for floating objects like tables and figures common to all CoCoTeX projects

```

24 %<floats>

25 %%
26 %% module for CoCoTeX that extends floating objects.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-floats}
34   [2024/03/23 0.4.1 CoCoTeX floats module]
35 \DeclareOptionX{nofigs}{\global\let\ccf@no@figs\relax}
36 \ProcessOptionsX

```

## 1 Package Setup

### 1.1 Hard requirements

```

37 \RequirePackage{coco-common}
38 \RequirePackage{rotating}
39 \RequirePackage{grffile}
40 \RequirePackage{footnote}
41 \RequirePackage[Export]{adjustbox}
42 \usepackage{stfloats}
43 \setcounter{dblbotnumber}{5}

```

### 1.2 Document Class Option overrides

for automatic typesetting and float positioning, we set very high tolerances in macros from L<sup>A</sup>T<sub>E</sub>X's standard

## 2 .clo

files:

```

44 \def\topfraction{0.9}
45 \def\textfraction{0.1}
46 \def\bottomfraction{0.8}

```

```

47 \def\totalnumber{8}
48 \def\topnumber{8}
49 \def\bottomnumber{8}
50 \def\floatpagefraction{0.8}
51 \@fptop\z@
52 \@fpbot\@flushglue

```

## 2.1 Internal registers

Some reserved box registers for measuring, the first one, `\ccf@floatbox`, is for the whole float, the second one, `\ccf@sub@box`, is for a single sub-float.

```

53 \newbox \ccf@floatbox
54 \newbox \ccf@sub@box

```

Internal counters: `\ccSubFloatCnt` counts the sub-floats within a single float, `\ccf@int@cnt` is the internal global counter for all floats.

```

55 \newcount\ccSubFloatCnt \ccSubFloatCnt=\z@\relax
56 \newcount\ccf@int@cnt \ccf@int@cnt=\z@
57 \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
- `\ccf@sub@sep` is the space between sub-floats
- `\ccf@total@width` stores the cumulated overall width of the entire float
- `\ccf@calc@width` is an internal dimension used to calculate the ratio between multiple sub-floats that should be scaled to the same height
- `\ccf@total@height` is the overall height of a float
- `\ccf@total@depth` is the overall depth of a float

```

58 \newdimen\ccf@sub@maxheight \ccf@sub@maxheight=\z@\relax
59 \newdimen\ccf@sub@sep \ccf@sub@sep=\fboxsep\relax
60 \newdimen\ccf@total@width \ccf@total@width=\textwidth\relax
61 \newdimen\ccf@total@height \ccf@total@height=\textwidth\relax
62 \newdimen\ccf@total@depth \ccf@total@depth=\textwidth\relax
63 \newdimen\ccf@calc@width \ccf@calc@width=\ccf@total@width\relax

```

Those two dimensions are used to pass the `intext-skip` and `float-skip` Properties to the render engine for spacing above and below the float, respectively.

```

64 \newskip\ccf@sep@top \ccf@sep@top=\z@\relax
65 \newskip\ccf@sep@bottom \ccf@sep@bottom=\z@\relax

```

Internal dimensions for the horizontal margins (right, left, inner and outer, respectively)

```

66 \newdimen\ccf@margin@r \ccf@margin@r=\z@\relax
67 \newdimen\ccf@margin@l \ccf@margin@l=\z@\relax
68 \newdimen\ccf@margin@i \ccf@margin@i=\z@\relax
69 \newdimen\ccf@margin@o \ccf@margin@o=\z@\relax

```

Locally adjustable switch to allow captions to break across pages

```

70 \newif\ifccf@break@capt \@ccf@break@captfalse

```

String definitions for Property value comparisons

```

71 \def\ccf@str@bottom{bottom}
72 \def\ccf@str@top{top}

```

## 2.2 AtBeginDocument hook

```

73 \AtBeginDocument{%

```

implementing the `nofigs` option, doing some minor adjustments to the `htmltabs` package and store the final definition of `includegraphics`.

```

74 \ifx\ccf@no@figs\relax
75   \renewcommand\includegraphics[2][]{}%
76 \fi
77 \global\let\ccf@ltx@includegraphics\includegraphics

```

Adjustments to the `htmltabs` package, if it is used:

```

78 \@ifpackageloaded{htmltabs}
79   {\global\let\cc@uses@htmltabs\relax
80   \def\ht@adjust@linewidth{%
81     \advance\ht@h@offset\leftskip
82     \advance\ht@h@offset\@totalleftmargin
83     \advance\linewidth-\rightskip
84   }%
85   }{}%

```

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`.

```

86 \gappto\@endfloatbox{%
87   \global\ccf@total@height=\ht\@currbox\relax%
88   \global\ccf@total@depth=\dp\@currbox\relax%
89 }%
90 }%

```

## 3 Internal macros

### 3.1 Generic resetter

`\ccf@reset@defaults` resets the parameters for sub-floats.

- #1 the caption type (e.g., `figure`, `table`)
- #2 abbreviation of the caption list (e.g., standard L<sup>A</sup>T<sub>E</sub>X uses `lof` for the List of Figures, `lot` for the List of Tables)

```

91 \def\ccf@reset@defaults{%
92   \global\ccSubFloatCnt=\z@
93   \global\ccf@total@width=\z@
94   \global\let\ccf@has@capt@top\@undefined
95   \global\let\ccf@has@capt@bottom\@undefined
96   \global\let\ccf@has@subcapt@top\@undefined
97   \global\let\ccf@has@subcapt@bottom\@undefined

```

```

98 \global\let\ccf@sub@contentsline@store\@empty
99 \global\ccf@sub@maxheight=\z@\relax
100 \@tempcnta=\z@\relax
101 \cc@reset@components{\cc@cur@cont}%
102 \let\ccf@prefix\@empty
103 \let\ht@cur@element\ccfCapType
104 \global\let\ccf@current@class\relax
105 }

```

### 3.2 Internal macros that handle Attributes

`\ccf@get@attr` invokes the parser for the optional argument of float environments.

#1 is the content of the optional argument,  
 #2 is the caption type.

```

106 \def\ccf@get@attr#1#2{%
107 \if!#1!\else
108 \ccParseAttributes{#2}{#1}%
109 \ccIfAttr{#2}{class}
110 {\global\letcs\ccf@current@class{\cc@#2@attr@class}%
111 \ccUseStyleClass{default}{\ccfCapType}%
112 \expandafter\ccUseStyleClass\expandafter{\csname cc@#2@attr@class\endcsname}{\ccfCapType}}
113 {}%
114 \ccIfAttr{#2}{break-caption}{\@ccf@break@capttrue}{}%
115 \fi
116 \ccf@get@pos{#2}}

```

`\ccf@get@pos` is the handler for determining the floating 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 caption type.

```

117 \def\ccf@get@pos#1{%
118 \ccIfAttr{#1}{float-pos}
119 {\letcs\ccf@floatpos{\cc@#1@attr@float-pos}}
120 {\let\ccf@floatpos\@empty}%
121 \def\@tempa{h!}\ifx\ccf@floatpos\@tempa\let\ccf@floatpos\@empty\fi
122 \def\@tempa{h}\ifx\ccf@floatpos\@tempa\def\ccf@floatpos{htpb!}\fi
123 \ifx\ccf@do@dbl\relax
124 \ifx\ccf@floatpos\@empty\def\ccf@floatpos{htpb!}\fi% 11514
125 \linewidth\dimeexpr2\columnwidth+\columnsep\relax
126 \hsize\linewidth\relax
127 \fi
128 \ccIfAttrIsStr{#1}{orientation}{landscape}
129 {\linewidth\textheight
130 \hsize\linewidth
131 \def\ccf@floatpos{p}}
132 {}

```

`\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.

```

133 \def\ccf@set@env{%
134 \ifx\ccf@floatpos\@empty
135 \let\ccf@begin@env\bgroup
136 \let\ccf@end@env\egroup
137 %\ifhmode\par\fi

```

```

138 \else
139   \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
140   {\edef\ccf@env@name{sideways\ccfCapType}%
141    \edef\ccf@begin@env{\noexpand\begin{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}%
142    \edef\ccf@end@env{\noexpand\end{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}}
143   {\edef\ccf@env@name{\ifx\ccf@do@dbl\relax db\fi float}%
144    \edef\ccf@begin@env{\expandafter\noexpand\csname @x\ccf@env@name\endcsname {\ccfCapType}[\ccf@floatpos]}%
145    \edef\ccf@end@env{\expandafter\noexpand\csname end@\ccf@env@name\endcsname}}%
146 \fi}

```

`\ccf@debug` prints some debug information to `stdout` for a single float that has the Attribute `debug` set.

```

147 \def\ccf@debug#1{%
148   \ccIfAttr{#1}{debug}
149   {\message{^^}[CoCo Float Debug]^^}
150    Textheight:\space\the\textheight^^}
151    Type:\space\space\space\space\space\space\space\space\cc@cur@cont^^}
152 \ifx\ccfCapType\cc@str@figure
153   Path: \space\space\space\space\space\space\space\ccf@fig@path^^}
154 \fi
155   Class:\space\space\space\space\space\space\space\ccf@current@class^^}
156   Floatpos:\space\space\space\ccf@floatpos^^}
157   Environ:\space\space\space\space\expandafter\noexpand\ccf@begin@env...\expandafter\noexpand
158     \ccf@end@env^^}
159   Subfloat:\space\space\space \the\ccSubFloatCnt^^}
160 \ifnum\ccSubFloatCnt=\z@
161   Width:\space\space\space\space\space\space\space\the\ccf@total@width^^}
162   Height:\space\space\space\space\space\space\space\the\ccf@total@height^^}
163   Depth:\space\space\space\space\space\space\space\the\ccf@total@depth^^}
164 \else
165   Width \the\ccSubFloatCnt:\space\space\space\space\space\space\space\expandafter\meaning\csname
166     ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcsname^^}
167   Height \the\ccSubFloatCnt:\space\space\space\space\space\space \expandafter\meaning\csname ccf@
168     \cc@cur@cont @height-\the\ccSubFloatCnt\endcsname^^}
169   Depth \the\ccSubFloatCnt:\space\space\space\space\space\space\space\expandafter\meaning\csname
170     ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcsname^^}
171 \fi}}}}

```

`\ccf@get@seps` determines the top and bottom skips dependent on float position and orientation

```

168 \def\ccf@get@seps{%
169   \ifx\ccf@floatpos@empty
170     \expandafter\ccf@sep@top\dimexpr\ccUseProperty{intext-skip-top}\relax%
171   \else
172     \expandafter\ccf@sep@top\dimexpr\ccUseProperty{float-skip-top}\relax%
173   \fi
174   \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}{}
175   {\ifx\ccf@floatpos@empty
176     \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{intext-skip-bottom}\relax%
177   \else
178     \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{float-skip-bottom}\relax%
179   \fi}}

```

`\ccf@set*@sep` Hooks to apply top and bottom skips, respectively.

```

180 \def\ccf@set@top@sep{\addvspace{\ccf@sep@top}}
181 \def\ccf@set@bot@sep{\addvspace{\ccf@sep@bottom}}

```

## 4 Float Container and Component Declarations

`\ccfMakeComp` is a shortcut for float Component declarations. #1 is the generic name of the Component.

```
182 \def\ccfMakeComp#1{%
183   \cc@def@counted@comp{#1-\the\ccSubFloatCnt}{#1}{}}%
184 }
```

`\ccfMakeCompL` is a shortcut to declare Float Components together with their *list-of* overrides. #1 is the generic name of the Component.

```
185 \def\ccfMakeCompL#1{%
186   \ccfMakeComp{#1}%
187   \ccfMakeComp{Listof#1}}
```

`\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.

```
188 \def\ccf@set@hsize{%
189   \expandafter\ccf@sub@sep\ccUseProperty{sub-float-sep}\relax%
190   \global\ccf@total@width=\hsize\relax
191   \expandafter\ccf@margin@l\ccUseProperty{margin-left}\relax
192   \expandafter\ccf@margin@r\ccUseProperty{margin-right}\relax
193   \expandafter\ccf@margin@i\ccUseProperty{margin-inner}\relax
194   \expandafter\ccf@margin@o\ccUseProperty{margin-outer}\relax
195   \ccf@set@margins
196   \global\advance\ccf@total@width-\ccf@margin@r\relax
197 }
```

`\ccf@set@margins` realises inner and outer margins via the left and right margins.

```
198 \def\ccf@set@margins{%
199   \ccTestPage
200   \if@cc@odd
201     \advance\ccf@margin@l\ccf@margin@i
202     \advance\ccf@margin@r\ccf@margin@o
203   \else
204     \advance\ccf@margin@l\ccf@margin@o
205     \advance\ccf@margin@r\ccf@margin@i
206   \fi
207 }
```

### Container float

```
208 \ccDeclareContainer{float}{%
209   \ccDeclareType{Components}{%
210     \def\cc@counted@comp@scheme##1{##1-\the\ccSubFloatCnt}%
211     \ccfMakeCompL{Caption}%
212     \ccfMakeCompL{Legend}%
213     \ccfMakeCompL{Source}%
214     \ccfMakeCompL{Number}%
215     \ccfMakeComp{RefLabel}%
216     \ccfMakeComp{AltText}% neu: 2023-06-08; TODO: muss noch implementiert werden
217     \ccfMakeComp{ListofEntry}%
218   }%
219   \ccDeclareType{Properties}{}%
220 }
```



`\ccDeclareFloat` is the user-level macro used to (re-)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 the tagging Role (*optional*, defaults to Div)
- #4 caption type (e.g., `table`, `figure`)
- #5 list (e.g., `lot`, `lof`)
- #6 Property list

```

221 \def\ccDeclareFloat{\cc@opt@empty\ccf@declare@float}
222 \def\ccf@declare@float[#1]#2{\ifnextchar[{\@ccf@declare@float[#1]#2[
    Div]}]}%
223 \long\def\@ccf@declare@float[#1]#2[#3]#4#5#6{%
224   \def\ccf@parent{#1}%

```

If the float Container has already been declared, we only load its parent's Properties and Containers (if any), and add the override Properties to the Container's Property List. Otherwise, we would re-load the system's defaults and override the Properties of the earlier Declaration.

```

225 \ifcsdef{cc@container@#2}{%
226   \ccPackageInfo{Floats}{\Appending to '#2'}%
227   \ifx\ccf@parent\@empty\else
228     \ccPackageError{Float}{Type}
229     {Attempt to change parent of pre-existing float^^}Container '#2'
230     {You cannot use the optional argument of \string\ccDeclareFloat\space for pre-existing^^}
    %
231 float containers!^^}^^}%
232 Use \string\ccAddToType{<Type>}{#2}{<code>}\space to alter the #2 container!}
233 \fi
234 \ccAddToType{Properties}{#2}{#6}%

```

Other than Properties, the Float's default caption type or list-of handler may also be overridden by a re-definition.

```

235 \ccAddToType{FloatEnvInfo}{#2}{%
236   \def\ccfCapType{#4}%
237   \def\ccf@cap@list@type{#5}%
238 }%
239 }{%

```

Otherwise, we declare a new Container and invoke all the Initializers.

```

240 \ccDeclareContainer{#2}{%
241   \ccPackageInfo{Floats}{\Declaring float '#2'}%
242   \ifx\ccf@parent\@empty
243     \ccInherit{Properties,Components}{float}
244   \else
245     \ccInherit{Properties,Components}{\ccf@parent}
246   \fi
247   \ccDeclareType{FloatEnvInfo}{%
248     \ccSetContainer{#2}%
249     \def\ccfCapType{#4}%
250     \def\ccf@cap@list@type{#5}%
251   }% /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.

```

252 \ccDeclareEnv[#2]{\ccf@float}{\endccf@float}%
253 \ccDeclareEnv[#2*]{\if@twocolumn\let\ccf@do@dbl\relax\else\fi\ccf@float}{\if@twocolumn\let\ccf@do@dbl\relax\fi\endccf@float}%

```

```

254 \ccDeclareType{Components}{%
255 \ccUseProperty{float-handler}%
256 }%

```

Generating the Handlers for the list-of entries and define the corresponding `l@` macros

```

257 \ccf@generate@listof@handlers{#5}{#4}{#2}%
258 \bgroup
259 \def\cc@cur@cont{#2}%
260 \cc@init@l@[list-of]{#4}{0}{#4}% Generate listof-Entries for first level floats
261 \cc@init@l@[list-of]{#5}{1}{sub#4}% Generate listof-Entries for sub-floats
262 \egroup
263 \ccDeclareType{Properties}{#6}%
264 }% /container
265 }% /ifcsdef{cc@container@#2}
266 \ifstrequal{Table}{#3}{}
267 {\ifstrequal{Figure}{#3}{}
268 {\ccaAddRolemap{#2}{#3}}}%
269 }

```

`\ccf@generate@listof@handlers` generates handlers for listof-entries.

- #1 is the file ending
- #2 is the caption type
- #3 is the Container name

```

270 \def\ccf@generate@listof@handlers#1#2#3{%

```

`cc@<list>@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.

```

271 \expandafter\gdef\csname cc@#1@extract@data\endcsname##1##2##3##4{%
272 \ccSetContainer{#3}%
273 \ccEvalType[#3]{Properties}%
274 \ccDeclareComponent{ListofCaption}{}{}%
275 \ccDeclareComponent{ListofLegend}{}{}%
276 \ccDeclareComponent{ListofSource}{}{}%
277 \ccDeclareComponent{ListofNumber}{}{}%
278 \ccDeclareComponent{ListofPage}{}{}%
279 \ccComponent{ListofPage}{\ccUseProperty{list-of-page-face}##4}%
280 \cc@expand@l@contents{##3}{#3}{Listof}{Caption}%
281 \cc@format@number{list-of-}{Listof}{##1}%
282 }%

```

`\csname cc@<list>@print@entry\endcsname` 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.

```

283 \expandafter\gdef\csname cc@#1@print@entry\endcsname##1{%
284 \bgroup
285 \ccUseHook{list-of-before-hook-##1}%
286 \ccUseProperty{list-of-before-entry}%
287 \ccUseProperty{list-of-block}%
288 \ccUseHook{list-of-after-hook-##1}%
289 \ccUseProperty{list-of-after-entry}%

```

```

290 \egroup}%
291 }

```

`\ccf@addcontentsline` fork of L<sup>A</sup>T<sub>E</sub>X's `\addtocontents` macro.

```

292 \def\ccf@addcontentsline{%
293   \ccWhenComp{ListofEntry}{%
294     \protected@write\auxout
295       {\ccGobble}%
296       {\string\@writefile{\ccf@cap@list@type}
297         {\protect\ccContentsline
298           {\ifnum\ccSubFloatCnt>\z@\ccIfAttr{\ccfCapType}{subfloat}{sub}}{\fi\ccfCapType}
299           {\ccUseComp{ListofEntry}}
300           {\thepage}
301           {\@currentHref}\protected@file@percent}}\relax}}

```

`\ccf@check@empty` is a wrapper for CoCoTeX kernel's `\cc@check@empty`

```

302 \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.

```

303 \def\ccf@compose@listof{%
304   \ccf@check@empty{Number}%
305   \ccf@check@empty{Caption}%
306   \ccf@check@empty{Legend}%
307   \ccf@check@empty{Source}%
308   \let\ccf@listof@entry\relax
309   \ccWhenComp{ListofCaption}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofCaption}}{\ccUseComp{ListofCaption}}}%
310   \ccWhenComp{ListofNumber}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofNumber}}{\ccUseComp{ListofNumber}}}%
311   \ccWhenComp{ListofLegend}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofLegend}}{\ccUseComp{ListofLegend}}}%
312   \ccWhenComp{ListofSource}{\csgappto{\ccf@listof@entry}{\string\ccComponent{ListofSource}}{\ccUseComp{ListofSource}}}%
313   \ifx\ccf@listof@entry\relax\else
314     \bgroup
315     \ccGobble
316     \protected@edef\ccf@listof@entry{\ccf@listof@entry}%
317     \ccComponentEA{ListofEntry}{\ccf@listof@entry}%
318     \egroup
319   \fi
320 }%

```

`\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.

```

321 \def\ccf@write@listof{%
322   \ccUnlessAttr{\ccfCapType}{nolist}
323   {\ifnum\ccSubFloatCnt=\z@\relax
324     \ccIfAttr{\ccfCapType}{subfloat}
325     {\ccSubFloatCnt=\z@\relax
326       \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
327       {\ccf@addcontentsline}}%
328     {\ccf@addcontentsline}%
329   \else
330     \ccIfAttr{\ccfCapType}{subfloat}{\ccf@addcontentsline}%

```

```

331 \fi}%
332 }

```

## 5 Label and Referencing mechanisms

### 5.1 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.

```

333 \def\ccf@create@counter{%
334 \ccIfAttrIsSet{\ccfCapType}{nonnumber}{}
335 {\ccUnlessComp{Number}
336 {\ccIfPropVal{numbering}{auto}
337 {\ccIfAttr{\ccfCapType}{subfloat}
338 {\ifnum\ccSubFloatCnt=\z@\relax
339 \ccf@set@top@counter%
340 \else
341 \ccIfPropVal{sub-numbering}{auto}
342 {\ccf@set@subcounter}{}%
343 \fi}
344 {\ccf@set@top@counter}}}}}%

```

`\ccf@set@top@counter` generates first level float counter.

```

345 \def\ccf@set@top@counter{%
346 \ccWhenComp{Caption}{%
347 \global\expandafter\advance\csname c@\ccfCapType\endcsname\@ne\relax
348 \ccdefFromProperty\ccf@name@prefix{auto-number-prefix}%
349 \ccdefFromProperty\ccf@name@sep{auto-number-prefix-sep}%
350 \protected@edef\@tempa{\ccf@name@prefix\ccf@name@sep\expandafter\the\csname c@\ccfCapType\endcsname}%
351 \ccComponentEA{Number}{\@tempa}%
352 }%
353 }

```

`\ccf@set@subcounter` generates second level counters for numbered sub-floats. #1 is the sub-float counter.

```

354 \def\ccf@set@subcounter{%
355 \ccSetPropertyVal{float-number}{\csname cc@\cc@cur@cont @Number-0\endcsname}%
356 \ccSetPropertyVal{sub-number}{%
357 \begingroup
358 \expandonce{\ccUseProperty{sub-number-face}}%
359 \relax\ccUseProperty{sub-number-before}%
360 \csname @\ccUseProperty{sub-number-style}\endcsname{\the\ccSubFloatCnt}%
361 \ccUseProperty{sub-number-after}%
362 \endgroup}%
363 \ccComponent{Number}{\ccUseProperty{sub-number-format}}%
364 }

```

### 5.2 Generation of L<sup>A</sup>T<sub>E</sub>X Labels

`\ccfCreateLabel` creates labels

```

365 \def\ccfCreateLabel{%
366   \ccIfComp{Number}
367   {\def\cc@fallback@anchor{%
368     \ccGobble
369     \ccdefFromComp\@currentlabel{Number}%
370     \ccdefFromComp\@currentlabelname{ListofCaption}}%
371   \def\cc@labelname@comp{Caption}}
372   {\def\cc@fallback@anchor{\phantomsection}}%
373   \expandafter\ccCreateLabel\expandafter{\ccfCapType}}

```

## 6 Processing the Float

### 6.1 Common Float and Sub-Float Environments

`\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)

```

374 \def\ccf@float{\ccopt@empty\@ccf@float}
375 \def\@ccf@float[#1]{%
376   \par
377   \begingroup
378   \@cc@is@finalfalse
379   \global\advance\ccf@int@cnt\@ne
380   \ccEvalType{FloatEnvInfo}%
381   \ccf@reset@defaults
382   \ccToggleCountedConditionals
383   \ccEvalType{Properties}%
384   \ccf@get@attr{#1}{\ccfCapType}%
385   \ccf@set@hsize
386   \ccf@get@seps
387   \ccEvalType{Components}%
388   \ccUseProperty{before-float}%
389   \ccf@set@env
390   \ifx\ccf@floatpos\@empty\else\savenotes\fi
391   \ignorespaces
392   \@cc@is@finaltrue
393 }

```

`\endccf@float` is the end of the common float environment.

```

394 \def\endccf@float{%
395   \ccf@begin@env
396   \@cc@is@finalfalse
397   \ccf@set@top@sep
398   \ccf@int@sub@flt@cnt=\ccSubFloatCnt\relax
399   \ccSubFloatCnt=\z@\relax
400   \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
401     {\ccf@create@counter
402       \ccf@compose@listof}%
403   \ccSubFloatCnt=\ccf@int@sub@flt@cnt\relax
404   \ccf@test@caption{0}{}{top}%
405   \ccf@test@caption{0}{}{bottom}%
406   \bgroup

```

```

407     \@cc@is@finaltrue
408     \ccaStructStart{\cc@cur@cont}%
409     \edef\ccf@parstruct{id{\ccaGetCurStruct{idx}}}%
410     \hsize\ccf@total@width
411     \ccf@process
412     \ccaStructEnd{\cc@cur@cont}%
413     \par
414     \egroup
415     \ccSavePage
416     \ccf@set@bot@sep
417     \ccf@end@env
418     \ccf@debug{\ccfCapType}%
419     \ifx\ccf@floatpos\@empty\else\spewnotes\fi
420 \endgroup
421 \immediate\write\@auxout
422   {\string\expandafter\string\gdef\string\csname\space cc-float-\the\ccf@int@cnt-dimens\string
     \endcsname{%
423     {\the\ccf@total@width}%
424     {\the\ccf@total@height}%
425     {\the\ccf@total@depth}%
426   }}%
427 \global\let\ccf@current@class\relax
428 }

```

**\ccSubFloat** is the user-level environment for sub-floats

```

429 \def\ccSubFloat{%
430   \ifx\ccf@is@subfloat\relax
431     \PackageError{coco-floats.sty}{Nested ccSubFloats detected!}{You cannot (yet) nest a ‘
       ccSubFloat’ environment into another ‘ccSubFloat’ environment!}%
432   \else
433     \global\let\ccf@is@subfloat\relax
434     \global\advance\ccSubFloatCnt\@ne
435   \fi
436   \global\cslet\ccf@made@label@for@\the\ccSubFloatCnt\relax
437   \ignorespaces}

```

**\endccSubFloat** is the end of the sub-float environment

```

438 \def\endccSubFloat{%
439   \ccUseProperty{subfloat-handler}%
440   \expandafter\xdef\csname ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcsname{\the\wd\
     ccf@sub@box}%
441   \expandafter\xdef\csname ccf@\cc@cur@cont @height-\the\ccSubFloatCnt\endcsname{\the\ht\
     ccf@sub@box}%
442   \expandafter\xdef\csname ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcsname{\the\dp\
     ccf@sub@box}%
443   \@tempdima=\dimexpr\the\ht\ccf@sub@box+\the\dp\ccf@sub@box\relax
444   \@tempdimb=\dimexpr\the\wd\ccf@sub@box\relax
445   \ifdim\@tempdima>\ccf@sub@maxheight\relax
446     \global\ccf@sub@maxheight=\@tempdima\relax
447   \fi
448   \global\setbox\ccf@sub@box\box\voidb@x
449   \global\let\ccf@is@subfloat\@undefined
450   \aftergroup\ignorespaces
451 }

```

## 6.2 Processing the Contents of the Float Environment

`\ccf@process` prints the contents of a float environment.

```

452 \def\ccf@process{%
453   \ifx\ccf@has@capt@top\@empty\leavevmode\fi
454   \ccf@make@outer@caption{top}%
455   \ifnum\the\ccSubFloatCnt=z\relax
456     \bgroup\advance\hsize-\ccf@margin@l
457     \@cc@is@finaltrue
458     \ccUseProperty{float-render}%
459     \egroup
460   \else
461     \ccf@test@subcapt
462     \@cc@is@finalfalse
463     \ccf@calc@sameheight
464     \def\ccf@prefix{sub}%
465     \ifx\ccf@has@subcapt@top\@empty\ccf@calc@row@ht{top}\fi%
466     \ifx\ccf@has@subcapt@bottom\@empty\ccf@calc@row@ht{bottom}\fi%
467     \@cc@is@finaltrue
468     \ccUseProperty{subfloat-render}%
469     \let\ccf@prefix\@empty
470   \fi
471   \ccf@make@outer@caption{bottom}%
472 }

```

## 6.3 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  
 #1 indicates if the caption belongs to the whole float (`capt`) or a sub-float (`subcapt`)  
 #1 `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  $\LaTeX$  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!

```

473 \def\ccf@test@caption#1#2#3{%
474   \@cc@is@finalfalse
475   \setbox\cc@tempboxa\hbox{\ccGobble\ccSubFloatCnt=0#1\relax\ccUseProperty{#2caption-#3}\relax}%
476   \setbox\cc@tempboxb\hbox{\ccGobble\ccSubFloatCnt=m@ne\relax\ccUseProperty{#2caption-#3}\relax}%
477   %
478   \edef\my@wda{\expandafter\strip@pt\wd\cc@tempboxa sp}%
479   \edef\my@wdb{\expandafter\strip@pt\wd\cc@tempboxb sp}%
480   \ifdim\my@wda>\my@wdb\relax
481     \expandafter\global\expandafter\let\csname ccf@has@#2capt@#3\endcsname\@empty
482   \fi
483   \@cc@is@finaltrue
484 }

```

`\ccf@test@subcapt` tests if the current float has any top or bottom captions that need to be printed

```

484 \def\ccf@test@subcapt{%
485   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%

```

```

486 \ccf@test@caption{\the\@tempcnta}{sub}{top}%
487 \ccf@test@caption{\the\@tempcnta}{sub}{bottom}%
488 }%
489 }

```

**\ccf@capt@top@offset** determines the spacing inserted **above both captions**.

```

490 \def\ccf@capt@top@offset#1{%
491 \ccIfStrEqual{#1}{top}{}{%
492 \par\ifccf@break@capt\else\nopagebreak\fi%
493 \expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-bottom}\relax%
494 \advance\@tempskipa\dimexpr-\topskip+\dp\strutbox\relax
495 \ifccf@break@capt\advance\@tempskipa\dimexpr-\baselineskip-\ht\strutbox+\topskip\relax\fi
496 \ifx\ccf@has@subcapt@bottom\empty
497 \ifnum\the\ccSubFloatCnt=\z@
498 %% subcapt-bot exists and capt-bot is rendered
499 \advance\@tempskipa\dimexpr\dp\strutbox\relax
500 \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-bottom}\relax%
501 \fi
502 \fi
503 \vskip\@tempskipa
504 \leavevmode
505 }}

```

**\ccf@capt@bottom@offset** determines the spacing inserted **below the captions**.

```

506 \def\ccf@capt@bottom@offset#1{%
507 \ccIfStrEqual{#1}{top}
508 {\@tempskipa=\z@\relax
509 \expandafter\advance\expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-top}%
510 \ifnum\the\ccSubFloatCnt=\z@\relax
511 \ifx\ccf@has@subcapt@top\empty
512 %% subcapt-top exists and capt-top is rendered
513 \advance\@tempskipa\dimexpr\ht\strutbox-\topskip-\p@\relax
514 \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-top}\relax%
515 \else
516 \advance\@tempskipa\dimexpr-\dp\strutbox\relax
517 \fi
518 \fi
519 \vskip\@tempskipa
520 \par\ifccf@break@capt\else\nopagebreak\fi}
521 {\ifnum\the\ccSubFloatCnt>\z@\relax
522 \vskip\dp\strutbox
523 \fi}}

```

**\ccf@make@caption** prints the caption.

- #1 is the placement (top, bottom)
- #2 is the vertical alignment (top, middle, bottom)

```

524 \long\def\ccf@make@caption#1#2{%
525 \ccf@capt@top@offset{#1}%
526 \ifnum\the\ccSubFloatCnt=\z@\relax
527 \def\ccf@caption@box{%
528 \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
529 {\setbox\@tempboxa\vbox\bgroup\hsize\textheight}
530 {\hskip\ccf@margin@l%
531 \setbox\@tempboxa\vbox\bgroup\advance\hsize-\ccf@margin@l}%

```



```

532 }%
533 \else
534 \expandafter\cc@tempskipa\csname ccf@capt@row@height@#1\endcsname\relax
535 \expandafter\advance\expandafter\cc@tempskipa\dimexpr-\baselineskip+\topskip\relax
536 \def\ccf@caption@box{\setbox\@tempboxa\ vbox to \cc@tempskipa\bgroup}%
537 \fi
538 \ccf@caption@box%
539 \ccIfStrEqual{#2}{top}{\if@ccf@break@capt\else\vss\fi}%
540 \ccUseProperty{\ccf@prefix caption-face}%
541 \ccUseProperty{\ccf@prefix caption-face-#1}%
542 \ccaStructStart{Caption}%
543 \cc@topstrut\ccUseProperty{\ccf@prefix caption-#1}\strut%
544 \ccaStructEnd{Caption}%
545 \ifx\ccf@measure\relax\else
546 \ccIfPropVal{label-pos}{#1}{%
547 \ccfCreateLabel%
548 \ccf@write@listof%
549 }%
550 \fi
551 \ccIfStrEqual{#2}{bottom}{\if@ccf@break@capt\else\vss\fi}%
552 \egroup%
553 \if@ccf@break@capt\unvbox\@tempboxa\else\box\@tempboxa\fi%
554 \ccf@capt@bottom@offset{#1}%
555 }

```

`\ccf@make@outer@caption` is a shell for the outer captions. #1 is the placement (top or bottom)

```

556 \def\ccf@make@outer@caption#1{%

```

now, we print the actual captions, if they contain contents.

```

557 \expandafter\ifx\csname ccf@has@capt@#1\endcsname\@empty
558 \setbox\z@\vbox{%
559 \cc@is@finalfalse
560 \let\ccf@measure\relax
561 \ccGobble
562 \ccSubFloatCnt\z@
563 \ccf@make@caption{#1}{top}%
564 }%
565 \immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\space ccFloat\the\
ccf@int@cnt Cap#1\string\endcsname{\the\dimexpr \ht\z@+\dp\z@\relax}}%
566 \bgroup
567 \cc@is@finaltrue
568 \savenotes
569 \if@ccf@break@capt\else\nopagebreak\fi
570 \ccSubFloatCnt\z@
571 \ccf@make@caption{#1}{top}%
572 \spewnotes
573 \egroup
574 \ccIfStrEqual{#1}{top}{\if@ccf@break@capt\else\nopagebreak\fi}}%
575 \fi}

```

`\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`.

```

576 \long\def\ccfRenderSubFloats#1#2{%
577 \leavevmode
578 \savenotes
579 \ifnum#1>\@ne\hfill\fi

```

```

580 \vtop\bgroup
581   \expandafter\hsize\csname cc@\cc@cur@cont @res@width-#1\endcsname\relax
582   \let\includegraphics\ccf@includesubgraphics
583   \leavevmode
584   \ccf@render@sub{#1}{#2}%
585 \egroup
586 \spewnotes
587 }

```

### depending

```

588 \def\ccf@make@subcaption#1{%
589   \expandafter\ifx\csname cc@has@\ccf@prefix capt@#1\endcsname\empty
590   \ccf@make@caption{#1}{\ccUseProperty{\ccf@prefix caption-valign-#1}}%
591   \fi}

```

`\ccf@render@sub` renders a single sub-float. For the arguments, see `\ccfRenderSubFloats`, above.

```

592 \long\def\ccf@render@sub#1#2{%
593   \ccSubFloatCnt=#1\relax
594   \ccf@make@subcaption{top}%
595   \bgroup\strut\ccUseComp{#2}\strut\par\egroup%
596   \ccf@make@subcaption{bottom}}

```

`\ccf@calc@row@ht` calculates the heights of all captions in the same row.

#1 determines if the `top` or `bottom` row is calculated.

```

597 \def\ccf@calc@row@ht#1{%
598   \@tempcnta\z@
599   \@tempdima\z@
600   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
601     \setbox\z@\vbox{%
602       \ccSubFloatCnt\@tempcnta\relax
603       \expandafter\hsize\expandafter\dimexpr\csname cc@\cc@cur@cont @res@width-\the\@tempcnta\endcsname\relax
604       \ccGobble
605       \ccUseProperty{\ccf@prefix caption-face}%
606       \ccUseProperty{\ccf@prefix caption-face-#1}%
607       \leavevmode
608       \strut\ccUseProperty{caption-#1}\strut%
609     }%
610     \expandafter\ifdim\dimexpr\ht\z@+\dp\z@>\relax>\@tempdima \@tempdima\dimexpr\ht\z@+\dp\z@
611     \relax\fi
612   }%
613   \expandafter\edef\csname ccfcapt@row@height@#1\endcsname{\the\@tempdima}%
614 }

```

`\ccf@calc@sameheight` calculates the ratio between each sub-float's height and the height of the largest sub-float

```

614 \def\ccf@calc@sameheight{%
615   \@tempdima=\z@\relax
616   \@tempcnta=\z@\relax
617   \ccf@calc@width=\ccf@total@width\relax
618   \advance\ccf@calc@width-\ccf@margin@1\relax
619   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
620     \edef\@tempa{CalcRatio{\ccf@sub@maxheight}{\csname cc@\cc@cur@cont @height-\the\@tempcnta\endcsname}}%
621     \ifnum\the\@tempcnta>\@ne\relax

```

```

622 \advance\ccf@calc@width-\ccf@sub@sep\relax%
623 \fi
624 \expandafter\@tempdimc\csname ccf@\cc@cur@cont @width-\the\@tempcnta\endcsname\relax
625 \@tempdimb=\@tempa\@tempdimc\relax
626 \expandafter\edef\csname ccf@\cc@cur@cont @adj@width-\the\@tempcnta\endcsname{\the\@tempdimb}%
627 \advance\@tempdima\@tempdimb
628 }%
629 \@tempcnta=\z@\relax
630 \@tempdimb=\z@\relax
631 \@tempdimc=\z@\relax
632 \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
633 \edef\@tempa{\CalcRatio{\csname ccf@\cc@cur@cont @adj@width-\the\@tempcnta\endcsname}{\
634 \expandafter\edef\csname ccf@\cc@cur@cont @res@width-\the\@tempcnta\endcsname{\dimexpr\@tempa
635 \ccf@calc@width\relax}%
636 \@tempdimc\dimexpr\csname ccf@\cc@cur@cont @height-\the\@tempcnta\endcsname\relax
637 \@tempdimc\dimexpr\@tempa\@tempdimc\relax
638 \ifdim\@tempa\@tempdimb<\@tempdimc\@tempdimb\@tempdimc\relax\fi
639 }%
640 \expandafter\edef\csname ccf@\cc@cur@cont @res@height\endcsname{\the\@tempdimb}%
641 }

```

## 7 Handlers for different float types

### 7.1 Handlers for generic floats

`\ccfGenericRender` is the Component that contains the contents of a generic float.

```

641 \def\ccfGenericRender{\ccUseComp{Content}}

```

`\ccfGenericHandler` is the generic content handler of a float

```

642 \def\ccfGenericHandler{\ccfMakeComp{Content}}

```

`\ccfSubGenericHandler` is the generic handler of a sub-float.

```

643 \def\ccfSubGenericHandler{}

```

### 7.2 Handlers for figures

`\ccfFigureHandler` tells the float module the name, main namespace, and main content Container of `tpFigure` type floats.

```

644 \def\ccfFigureHandler{\ccfMakeComp{Fig}}

```

`\ccf@create@natural` is the actual handler for sub-figures.

```

645 \def\ccf@create@natural{\ccUseComp{Fig}}

```

`\ccfSubFigureHandler` is the User-level macro that defines the handler for sub-figures. It also contains code for the `nofigs` package option.

```

646 \def\ccfSubFigureHandler{%
647   \ifx\ccf@no@figs\relax
648     \setbox\ccf@sub@box\hbox{\rule{0pt}{1pt}\rule{1pt}{0pt}}%
649   \else
650     \setbox\ccf@sub@box\hbox{\ccGobble\ccf@create@natural}%
651   \fi}

```

`\ccfFigureRender` tells the module how `tpFigures` are to be rendered.

```

652 \def\ccfFigureRender{%
653   \bgroup
654   \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
655   {\hsize\dimexpr\textwidth-\ccf@margin@r-\ccf@margin@l\relax}%
656   {}%
657   \let\includegraphics\ccf@includesubgraphics
658   \hskip\ccf@margin@l
659   \ccWhenComp{AltText}{\ccaAddAltText{\ccUseComp{AltText}}}{%
660     \strut\ccUseComp{Fig}\strut
661   \egroup}

```

`\ccfSubFigureRender` tells the module how sub-floats of `tpFigure` type floats are to be rendered.

```

662 \def\ccfSubFigureRender{%
663   \hskip\ccf@margin@l
664   \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
665     \ccfRenderSubFloats{\the\@tempcnta}{Fig}%
666   }}

```

`\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.

```

667 \def\ccf@includesubgraphics{\cc@opt@empty\@ccf@includesubgraphics}%
668 \def\@ccf@includesubgraphics[#1]#2{%
669   \ifx\ccf@current@class\relax
670     \def\@igopts{max width=\hsize,max height=\vsize}%
671   \else
672     \def\@igopts{width=\hsize}%
673   \fi
674   \if!#1!\else
675     \def\@igopts{#1,width=\hsize}%
676   \fi
677   \gdef\ccf@fig@path{#2}%
678   \if@cc@is@final\ccaAddPlacement{Block}\fi%
679   \expandafter\ccf@ltx@includegraphics\expandafter[\@igopts]{#2}%
680 }

```

### 7.3 Handlers for tables

`\ccf@reserve@tabular` is a shell macro that stores the default macro definitions for various tabular mechanisms (currently, only plain `tabular`, `tabulary`, `tabularx`, and `htmltabs` are supported as content Component of `\ccPrefix Table`)

```

681 \def\ccf@reserve@tabular{%
682   \ccf@reserve@tab{}%
683   \ccf@reserve@tab{x}%
684   \ccf@reserve@tab{y}%

```

```

685 \ccf@reserve@htmltab%
686 }

```

`\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.

```

687 \def\ccf@reserve@tab#1{%
688 \expandafter\expandafter\expandafter\let\expandafter\csname orig@tabular#1\expandafter\
  endsname\csname tabular#1\endsname
689 \expandafter\expandafter\expandafter\let\expandafter\csname orig@endtabular#1\expandafter\
  endsname\csname endtabular#1\endsname
690 \expandafter\def\csname tabular#1\endsname{%
691 \global\setbox\ccf@floatbox
692 \vbox\bgroup
693 \if!#1!\else
694 \let\tabular\orig@tabular
695 \let\endtabular\orig@endtabular
696 \fi
697 \csname orig@tabular#1\endsname}%
698 \expandafter\def\csname endtabular#1\endsname{\csname orig@endtabular#1\endsname\egroup}%
699 }

```

`\ccf@reserve@htmltab` special handler for tables using the `htmltabs` package:

```

700 \AtBeginDocument{%
701 \ifpackageloaded{htmltabs}{%
702 \def\ccf@reserve@htmltab{%
703 \let\ccf@add@style\@empty
704 \ifx\ccf@floatpos\@empty
705 \expandafter\ifx\csname \ccPrefix Float\the\ccf@int@cnt Captop\endsname\relax\else
706 \htInitSkip\csname \ccPrefix Float\the\ccf@int@cnt Captop\endsname
707 \advance\htInitSkip\ccf@sep@top%
708 \fi
709 \expandafter\ifx\csname \ccPrefix Float\the\ccf@int@cnt Capbottom\endsname\relax\else
710 \htAddToBottom\csname \ccPrefix Float\the\ccf@int@cnt Capbottom\endsname
711 \advance\htAddToBottom\ccf@sep@bottom%
712 \fi
713 \else
714 \def\ccf@add@style{;break-table:false;}%
715 \fi
716 \edef\cc@tempa{margin-left:\ccf@margin@1\ccf@add@style}%
717 \expandafter\htAddStyle\expandafter{\cc@tempa}%
718 \global\setbox\htTableBox\box\voidb@x
719 \let\htOutputTable\relax
720 }}{\let\ccf@reserve@htmltab\relax}%
721 }

```

`\ccfTableHandler` defines the content handler for `\ccPrefix Table`.

```

722 \def\ccfTableHandler{%
723 \ccfMakeComp{Content}%
724 \ccf@reserve@tabular
725 }

```

`\ccfGetTableContent` returns the `\ccf@floatbox` if it is not un-itialized or void.

```

726 \def\ccfGetTableContent{%
727 \ifx\htTableBox\@undefined\else

```

```

728 \ifvoid\htTableBox\else
729 \let\ccf@floatbox\htTableBox%
730 \fi\fi}

```

`\ccfSubTableHandler` is the handler for sub-tables. So far, `coco-floats.sty` does not support tables to be sub-floats, so we just generate an Error message.

```

731 \def\ccfSubTableHandler{%
732 \PackageError{coco-floats.sty}{ccSubFloat does not support sub-tables (yet)!}{You cannot yet
    use a tables within the 'ccSubFloat'!}%
733 }

```

`\ccfTableRender` defines the Renderer for `\ccPrefix Table` content Components

```

734 \def\ccfTableRender{%
735 \ccfGetTableContent
736 \ccComponent{Content}{\unvbox\ccf@floatbox}%
737 \ccUseComp{Content}%
738 \par\ifccf@break@capt\else\nopagebreak\fi
739 \vskip\dp\strutbox
740 }

```

`\ccfSubTableRender` Is the Renderer for table sub-floats (which we don't allow yet, so this definition is un-used at the moment)

```

741 \def\ccfSubTableRender{%
742 \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
743 \ccfGetTableContent
744 \cc@is@finalfalse
745 \ccComponent{Content}{\unvbox\ccf@floatbox}%
746 \cc@is@finaltrue
747 \ccfRenderSubFloats{\the\@tempcnta}{Content}%
748 }}

```

## 7.4 Helpers

`\ccFloatBarrier` can be used to force all pending floats to be printed at the next shipout.

```

749 \def\ccFloatBarrier{\AtBeginShipoutNext{\clearpage}}

```

## 8 Default Settings

```

750 \ccAddToType{Properties}{float}{%
751 \ccSetProperty{auto-number-prefix}{\csname\ccfCapType name\endcsname}%% Prefix for auto-generated
    Number components
752 \ccSetProperty{auto-number-prefix-sep}{~}%% Prefix for auto-generated Number components
753 \ccSetProperty{intext-skip-top}{\intextsep}%% non-float sep top
754 \ccSetProperty{intext-skip-bottom}{\intextsep}%% non-float sep bottom
755 \ccSetProperty{float-skip-top}{\z@}%% float sep top
756 \ccSetProperty{float-skip-bottom}{\z@}%% float sep bottom
757 \ccSetProperty{sub-float-sep}{\ccf@sub@sep}%% space between sub-floats
758 \ccSetProperty{margin-inner}{\z@}%% left margin on odd pages/right margin on even pages
759 \ccSetProperty{margin-outer}{\z@}%% right margin on odd pages/left margin on even pages

```

```

760 \ccSetProperty{margin-left}{\z@}% left margin
761 \ccSetProperty{margin-right}{\z@}% right margin
762 \ccSetProperty{before-float}{\parindent\z@}% executed before content is evaluated
763 \ccSetProperty{float-handler}{\ccfGenericHandler}% Alias for the caption type specific content
    handler
764 \ccSetProperty{subfloat-handler}{\ccfSubGenericHandler}% Alias for the caption type specific
    content handler
765 \ccSetProperty{float-render}{\ccfGenericRender}% Alias for the caption type specific content printer
766 \ccSetProperty{subfloat-render}{\ccfGenericRender}% Alias for the caption type specific content
    printer for sub-floats
767 \ccSetProperty{subfloat-same-height}{}% if true, the subfloat must/can be adjusted to the same
    heights
768 %% captions
769 \ccSetProperty{caption-face}{}% style applied to top and bottom captions
770 \ccSetProperty{caption-face-top}{}% style applied to top captions
771 \ccSetProperty{caption-face-bottom}{}% style applied to bottom captions
772 \ccSetProperty{source-face}{}% Format of source, additional to caption-format
773 \ccSetProperty{legend-face}{}% Format of legend, additional to caption-format
774 \ccSetProperty{caption-sep-top}{\z@}% vertical space between top caption and content
775 \ccSetProperty{caption-sep-bottom}{\z@}% vertical space between content and bottom caption
776 \ccSetProperty{caption-top}{%
777   \ccIfComp{Number}{\ccUseProperty{number-face}\ccUseComp{Number}\ccUseProperty{number-sep
    }}}%
778   \ccUseComp{Caption}%
779 }%
780 \ccSetProperty{caption-bottom}{%
781   \ccIfComp{Legend}{\ccUseProperty{legend-face}\ccUseComp{Legend}}}%
782   \ccIfComp{Source}{%
783     \ccIfComp{Legend}{\par\nopagebreak}}%
784     {\ccUseProperty{source-face}%
785       \ccUseComp{Source}}}%
786 \ccPropertyLet{subcaption-face}{caption-face}% style applied to top and bottom captions
787 \ccSetProperty{subcaption-face-top}{\ccUseProperty{caption-face-top}}% style applied to top
    captions
788 \ccSetProperty{subcaption-face-bottom}{\ccUseProperty{caption-face-bottom}}% style applied to
    bottom captions
789 \ccSetProperty{subcaption-add-sep-top}{\z@}% additional vertical space between top caption and top
    sub-caption
790 \ccSetProperty{subcaption-add-sep-bottom}{\z@}% additional vertical space between bottom sub-
    caption and bottom caption
791 \ccSetProperty{subcaption-sep-top}{\ccUseProperty{caption-sep-top}}% vertical space between top
    sub-caption and content
792 \ccSetProperty{subcaption-sep-bottom}{\ccUseProperty{caption-sep-bottom}}% vertical space
    between content and bottom sub-caption
793 \ccSetProperty{subcaption-top}{\ccUseProperty{caption-top}}% in case, sub-float captions diverge
    from main caption
794 \ccSetProperty{subcaption-bottom}{\ccUseProperty{caption-bottom}}% in case, sub-float captions
    diverge from main caption
795 \ccSetProperty{subcaption-valign-top}{top}% vertical alignment of neighboring top-placed sub-
    captions
796 \ccSetProperty{subcaption-valign-bottom}{top}% vertical alignment of neighboring bottom-placed sub-
    captions
797 %% Numbers
798 \ccSetProperty{numbering}{auto}% automatic numbering for missing Number component
799 \ccSetProperty{sub-numbering}{}% automatic numbering for missing Number component in real(!) sub-
    floats
800 \ccSetProperty{number-sep}{\enskip}% Separator between label and caption
801 \ccSetProperty{number-face}{\bfseries}% Format of number, additional to caption-format
802 \ccSetProperty{sub-number-sep}{\,}% when sub-captions, this is placed between the float counter and
    the sub-float counter
803 \ccSetProperty{sub-number-style}{alph}% counting style of subcaption counters

```



```

804 \ccSetProperty{sub-number-face}{}% format of subcaption counters
805 \ccSetProperty{sub-number-before}{{}}% stuff that is put immediately before the sub counter
806 \ccSetProperty{sub-number-after}{}% stuff that is put immediately after the sub counter
807 \ccSetProperty{sub-number-format}{}% Format of the sub number
808 \ccUseProperty{float-number}%
809 \ccUseProperty{sub-number-sep}%
810 \ccUseProperty{sub-number}%
811 %% Labels
812 \ccSetProperty{label-pos}{top}% The position of the caption, before which the \ref{} target should be
    placed
813 \ccSetProperty{sublabel-pos}{top}% The position of the subcaption, before which the \ref{} target
    should be placed
814 %% List-of entries
815 \ccSetProperty{list-of-page-sep}{\dotfill}%
816 \ccPropertyLet{list-of-number-face}{list-of-caption-face}%
817 \ccSetProperty{list-of-number-sep}{\enskip}%
818 \ccSetProperty{list-of-number-align}{left}%
819 \ccSetProperty{list-of-number-format}{}%
820 \bgroup
821 \ccUseProperty{list-of-number-face}%
822 \ccUseComp{ListofNumber}%
823 \ccUseProperty{list-of-number-sep}%
824 \egroup}%
825 \ccSetProperty{list-of-parfillskip}{-\rightskip}%
826 \ccSetProperty{list-of-margin-right}{\@pnumwidth \@plus 1fil}%
827 \ccSetProperty{list-of-margin-left}{auto}%
828 \ccSetProperty{list-of-indent}{auto}% list-of-float appearance
829 \ccSetProperty{list-of-block}{}%
830 \ccUseProperty{list-of-caption-face}%
831 \ccIfComp{ListofNumber}
832 {\ccUseComp{list-of-hang-number}}
833 {\leftskip0pt}%
834 \ccUseComp{ListofCaption}%
835 \ccUseProperty{list-of-page-sep}\ccUseComp{ListofPage}%
836 }% list-of-float appearance
837 \ccSetProperty{list-of-before-entry}{}%
838 \ccGobble
839 \leftskip\ccUseProperty{list-of-margin-left}\relax%
840 \rightskip \ccUseProperty{list-of-margin-right}\relax%
841 \parfillskip \ccUseProperty{list-of-parfillskip}\relax
842 \parindent\z@
843 \@afterindenttrue
844 \interlinepenalty\@M
845 \leavevmode
846 \null\nobreak
847 }% list-of-float appearance
848 \ccSetProperty{list-of-after-entry}{\par}% list-of-float appearance
849 }

```

**Container Figure** defines the defaults for the `\ccPrefix Figure` Container.

```

850 \ccDeclareFloat{Figure}[Figure]{figure}{lof}{}%
851 \ccSetProperty{subfloat-same-height}{true}% if true, the subfloat must/can be adjusted to the same
    heights
852 \ccSetProperty{float-handler}{\ccfFigureHandler}%
853 \ccSetProperty{subfloat-handler}{\ccfSubFigureHandler}%
854 \ccSetProperty{float-render}{\ccfFigureRender}%
855 \ccSetProperty{subfloat-render}{\ccfSubFigureRender}%
856 }

```



**Container Table** defines the default Properties of the `\ccPrefix Table` Container.

```
857 \ccDeclareFloat{Table}[Table]{table}{lot}{%  
858   \ccSetProperty{subcaption-valign-top}{bottom}%  
859   \ccSetProperty{float-handler}{\ccfTableHandler}%  
860   \ccSetProperty{subfloat-handler}{\ccfSubTableHandler}%  
861   \ccSetProperty{float-render}{\ccfTableRender}%  
862   \ccSetProperty{subfloat-render}{\ccfSubTableRender}%  
863 }
```

```
864 %</floats>
```



## Modul 11

# coco-frame.dtx

This file provides facilities to visualise crop marks and the print area.

```

24 %<* frame>

25 %%
26 %% module for CoCoTeX for crop marks and print area frames.
27 %%
28 %% Maintainer: p.schulz@le-tex.de
29 %%
30 %% lualatex - texlive > 2019
31 %%
32 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
33 \ProvidesPackage{coco-frame}
34 [2024/03/23 0.4.1 coco-frame]\relax

```

## 1 Top-Level Interface

```

35 \let\cc@frame@mode n
36 \define@choicekey{coco-frame.sty}{frame}[\cc@frame@mode\nr]{none,crop,frame}{%
37 \ifcase\nr\relax% none
38 \let\cc@frame@mode n
39 \or% crop
40 \let\cc@frame@mode p
41 \else% frame
42 \let\cc@frame@mode w
43 \fi
44 }%
45 \ProcessOptionsX\relax

```

## 2 Cropmark printer

```

46 \ifx\cc@frame@mode p\relax
47 \ifx\bleed\undefined \newdimen\bleed \bleed4mm\relax\fi
48 \ifx\cc@frame@offset\undefined \newdimen\cc@frame@offset \cc@frame@offset4em\relax\fi
49 \voffset\dimexpr\cc@frame@offset-1in\relax
50 \hoffset\dimexpr\cc@frame@offset-1in\relax
51 \edef\l@offset{\strip@pt\dimexpr\cc@frame@offset*7200/7227\relax}
52 \edef\r@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperwidth)*7200/7227\relax}
53 \edef\u@offset{\strip@pt\dimexpr(\cc@frame@offset)*7200/7227\relax}
54 \edef\o@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperheight)*7200/7227\relax}
55 \edef\b@l@offset{\strip@pt\dimexpr(\cc@frame@offset-\bleed)*7200/7227\relax}

```

```

56 \edef\b@r@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperwidth+\bleed)*7200/7227\relax}
57 \edef\b@u@offset{\strip@pt\dimexpr(\cc@frame@offset-\bleed)*7200/7227\relax}
58 \edef\b@o@offset{\strip@pt\dimexpr(\cc@frame@offset+\paperheight+\bleed)*7200/7227\relax}
59 \edef@tempa{%
60   /TrimBox [\l@offset\space\u@offset\space\r@offset\space\o@offset]
61   /BleedBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
62   %/CropBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
63   %/MediaBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
64 }
65 \expandafter\pdfpageattr\expandafter{\@tempa}
66 \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:

```

67 \@ifundefined{pdfpagewidth}{%
68   \RequirePackage{luatex85}
69   \pdfpagewidth\paperwidth
70   \pdfpageheight\paperheight
71 }{}

```

Setting PDF boundaries

```

72 \ifx\cc@frame@mode n\relax\else
73   \ifx\cc@frame@mode p\relax
74     \edef\stockwidth{\the\dimexpr\paperwidth+\cc@frame@offset+\cc@frame@offset\relax}
75     \edef\stockheight{\the\dimexpr\paperheight+\cc@frame@offset+\cc@frame@offset\relax}
76   \fi

```

Cropmarks and page area frames both are painted via the `crop` package.

```

77 \RequirePackage{crop}
78 \renewcommand*\CROP@marks{%
79   \CROP@setmarkcolor
80   \CROP@user@b
81   \vskip1in\hskip1in\relax
82   \CROP@ulc\null\hfill\CROP@@@info\CROP@upedge\hfill\null\CROP@urc\hskip-1in\null
83   \vfill
84   \CROP@ledge\hfill\CROP@redge
85   \vfill
86   \hskip1in\relax
87   \CROP@llc\null\hfill\CROP@loedge\hfill\null\CROP@lrc\hskip-1in\null
88   \vskip-1in}%
89 \ifx\cc@frame@mode p\relax
90   \def\camcross{%
91     \smash{\rlap{%
92       \kern-0.15\p@
93       \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
94       \kern-0.15\p@
95       \kern-1.7mm\relax
96       \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
97       \kern-0.3\p@
98       \raise1.7mm\rlap{\vrule\@width3.4mm\@height\z@\@depth0.3\p@}%
99       \lower1.7mm\rlap{\vrule\@width3.4mm\@height0.3\p@\@depth\z@}%
100      \hbox{\vrule\@width3.4mm\@height0.15\p@\@depth0.15\p@}%
101      \kern-0.3\p@
102      \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax}}}
103   \def\cammcrossleft{%
104     \llap{\camcross\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\kern\bleed}}
105   \def\cammcrossright{%

```

```

106 \rlap{\kern\bleed\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\@
    camcross}}
107 \def\cammcrossup{%
108 \rlap{\smash{\raise\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
109 \kern-0.15\p@\vrule\@width0.3\p@\@height\dimexpr\cc@frame@@offset-2mm\relax\@depth-\@
    bleed}}}
110 \def\cammcrossdown{%
111 \rlap{\smash{\lower\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
112 \kern-0.15\p@\vrule\@width0.3\p@\@height-\@bleed\@depth\dimexpr\cc@frame@@offset-2mm\@
    relax}}}
113 \def\CROP@ulc{\cammcrossup\cammcrossleft}
114 \def\CROP@urc{\cammcrossup\cammcrossright}
115 \def\CROP@llc{\cammcrossdown\cammcrossleft}
116 \def\CROP@lrc{\cammcrossdown\cammcrossright}
117 \renewcommand*\CROP@info{%
118 \global\advance\CROP@index\@ne
119 \def\x{\discretionary{}{}{\hbox{\kern.5em--\kern.5em}}}%
120 \ifx\CROP@pagecolor\@empty
121 \else
122 \advance\dimen@\CROP@overlap
123 \fi
124 \hb@xt@\z@{%
125 \hss
126 \lower1em\vbox to\z@{\vss
127 \centering
128 \hsize\dimexpr\paperwidth-20\p@\relax
129 \normalfont
130 \large
131 \vskip5mm\relax
132 \addvspace{\bleed}}%
133 \hss}%
134 }%
135 \crop[cam]

```

the code for the page area frame

```

136 \else% w
137 \@tempdima\dimexpr\textheight\relax
138 \divide\@tempdima by\baselineskip
139 \multiply\@tempdima by65536\relax
140 \edef\cnt@baselines{\strip@pt\@tempdima}%
141 \def\cc@frame@lines{%
142 \@tempcnta\z@
143 \loop\advance\@tempcnta\@ne
144 \hsize1em\relax
145 \ifodd\count\z@
146 \vrule\@width1em\@height0.2\p@\@depth0.02\p@
147 \llap{\smash{\the\@tempcnta\,}}%
148 \fi%
149 \rlap{%
150 \ifodd\count\z@\else\fi
151 \vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
152 \if@twocolumn
153 \kern\columnsep\vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
154 \fi
155 \ifodd\count\z@\else
156 \vrule\@width1em\@height0.00005\p@\@depth0\p@%
157 \llap{\smash{\the\@tempcnta\,}}%
158 \fi
159 }%
160 \break

```

```

161 \ifnum\@tempcnta<\cnt@baselines
162 \repeat}
163 \def\cc@frame@margin{%
164 \vrule height\textheight%
165 \hskip-\marginparwidth\relax
166 \vbox to\textheight{\hsize\marginparwidth\relax
167 \rlap{\vbox to\z@{\hrule width\marginparwidth}}}%
168 \null\vss
169 \rlap{\vbox to\z@{\hrule width\marginparwidth}}}%
170 }%
171 \vrule height\textheight%
172 }
173 \renewcommand*\CROP@@frame{%
174 \vskip0in%
175 \color[cmymk]{0.4,0,0}%
176 \ifodd\count\z@let\@themargin\oddsidemargin\elselet\@themargin\evensidemargin\fi
177 \advance\@themarginlin
178 \moveright\@themargin
179 \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@
180 \vskip\topmargin\vbox to\z@{\vss\hrule width\textwidth}%
181 \vskip\headheight\vbox to\z@{\vss\hrule width\textwidth}%
182 \vskip\headsep\vbox to\z@{\vss\hrule width\textwidth}%
183 \hbox to\textwidth{%
184 \ifodd\count\z@
185 \rlap{\hskip\dimexpr\textwidth+\marginparsep+\marginparwidth\relax\cc@frame@margin}%
186 \else
187 \rlap{\hskip-\marginparsep\relax\cc@frame@margin}%
188 \fi
189 \llap{\vbox to\textheight{\tiny\let\@tempa\fontsize\normalsize\let\fontsize\@tempa\
190 selectfont
191 \vskip\topskip\cc@frame@lines\null\vss}}}%
192 \llap{\vrule height\textheight}%
193 \if@twocolumn
194 \hskip\columnwidth\rlap{\vrule height\textheight}%
195 \hskip\columnsep\rlap{\vrule height\textheight}%
196 \fi
197 \hfil\vrule height\textheight
198 }%
199 \vbox to\z@{\vss\hrule width\textwidth}%
200 \vskip\footskip\vbox to\z@{\vss\hrule width\textwidth}%
201 \vss}%
202 \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@%
203 \vskip-0in\rlap{\hskiplin%
204 \vbox to\z@{\vbox to\z@{\vss\hrule width\paperwidth}%
205 \hbox to\paperwidth{\llap{\vrule height\paperheight}\hfil%
206 \vrule height\paperheight}%
207 \vbox to\z@{\vss\hrule width\paperwidth}%
208 \vss}}\vss}}
209 \crop[frame,noinfo]%
210 \fi

```

```

211 %</frame>

```

## Modul 12

# coco-lists.dtx

This module provides handlers for list-like environments like item lists, enumerations, glossaries and descriptions.

```
24 %<*lists>
```

**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” L<sup>A</sup>T<sub>E</sub>X lists. Configuration of lists, however, follows the CoCoTeX playbook.

## 1 Preamble

```
25 \NeedsTeXFormat{LaTeX2e}[2018/12/01]
26 \ProvidesPackage{coco-lists}
27   [2024/03/23 0.4.1 CoCoTeX lists module]
28 \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 L<sup>A</sup>T<sub>E</sub>X’s `enumerate`, `itemize`, and `description` environments.

```
29 \newif\if@ccl@replace \@ccl@replacefalse
30 \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`.

```
31 \def\ccl@ih@common{common}
32 \def\ccl@ih@conseq{conseq}%
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 }
```

The nesting option sets whether the nesting level of a list should be counted list-specific (value `local`), or globally (value `global`, default).

```

39 \def\ccl@str@local{local}%
40 \def\ccl@str@global{global}%
41 \let\ccl@nesting\ccl@str@global
42 \define@choicekey{coco-lists.sty}{nesting}[\@ccl@nesting\nr]{local,global}{%
43   \ifcase\nr\relax% local
44     \global\let\ccl@nesting\ccl@str@local
45   \fi
46 }
47 \ProcessOptionsX

```

## 2 The List Container

The `List` Container is the most abstract Container for lists.

```

48 \ccDeclareContainer{List}{%
49   \ccDeclareType{Properties}{%
50     %% list formatting
51     \ccSetProperty{before-list}{% at the very beginning of each (nested) list
52       \if@noskipsec \leavevmode \fi
53     \ifvmode\else
54       \unskip \par
55     \fi
56     \ccaStructStart{L}% Start Tag for the (nested) list
57   }%
58   \ccSetProperty{after-list}{% after each (nested) list
59     \ccUseProperty{after-item}%
60     \ccaStructEnd{L}% end tag for the (nested) list
61   }%
62   %% list margins
63   \ccSetProperty{margin-top}{\z@}% vertical space after the list.
64   \ccSetProperty{margin-bottom}{\z@}% vertical space before the list.
65   \ccSetProperty{margin-left}{\csname leftmargin\@roman\cclCurDepth\endcsname-\ccUseProperty{
66     label-sep}+\ccUseProperty{prev-margin-left}}% horizontal space to the left of each item, from
67     left boundary of the page area (auto=width of widest label + prev-margin-left, top-level-list-wise)
68   \ccSetProperty{max-label-width}{.33\textwidth}% maximum margin reserved for list labels
69   \ccSetProperty{margin-right}{\z@}% horizontal space to the right of each list item
70   %% between list items
71   \ccSetProperty{item-sep}{\z@}% vertical space between two adjacent list items (real: this value +
72     par-skip)
73   \ccSetProperty{after-indent}{false}% whether the paragraph after the list should have an indent (
74     true) or not (false)
75   \ccSetProperty{at-begin-item-body}{\ccaVstructStart{LBody}}% right at the beginning of a new item
76     body
77   \ccSetProperty{at-end-item-body}{\ccaVstructEnd{LBody}}% at the very end of an item body, but
78     before \par
79   \ccSetProperty{after-item}{% material after each item
80     \ccUseProperty{at-end-item-body}%
81     \ccaVstructEnd{LI}% Close list item tags
82     \par}%
83   \ccSetProperty{before-item}{%
84     \ifcclFirst
85       \global\cclFirstfalse
86     \else
87       \ccUseProperty{after-item}%
88     \vskip\ccUseProperty{item-sep}%

```



```

83 \fi
84 \ccUseProperty{par-indent}\relax%
85 \parskip\ccUseProperty{par-skip}\relax%
86 \parfillskip\ccUseProperty{par-fill-skip}\relax%
87 \noindent
88 \leavevmode
89 \ccaVstructStart{LI}% Start tag for a list item
90 }%
91 \ccSetProperty{item-offset}{% Setting the label indent and first-line offset
92 \cclItemIndent\ccUseProperty{indent}%
93 \advance\cclItemIndent\dimexpr-\ccUseProperty{label-sep}\relax
94 \hskip\cclItemIndent\relax%
95 \ifdim\ccUseProperty{indent}>\z@
96 \cclItemIndent\ccUseProperty{indent}%
97 \else
98 \cclItemIndent-\ccUseProperty{indent}%
99 \fi
100 }%
101 %% inside list items
102 \ccSetProperty{par-indent}{\parindent}% indent of the first line of a *new* paragraph inside a list
103 item
104 \ccSetProperty{par-fill-skip}{\@flushglue}% skip at the end of the last line of each paragraph
105 inside a list item
106 \ccSetProperty{par-skip}{\z@}% vertical space between two adjacent paragraphs inside a list item
107 %% label formatting
108 \ccSetProperty{label}{\ccUseComp{Label}}% The Label Component is set via the optional argument of \
109 Item, otherwise it is generated
110 \ccSetProperty{indent}{-\dimexpr\csname leftmargin\@roman\cclCurDepth\endcsname-\
111 ccUseProperty{label-sep}\relax}% indent of each list item's first line (relative to margin-left
112 ) NOTE: auto-global is valid, but it causes *all* lists -- despite the nesting level -- to have the
113 same left margin and indent!
114 \ccSetProperty{label-sep}{.5em}% horizontal skip between each item's label and its content
115 \ccSetProperty{label-face}{}% font of the item's label
116 \ccSetProperty{label-align}{left}% alignment of label within its \hbox
117 \ccSetProperty{label-format}{% format of the label itself
118 \ccUseProperty{label-face}%
119 \ccaVstructStart{Lbl}% Start Tag for the item's label
120 \ccUseProperty{label}%
121 \ccaVstructEnd{Lbl}% End tag for the item's label
122 }%
123 \ccSetProperty{label-box}{% hbox that contains and aligns the label
124 \hbox to \cclItemIndent{%
125 \ccIfPropVal{label-align}{left}{\hss}%
126 \ccUseProperty{label-format}%
127 \ccIfPropVal{label-align}{right}{\hss}%
128 }%
129 }%
130 \ccDeclareType{Components}{%
131 \ccDeclareComponent{Label}%
132 }%
133 \ccDeclareEnv{cc@list}{endcc@list}%
134 }

```

### 3 Declaring List Types

List Types are the next layer of abstraction for lists. This layer distinguishes numbered from unnumbered 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.

```
135 \long\def\ccDeclareListType#1#2#3{%
```

`\DeclareAttributeHandler` declares a new handler for a list's attributes. ##1 is the definition body.

```
136 \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.

```
137 \def\DeclareLabelHandler##1{\csdef{ccl@make@label@#1}{##1}}%
```

```
138 \ccDeclareContainer{#1List}{%
139   \ccInherit{Components,Properties}{List}%
140   \ccDeclareType{Properties}{%
141     \ccSetProperty{list-type}{#1}%
142     #3%
143   }%
144   \ccDeclareEnv[#1-list]{cc@list}{endcc@list}%
145 }%
146 #2%
147 }
```

### 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.

```
148 \def\ccDeclareList#1#2#3{%
149   \csxdef{cc@cur@depth@#1}{\z@}%
150   \ccDeclareContainer{#1}{%
151     \ccInherit{Properties,Components}{#2List}%
152     \ccDeclareType{Properties}{#3}%
153     \ccDeclareEnv[#1]{\cc@list}{\endcc@list}%
154   }%
155   \ccDeclareNested{#1}{\z@}{#3}%
156 }
```

`\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.

```

157 \def\ccDeclareNested#1#2#3{%
158   \@tempcnta=#2\relax
159   \ifx\@tempcnta<\@ne\relax
160     \ccPackageError{lists}{Nesting}{Invalid nesting level!}{You cannot declare nesting levels
161       less than 1!}%
162   \fi
163   \advance\@tempcnta\@ne\relax
164   \ccDeclareContainer{#1-\the\@tempcnta}{%
165     \ifcsdef{cc@container@#1}
166       {\ccInherit{Properties,Components}{#1}}
167       {\ccPackageError{lists}{Inheritance}
168         {List ‘#1’ undefined!}
169         {You need to define the list ‘#1’ before you can declare nested list overrides!}}%
170     \ccDeclareType{Properties}{#3}%
171   }%

```

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.

```

172 \newcount\ccl@depth

```

`\ccl@item@cnt` is the internal counter for the items within a (nested) list level.

```

173 \newcount\ccl@item@cnt

```

`\ifcclFirst` is true as long as the first item of a list is processed.

```

174 \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.

```

175 \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.

```

176 \def\ccl@advance@depth@global#1{%
177   \edef\cclPrevDepth{\the\ccl@depth}%
178   \global\advance\ccl@depth#1\relax
179   \edef\cclCurDepth{\the\ccl@depth}%
180 }

```

`\ccl@advance@depth@local` is called when the nesting level should be counted for each list type individually.

```

181 \def\ccl@advance@depth@local#1{%
182   \letcs\cclPrevDepth{cc@cur@depth@\cc@cur@cont}%
183   \expandafter\@tempcnta\csname cc@cur@depth@\cc@cur@cont\endcsname\relax
184   \advance\@tempcnta#1\relax
185   \csxdef{cc@cur@depth@\cc@cur@cont}{\the\@tempcnta}%
186   \edef\cclCurDepth{\csname cc@cur@depth@\cc@cur@cont\endcsname}%
187   \global\advance\ccl@depth#1\relax

```

```

188 }
189 \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.

```

190 \newcount\cclTopID \cclTopID\z@\relax

```

`\cclID` stores a unique “identifier” number for each list, irrespective their nesting levels. 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.

```

191 \newcount\cclID \cclID\z@\relax
192 \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.

```

193 \def\ccl@incr@count{%
194   \csxdef{ccl@prev@cnt@the\ccl@depth}{\the\cclID}%
195   \global\advance\ccl@total@list@cnt\@ne\relax
196   \global\cclID\ccl@total@list@cnt\relax
197   \ifnum\cclCurDepth=\@ne\relax
198     \global\advance\cclTopID\@ne\relax
199   \fi
200 }

```

`\ccl@decr@count` resets the list counter for the next lower nesting level, whenever a nested list is closed.

```

201 \def\ccl@decr@count{%
202   \global\cclID\csname ccl@prev@cnt@the\ccl@depth\endcsname\relax
203 }

```

## 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.

```

204 \def\cc@list{\cc@opt@empty\cc@list}
205 \def\@cc@list[#1]{%
206   \ccl@advance@depth\@ne%
207   \ccl@incr@count%
208   \edef\ccl@cur@cont{\cc@cur@cont-\cclCurDepth}%
209   \global\cclFirsttrue

```

If the nesting goes deeper than the style programmer anticipated:

```

210 \ifcsdef{cc@container@\ccl@cur@cont}{}
211   {\ifx\ccl@inherit\ccl@ih@common
212     \let\ccl@cur@cont\cc@cur@cont%
213   \else
214     \global\csletcs
215     {cc@type@Properties@\cc@cur@cont-\cclCurDepth}

```

```

216 {cc@type@Properties@{cc@cur@cont-\cclPrevDepth}%
217 \fi}%

```

Horizontal margin Properties from the previous nesting level are stored so that the nested lists can use them:

```

218 \ccSetPropertyX{prev-margin-left}{\the\leftskip}%
219 \ccSetPropertyX{prev-margin-right}{\the\rightskip}%
220 \ccEvalType[\ccl@cur@cont]{Properties}%
221 \edef\ccl@list@type{\ccUseProperty{list-type}}%

```

Processing of the optional argument.

```

222 \cclUseAttributeHandler{#1}%

```

The macro that separates the items of the list is defined locally so that we can use Properties:

```

223 \cclCalculateMarginLeft%
224 \cclCalculateVMargin{top}%
225 \cclCalculateVMargin{bottom}%
226 \csdef{\ccPrefix Item}{\cc@opt@empty\ccl@item}%
227 \def\ccl@item[##1]{%
228   \edef\ccl@item@label{##1}%
229   \ifx\ccl@item@label\@empty
230     \cclUseLabelHandler%
231   \else
232     \ccComponent{Label}{##1}%
233   \fi
234   \sbox\z@{\@cc@is@finalfalse\ccUseProperty{label-format}}%
235   \@tempdima=\dimexpr\ccUseProperty{max-label-width}\relax
236   \ifdim\wd\z@<\@tempdima\relax
237     \@tempdima=\the\wd\z@\relax%
238   \fi
239   \bgroup
240     \def\cc@cur@cont{list}%
241     \cc@store@latest{\the\cclTopID-number-\cclCurDepth-maxwd}{\the\@tempdima}%
242     \cc@store@latest{\the\cclTopID-number-maxwd}{\the\@tempdima}%
243   \egroup
244   \ccSetPropertyX{label-width}{\the\@tempdima}%
245   \ccUseProperty{item-format}%
246   \ccUseProperty{at-begin-item-body}\ignorespaces%
247 }%

```

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`.

```

248 \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:

```

249 \ccUseProperty{before-list}%
250 \ccUseProperty{int-margin-top}%
251 \leftskip\dimexpr\ccUseProperty{margin-left}+\ccUseProperty{label-sep}\relax%
252 \rightskip\dimexpr\ccUseProperty{margin-right}\relax%
253 }

```

`\endcc@list` is expanded at the end of each List Container's respective environment. It basically calls the after-list Property one last time and decrements the depth counter(s).

```

254 \def\endcc@list{%
255   \ccUseProperty{after-list}%

```

```

256 \ccl@decr@count%
257 \ccl@advance@depth\m@ne%
258 \ccUseProperty{int-margin-bottom}%
259 \ifnum\cclCurDepth=\z@\relax
260 \ccIfPropVal{after-indent}{false}%
261 \global\@afterindentfalse
262 \global\everypar{%
263 \if@afterindent \else
264 {\setbox\z@\lastbox}%
265 \fi
266 \global\everypar{}}}%
267 \fi
268 }

```

`\cclCalculateVMargin` generates a macro that realizes the internal vertical margin of the (nested) list. #1 is the orientation (`top` or `bottom`).

```

269 \def\cclCalculateVMargin#1{%
270 \ifdim\ccUseProperty{margin-#1}=\z@\relax
271 \ccSetProperty{int-margin-#1}{\relax}%
272 \else
273 \ccSetProperty{int-margin-#1}{\addvspace{\ccUseProperty{margin-#1}}}%
274 \fi
275 }

```

`\cclCalculateLeftMargin` generates the value that `\leftskip` is set to.

```

276 \def\cclCalculateMarginLeft{%
277 \ifcsdef{cc-list-\the\cclTopID-number-maxwd}
278 {\ccSetPropertyVal{number-width-max}{\csname cc-list-\the\cclTopID-number-maxwd\endcsname}}
279 {\ccSetPropertyVal{number-width-max}{1sp}}%
280 \ifcsdef{cc-list-\the\cclTopID-number-\cclCurDepth-maxwd}
281 {\ccSetPropertyVal{number-width-level-max}{\csname cc-list-\the\cclTopID-number-\cclCurDepth
282 -maxwd\endcsname}}
283 {\ccSetPropertyVal{number-width-level-max}{1sp}}%
284 \cc@get@indent[\ccl@calc@margin@left]{\the\cclTopID}%
285 }

```

`\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.

```

285 \def\ccl@calc@margin@left#1#2{%
286 \@tempdima=\ccUseProperty{prev-margin-left}\relax%
287 \ccSetPropertyX{margin-left}{\the\dimexpr\@tempdima-\ccUseProperty{indent}\relax}%
288 }

```

## 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.

```

289 \def\cclUseLabelHandler{%
290   \expandafter\ifx\csname ccl@make@label@\ccl@list@type\endcsname\relax
291     \ccPackageError{lists}{type}
292     {List type ‘\ccl@list@type’ does not provide a Label Handler.}
293     {Make sure that the body of \ccl@list@type’s declaration contains a \string\
      DeclareLabelHandler.}
294   \else
295     \csname ccl@make@label@\ccl@list@type\endcsname
296   \fi
297 }

```

`\cclUseAttributeHandler` checks if the list type specific attribute handler exists and applies it to the attribute list #1.

```

298 \def\cclUseAttributeHandler#1{%
299   \ccParseAttributes{\cc@cur@cont-\cclCurDepth}{#1}%
300   \expandafter\ifx\csname ccl@eval@attrs@\ccl@list@type\endcsname\relax
301     \ccPackageError{Lists}{Type}
302     {List type ‘\ccl@list@type’ does not provide an Attribute Handler.}
303     {Make sure that the body of \ccl@list@type’s declaration contains a \string\
      DeclareAttributeHandler.}
304   \else
305     \csname ccl@eval@attrs@\ccUseProperty{list-type}\endcsname
306   \fi
307 }

```

## 5 Default List Types

Vanilla CoCoTeX 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

```

308 \ccDeclareListType{unnumbered}{%

```

`\ccl@make@label@unnumbered` creates the label of an unnumbered list type.

```

309   \DeclareLabelHandler{%
310     \ccComponent{Label}}{\ccUseProperty{default-label}}

```

`\ccl@eval@attrs@itemize` is the handler for attributes of itemize-like list types. Currently, it does nothing.

```

311   \DeclareAttributeHandler{}

```

#### Itemize-Type List Specific Properties

For unnumbered lists there is one new Property, `default-label` which defines a fallback label.

```

312 }{\ccSetProperty{default-label}{-}}

```

### Itemize-Style Default Lists

```

313 \ccDeclareList{Itemize}{unnumbered}{\ccSetProperty{default-label}{\textbullet}}
314 \ccDeclareNested{Itemize}{1}{%
315   \ccSetProperty{label-face}{\normalfont\bfseries}%
316   \ccSetProperty{default-label}{\textendash}}
317 \ccDeclareNested{Itemize}{2}{\ccSetProperty{default-label}{\textasteriskcentered}}
318 \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 numebred lists should advance for each item.

```

319 \newcount\ccl@item@adv

```

```

320 \ccDeclareListType{numbered}{%

```

`\ccl@eval@attrs@numbered` is the handler for attributes specific to the enumerate-like list types.

```

321 \DeclareAttributeHandler{%

```

The attribute `step` indicates by what amount the interal counter should be advanced for each item. Defaults to +1 if none is given.

```

322   \ccIfAttr{\cc@cur@cont-\cclCurDepth}{step}
323     {\ccl@item@adv=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@step\
324       endcsname\relax}%
325     {\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 substract 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 coutner is initialized to 0.

```

326   \ccIfAttr{\cc@cur@cont-\cclCurDepth}{start}
327     {\ccl@item@cnt=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@start\
328       endcsname\relax
329       \advance\ccl@item@cnt-\ccl@item@adv}%
330     {\ccl@item@cnt=\z@\relax}%
331   }

```

`\ccl@make@label@numbered` is the label handler of a numbered list type.

```

330 \DeclareLabelHandler{%
331   \advance\ccl@item@cnt \ccl@item@adv\relax
332   \expandafter\ifx\csname ccl@label@type@\ccUseProperty{enum-type}\endcsname\relax
333     \ccPackageWarning{lists}{type}{Enum type \ccUseProperty{enum-type} is unknown, revert to
334       numeric counters!}
335     \let\ccl@label\ccl@label@type@arabic%
336   \else
337     \letcs\ccl@label{\ccl@label@type@\ccUseProperty{enum-type}}%
338   \fi
339   \ccComponent{Label}{\ccl@label{\ccl@item@cnt}}
340 }%

```

```

340 }{%

```



## Numbered List-Specific Properties

### New Properties

The new Property `enum-type` 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.

```
341 \ccSetProperty{enum-type}{arabic}%
```

### Properties with Deviating Default Values

By default, numeric labels are followed by a period to accommodate L<sup>A</sup>T<sub>E</sub>X customs.

```
342 \ccSetProperty{label}{\ccUseComp{Label}.}}
```

### Available Counting Styles

`\ccl@label@type@arabic`, `\ccl@label@type@roman`, `\ccl@label@type@Roman`, `\ccl@label@type@alph`, `\ccl@label@type@Alph` are wrappers for all the available counter transformers built-in the `\LaTeX` kernel.

```
343 \def\ccl@label@type@arabic{\@arabic}
344 \def\ccl@label@type@roman{\@roman}
345 \def\ccl@label@type@Roman{\@Roman}
346 \def\ccl@label@type@alph{\@alph}
347 \def\ccl@label@type@Alph{\@Alph}
```

### Enumerate-Style Default Lists

```
348 \ccDeclareList{Enumerate}{numbered}{}
349 \ccDeclareNested{Enumerate}{1}{% (
350   \ccSetProperty{label}{\ccUseComp{Label}})%
351   \ccSetProperty{enum-type}{alph}%
352 }
353 \ccDeclareNested{Enumerate}{2}{\ccSetProperty{enum-type}{roman}}
354 \ccDeclareNested{Enumerate}{3}{\ccSetProperty{enum-type}{Alph}}
```

## 5.3 Description Lists

```
355 \ccDeclareListType{text}{%
```

`\ccl@eval@attrs@text` is the handler for the attributes of description-like list types.

```
356 \DeclareAttributeHandler{%
357   \ccIfAttr{\cc@cur@cont-\cclCurDepth}{width}
358   {\ccSetPropertyVal{min-margin-left}{\expandafter\dimexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@width\endcsname\relax}}%
359   {\ccSetProperty{min-margin-left}{2em}}%
360 \ccIfPropVal{label-growth}{down}
361   {\long\def\ccl@vbox##1{\smash{\vtop{##1}}}}
362   {\long\def\ccl@vbox##1{\vbox{##1}}}%
363 }
```

`\ccl@make@label@text` creates the label of a description-like list type.

```

364 \DeclareLabelHandler{%
365   \ccComponent{Label}{}%
366 }

```

## Description-Type Specific Properties

### New Properties

There is a new Property, `label-growth`, which can be set to `up` or `down` and which 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.

### Properties with Deviating Default Values

TO accommodate for the new option, the `label-box` has a conditional that switches between regular `\hbox` labels and the two `\vbox` variants described above..

The Properties `margin-left` and `indent` of text-type lists are by default set to `auto`.

```

367 }{%
368   \ccSetProperty{label-growth}{up}% or 'down'; where a multi-line label should "grow" to.
369   \ccSetProperty{indent}{auto}%
370   \ccSetProperty{margin-left}{auto}%
371   \ccSetProperty{label-box}{%
372     \ifdim\ccUseProperty{label-width}<\ccUseProperty{max-label-width}\relax
373       \hbox to \cclItemIndent{%
374         \ccIfPropVal{label-align}{left}{}{\hss}%
375         \ccUseProperty{label-format}%
376         \ccIfPropVal{label-align}{right}{}{\hss}}%
377     \else
378       \ccl@vbox{\relax%
379         \hsize\dimexpr\cclItemIndent%
380         \leftskip\z@
381         \rightskip\z@
382         \parindent\z@
383         \leavevmode
384         \ccUseProperty{label-format}%
385         \@@par
386       }%
387     \fi
388   }%
389 }

```

## Description-Type Default Lists

As with the standard `\LaTeX description` environment, there are no default definitions for nested Description-type lists.

```

390 \ccDeclareList{Description}{text}{%
391   \ccSetProperty{label-face}{\bfseries}
392 }

```

## 5.4 Replacing L<sup>A</sup>T<sub>E</sub>X's Default Lists

At the User's discretion (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.

```
393 \if@ccl@replace
394   \letcs\itemize{\ccPrefix Itemize}
395   \letcs\enditemize{end\ccPrefix Itemize}
396   \letcs\enumerate{\ccPrefix Enumerate}
397   \letcs\endenumerate{end\ccPrefix Enumerate}
398   \letcs\description{\ccPrefix Description}
399   \letcs\enddescription{end\ccPrefix Description}
400 \fi
```

```
401 %</lists>
```