The cocotex.dtx Package

A modular package suite for automatic, flexible typesetting

Version v0.5.0

(2024/12/13)

Lupino

lupino@le-tex.de

Table of contents

Introduction

1 Basic concepts

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

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

The final realization of a *Container* in the rendered output is done in local style files with so-called *Properties*; short snippets of LATEX code, which tell the LATEX interpreter how the Components in the Instances of Containers are to be read, processed and eventually rendered.

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

1.1 Types, Inheritance and Abstract Containers

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

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

1.2 Complex Components

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

Other Components may occur more than once in the same Container Instance, for example, a "chapter" (which itself may be a Sub Container of a more abstract Parent Container "Heading") may have more than one "Author". Such Components are called *Group Components*. They are usually realized as LATEX environments within a Container's environment and contain themselves other Components. Those "second-level" Components are called *Counted Components*, as they are "enumerated" across all Group Component instances within the same Container Instance. For each Group Component, there is a *Collection Component*, in which all instances of a Group Component are collected during processing. How this collection is put together is controlled by a special *Collection Property*.

1.3 Relation to LaTeX Templates

Newer version of LaTeX adopt a quite similar design principle with the introduction of *Templates* into the LaTeX Kernel in mid 2024[1]

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

As an example, an *Object Type* may be "sectioning" that provides the Arguments `title`, `short title` and `number`. A *Template* `heading` derived from that *Object Type* may introduce the Interface key-value pairs `pre-skip`, `font-size` and *after-skip*. A *Template Instance* then might be *subsection*, which defines the `pre-skip` to `2\baselineskip`, the `after-skip` to `1\baselineskip` and the `font-size` to `{10}{12}`. The *Template Instance* is then being called in the definition of a user level macro like `\subsection[<label>][<shorttitle>]{<title>}`.

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

In summary, CoCoTeX has no formal distinction between *Object Types*, *Templates* and *Template Instaces*. However, those "layers of abstraction" can be realized by the *Inheritance* mechanism, but there is no hard restrictions about *if* and *when Properties* and/or *Components* are introduced.

2 How to Read This Documentation

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

```
This is the documented source code
```

Code and usage examples are printed in blue boxes:

```
This is a {\LaTeX} example.
```

2.1 Keyword Markings

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

📦 `Containers` are orange and marked with a box symbol 📦,
☰ `Hooks` are green and marked with an insertion icon ☰,
➔ `Components` are blue and marked with an arrow to box symbol ➔,
⚙ `Properties` are purple and marked with a gear symbol ⚙,
🏷 `PDF-Tags` are cyan and marked with a tag symbol 🏷,
🔗 `Attributes` are dark green and marked with a chain symbol 🔗, and
`LaTeX-Macros` are red and have no symbol.

[1] It is however noteworthy that the principal functionality has been available for much longer in form of the `xtemplate` package.

2.2 Data Types of Properties

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

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

2.3 Types of Components

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

One more driver function

```
<*driver>
```

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

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

End driver function

```
</driver>
```

Module 1
cocotex.dtx

```
<*class>
```

This is the main class file for the CoCoTEX Framework.

File Preamble

```
%%
%% Common document class for \textit{xerif} projects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesClass{cocotex}
   [2024/12/13 v0.5.0 cocotex]
```

1 Hard-coded requirements

First,we set the default hook label for all CoCoTEX modules. Since some modules are ‘‘stand-alone’’, we do this in all kernel files, i.e., here, in `coco-kernel` and in `coco-common`.

```
\SetDefaultHookLabel{cc}
```

2 Class Options

2.1 Options passed down to mandatory standard LATEX packages

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

```
\ExplSyntaxOn
\keys_define:nn { cocotex/cls }
{
  main .code:n = { \PassOptionsToPackage{main=#1}{babel} },
```

The next two options are used for Spanish documents:

```
  es-noindentfirst .code:n = { \PassOptionsToPackage{es-noindentfirst}{babel} },
  es-noshorthands .code:n = { \PassOptionsToPackage{es-noshorthands}{babel} }
```

By default, we disable babel shorthands as its character encoding may interfere with some CoCoTEX functionality.

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

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

```
\keys_define:nn { cocotex/cls }
{
  no-hyperindex .code:n = {\global\let\cc@no@hyperindex\relax}
}
```

2.2 The Publication Type

The option `pubtype` (short for ‘‘publication type’’) has four possible values: `mono`, `collection`, `journal`, and `article`. `mono` (also the default when no `pubtype` is given) and `collection` are used to switch between single and multiple contributor documents; `collection` and `journal` to switch between one-time text collections and periodicals, respectively. All three types implicitly load the LaTeX 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 publicaten type `article` is intended for single articles of a journal. It loads the LaTeX standard class `article`.

```
\newif\ifcollection \collectionfalse
\newif\ifarticle \articlefalse
\newif\ifmonograph \monographfalse
\newif\ifjournal \journalfalse
\keys_define:nn { cocotex/cls }
{
  pubtype .choice:,
  pubtype / collection .code:n = { \global\collectiontrue },
  pubtype / article .code:n = { \global\articletrue },
  pubtype / journal .code:n = { \global\journaltrue },
  pubtype / mono .code:n = { \global\monographtrue },
  pubtype .initial:n = mono,
}
```

2.3 User-Level Macro Names and Debugging Options

Next, we capture all options that needs passing down to the various CoCoTeX modules.

The prefix option is used to define prefix for user level macros. If CoCoTeX is used in conjunction with *xerif*, this will be `tp`.

```
\keys_define:nn { cocotex/cls }
{
  prefix .code:n = { \PassOptionsToPackage{prefix=#1}{coco-kernel} },
```

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

```
  debug .code:n = { \PassOptionsToPackage{debug}{coco-kernel} },
  debug-domain .code:n = { \PassOptionsToPackage{debug-domain=#1}{coco-kernel} },
```

The `silent` options attempts to supress most unneccessary messages sent to the shell.

```
  silent .code:n = {\PassOptionsToPackage{silent}{coco-common}},
```

The `nofigs` options disables the `\includegraphics` command and prints a placeholder instead. This is inteded to ensure successful LaTeX runs even thou image files may be missing.

```
  nofigs .code:n = {\PassOptionsToPackage{nofigs}{coco-floats}},
}
```

2.4 Accessibility Features

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

`\if@cc@pdf@two` is a switch that indicates PDF version 2.0 is used (`true`) instead of 1.x (`false`).

```
\newif\if@cc@pdf@two \@cc@pdf@twofalse
```

```
\keys_define:nn { cocotex/cls }
{
```

Options for the aimed PDF standard. Currently supported:
A-1b, A-1a, A-2b, A-2a, A-2u, A-3b, A-3a, A-3u, A-4,
X-3, X-4, X-4p, X-5g, X-5n, X-5pg, X-6, X-6n, X-6p,
UA-1 or UA-2.
Those choices are case-insensitive.

```
  pdf-standard .code:n =
    {
      \exp_args:Nnx
      \keys_set:nn {cocotex/cls} {_pdfstandard=\str_uppercase:n{#1}}
    },
  _pdfstandard .choices:nn =
    {
      A-1B, A-1A, A-2B, A-2A, A-2U, A-3B, A-3A, A-3U, A-4,
      X-3, X-4, X-4P, X-5G, X-5N, X-5PG, X-6, X-6N, X-6P,
      UA-1, UA-2
    }
    {
      \xdef\cc@pdf@standard{ \tl_use:N \l_keys_choice_tl }
      \xdef\cc@pdf@std{\expandafter\@car\cc@pdf@standard\@nil}
    },
  _pdfstandard .initial:n = A-2B,
  _pdfstandard / unknown .code:n =
    { \msg_warning:nnn{pdf}{unknown-standard}{#1} },
```

Options for the aimed PDF version. Currently supported are PDF 1.3 through 1.7 and PDF 2.0. Default is 1.7.

```
  pdf-version .choices:nn =
    { 1.3, 1.4, 1.5, 1.6, 1.7, 2.0 }
    {
      \sys_ensure_backend:
      \exp_args:Ne \pdf_version_gset:n { \tl_use:N \l_keys_choice_tl }
    },
  pdf-version .initial:n = { 1.7 },
```

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

The following two options are there for backard compatibility and are considered legacy code. Both cause PDF/UA, `a11y` UA-1 and PDF v1.7, `a11y20` UA-2 with PDF v2.0.

```
a11y .code:n = { \keys_set:nn { cocotex/cls } { pdf-standard = UA-1, pdf-version=1.7 } },
a11y20 .code:n = { \keys_set:nn { cocotex/cls } { pdf-standard = UA-2, pdf-version=2.0 } },
```

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

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

```
lang-id .code:n = {},
```

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

```
nodetree .code:n = {\PassOptionsToPackage{nodetree}{coco-accessibility}},
```

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

```
show-spaces .code:n = {\PassOptionsToPackage{show-spaces}{coco-accessibility}},
```

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

```
no-spaces .code:n = {\PassOptionsToPackage{no-spaces}{coco-accessibility}},
```

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

```
no-paras .code:n = {\PassOptionsToPackage{no-paras}{coco-accessibility}},
```

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

```
no-compress .code:n =
{
  \ifx\pdfobjcompresslevel\@undefined
    \edef\pdfobjcompresslevel{\pdfvariable objcompresslevel}%
  \fi
  \pdfcompresslevel=0
  \pdfobjcompresslevel=0
  \pdf_uncompress:
},
```

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

```
color-enc .code:n = {\PassOptionsToPackage{color-enc=#1}{coco-common}},
```

2.5 Options for Other CoCoTEX Modules

Options for the Script Module

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

```
usescript .code:n = {\PassOptionsToPackage{usescript={#1}}{coco-script}},
```

Options for the Notes Module

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

```
  endnotes .code:n = {\PassOptionsToPackage{endnotes}{coco-notes}},
```

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

```
  ennotoc .code:n = {\PassOptionsToPackage{ennotoc}{coco-notes}},
```

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

```
  endnoteswithchapters .code:n = {\PassOptionsToPackage{endnoteswithchapters}{coco-notes}},
```

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

```
  resetnotesperchapter .code:n = {\PassOptionsToPackage{resetnotesperchapter}{coco-notes}},
```

Remaining Options

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

```
}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{book}}
```

Process the options.

```
\ProcessOptions
\ProcessKeyOptions[cocotex/cls]
```

After processing the options, we need to do some checks since some options depend on other options.

First we check if the user requests one of the PDF/UA standards which trigger initialization of the coco-accessibility module. Further, we check if pdf version is 2.0 when the user selects PDF/UA-2 and issue a soft error if this is not the case.

```
\msg_new:nnnn {cocotex/dtx} { wrong_pdf_version }
  {PDF/UA-2~requires~PDF~version~2.0~or~later!}
  {You~want~to~create~a~PDF~that~conforms~to~the~\cc@pdf@standard~standard,~which\\
  requires~at~least~`pdf-version=2.0',~but~you~requested~`pdf-version=\pdf_version:'}
\str_if_eq:VnT \cc@pdf@std {U}{ \PassOptionsToPackage{init}{coco-accessibility} }
\str_if_eq:VnT \cc@pdf@standard {UA-2}
  {
    \str_if_eq:eeF { 2.0 } {\pdf_version:}
      {
        \msg_error:nn
          { cocotex/dtx }
          { wrong_pdf_version }
      }
    \global\@cc@pdf@twotrue
  }
\ExplSyntaxOff
```

3 Class Hook

\ccAfterClassHook Almost all user level macros have been renamed when CoCoTEX 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 CoCoTEX itself, but included in xerif's common files[1]). 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.

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

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

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

4 Internal Requirement

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

5 Loading and Adjusting Underlying DocumentClass

All publication types supported by CoCoTEX are based on one of LATEX's default classes article (when pubtype=article) or book (all other pubtypes):

```
\ifarticle
  \LoadClass[10pt,a4paper]{article}
\else
  \LoadClass[10pt,a4paper]{book}
\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.

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

Automatted typesetting needs some room to play

```
\emergencystretch=2em
```

and strong restrictions:

```
\frenchspacing
\clubpenalty10000
\widowpenalty10000
```

[1] see https://github.com/transpect/xerif-latex

Empty Pagestyle

Page style without any headers or footers

```
\def\ps@empty{%
  \let\@oddhead\@empty
  \let\@evenhead\@empty
  \let\@oddfoot\@empty
  \let\@evenfoot\@empty
}
```

Vacancy Pages

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

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

Book Parts

The macros `\frontmatter`, `\mainmatter` and `\backmatter` are re-defined to make front- and backmatter components in book derivates distinguish-able.

Note that we need to (re-)define the conditionals outside, because `\if@mainmatter` is undefined for articles and the coditional inside the definition of `\mainmatter` disturbs the outer `\ifarticle`.

```
\newif\if@frontmatter \@frontmatterfalse
\newif\if@mainmatter \@mainmatterfalse
\newif\if@backmatter \@backmatterfalse
\ifarticle\else
  \renewcommand\frontmatter{%
    \cleardoublepage
    \cchResetNesting
    \global\@mainmatterfalse
    \global\@backmatterfalse
    \global\@frontmattertrue
    \ccaVstructStart{Frontmatter}%
    \pagenumbering{arabic}}
  \renewcommand\mainmatter{%
    \if@frontmatter\ccaVstructEnd{Frontmatter}\fi
    \cleardoublepage
    \cchResetNesting
    \global\@frontmatterfalse
    \global\@backmatterfalse
    \global\@mainmattertrue
    \ccaVstructStart{Mainmatter}%
  }
  \renewcommand\backmatter{%
    \if@mainmatter\ccaStructEnd{Mainmatter}\fi%
    \cleardoublepage
    \cchResetNesting
    \global\@mainmatterfalse
    \global\@frontmatterfalse
    \global\@backmattertrue
    \ccaVstructStart{Backmatter}%
  }
\fi% \ifarticle
```

6 Loading other CoCoTEX Modules

6.1 coco-accessibility

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

```
\RequirePackage{coco-accessibility}
```

6.2 coco-script

Inclusion of the script module which also loads the babel package

```
\ifLuaTeX
\RequirePackage{coco-script}
\else
\RequirePackage{babel}
\fi
```

6.3 coco-headings

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

6.4 coco-floats

Inclusion of the float module

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

6.5 coco-title

Inclusion of the title page module

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

6.6 coco-notes

Inclusion of the end-/footnotes module

```
\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:

```
\RequirePackage{index}
\makeindex
```

7.2 Hyperref

The hyperref package allows some limited interactiveness of PDF files insofar as that internal and external references become click-able.

PDF/X standards, however, must not be interative, so we disable all hyperref markup by invoking the `nohyperref` package, which disables all `hyperref` markup without rendering the respective macros undefined. This way, we can generate linked and unlinked PDFs from the same sources without the need to remove makros from the tex files.

```
\if\cc@pdf@std X
  \let\href\relax
  \RequirePackage{nohyperref}
\else
```

For PDF/A and PDF/UA, we enable/allow linking by invoking the normal hyperref package…

```
\RequirePackage{hyperref}
```

…with some preset options:

```
\hypersetup{%
```

first, we want links to be breakable

```
    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.

```
   ,linktoc=none%
```

pdf broders are controlled via the coco-frame module, if desired

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

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

```
   ,pdfencoding=unicode
   ,unicode=true
```

Bookmarks are numbered and open by default.

```
   ,bookmarksnumbered=true%
   ,bookmarksopen=false%
```

index is linked by default unless the `no-hyperindex` class option is set.

```
   ,hyperindex=\ifx\cc@no@hyperindex\relax false\else true\fi
}
\fi
```

8 End of Document Class Hook

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

```
\ccAfterClassHook
```

```
</class>
```

Module 2

coco-kernel.dtx

```
<*kernel>
```

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

1 Preamble

```
%%
%% CoCoTeX Kernel
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-kernel}
   [2024/12/13 v0.5.0 cocotex kernel]
```

Before we do anything, we check if the user uses a (more or less) current LATEX kernel version. If not, we issue a hard error message. The pivot that separates "new" from "old" is June 1, 2020.

```
\ifx\IfFormatAtLeastTF\@undefined
  \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}%
\fi
\IfFormatAtLeastTF{2020/06/01}{}%
  {\PackageError{CoCoTeX Kernel}
    {LaTeX kernel too old!}
    {CoCoTeX v0.5.0 and newer needs at least LaTeX kernel version 2023/11/01!}}
```

1.1 Hard dependencies

```
\RequirePackage{etoolbox}
```

Default hook label for CoCoTEX modules:

```
\SetDefaultHookLabel{cc}
```

1.2 Package Options

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

```
\newif\if@cc@debug \@cc@debugfalse
\ExplSyntaxOn
\keys_define:nn { cocotex/kernel }
```

```
{
  debug .code:n = {\global\@cc@debugtrue}
}
```

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

Implies `debug`.

```
\global\let\debug@domain@list\relax
\keys_define:nn { cocotex/kernel }
{
  debug-domain .code:n = {%
    \global\@cc@debugtrue
    \if!#1!\else
      \def\do##1{\listadd\debug@domain@list{##1}}%
      \docsvlist{#1}%
    \fi
  }%
}
```

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

```
\let\cc@prefix\@empty
\keys_define:nn { cocotex/kernel }
{
  prefix .code:n = {\gdef\cc@prefix{#1}},%
  prefix .initial:n = {}
}
\ProcessKeyOptions[cocotex/kernel]
```

2 Exception handlers

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

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

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

```
\def\ccPackageError#1#2#3#4{%
   \GenericError{%
      (#1)\@spaces\@spaces\@spaces\@spaces
   }{%
      [CoCoTeX #1 #2 Error] #3%
   }{}{#4}%
}
```

`\cc@patch@error` issues an error when a CoCoTeX module `{#1}` could not patch macro `{#2}`.

```
\def\cc@patch@error#1#2{%
  \ccPackageError{#1}{compatibility}
  {Could not patch \noexpand#2}
  {You probably use a LaTeX package that re-defines the \noexpand#1 control sequence. It is
       apparently not compatbile with CoCoTeX. Sorry}}
```

`\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

```
\def\ccPackageWarning#1#2#3{%
   \GenericWarning{%
      (#1)\@spaces\@spaces\@spaces\@spaces
   }{%
      [CoCoTeX #1 \if!#2!\else#2 \fi Warning] #3%
   }%
}
```

`\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

```
\def\ccPackageInfo#1#2#3{%
  \GenericInfo{%
    (#1)\@spaces\@spaces\@spaces\@spaces
  }{%
    [CoCoTeX #1\if!#2!\else\space#2\fi] #3%
  }%
}
```

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

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

```
\if@cc@debug
  \def\do#1{\csgdef{cc@debug@#1@message}##1{\typeout{[CoCo #1 Debug]\space##1}}}%
  \dolistloop{\debug@domain@list}%
  \def\cc@debug@@message#1{\typeout{[CoCo Debug]\space#1}}
\else
  \let\cc@debug@@message\@gobble
\fi
```

`\ccDebugMsg` prints a domain specific debug message.

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

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

The following debug domain filters are currently used:

`a11y`	Messages related to PDF tagging and accessibility features
`eval`	final values of Type expansions

`inheritance` Inheritance mechanism of Types

```
\def\ccDebugMsg{\cc@opt@empty\cc@debug@msg}
\def\cc@debug@msg[#1]#2{%
  \expandafter\ifx\csname cc@debug@#1@message\endcsname\relax
    \@gobble{#2}%
  \else
    \csname cc@debug@#1@message\endcsname{#2}%
  \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**rans**p**ect, the XML conversion tool in the backend of *xerif*). After CoCoTeX became stand-alone, the tp prefix became obsolete, but the converters running at the time needed to be backward-compatible. Therefore, all xerif-bound CoCoTeX instances still set this macro to ensure user-level macros bear the tp-prefix.

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

`\if@cc@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 struff, or if it is just processed.

```
\newif\if@cc@is@final \@cc@is@finalfalse
\AtBeginDocument{\@cc@is@finaltrue}
```

`\ccWhenAlly` is a stub that eats its argument. It only does stuff when the `coco-accessibility` package is loaded, which we cannot know, yet.

```
\let\ccWhenAlly\@gobble
```

`\ccUnlessAlly` is a stub that does nothing.

```
\let\ccUnlessAlly\@iden
```

`\ccIfAlly` is the same as `\ccWhenAlly`, but it takes two arguments, one for the *true* case and a second for the *false* case. We default to the *else* case, so we always gobble the first argument. This will be altered if the `coco-accessibility` package is loaded later.

```
\let\ccIfAlly\@gobble
```

4 Containers

Containers are the package's core data structure. They are basicly sets of properties that are processed in the same way.

`\ccDeclareContainer` is the constructor for new Containers.

`{#1}` is the Container's name
`{#2}` is its body, which conists of Inheritance instructions, Type and Env declarations.

```
\def\cc@warning@spaces{\space\space\space\space\space\space\space\space\space\space\space\space\
    space\space\space\space\space\space\space\space\space\space\space\space\space}%
\long\def\ccDeclareContainer#1#2{%
  \ifcsdef{cc@container@#1}
    {\ccPackageWarning{Kernel}{}{Re-declaring Container `#1'^^J%
\cc@warning@spaces All Type settings up to this point will remain!}}
    {\csdef{cc@container@#1}{}}%
  \csdef{cc@cur@cont}{#1}%
```

We want the declarator macros to be only allowed inside the `\ccDeclareContainer` macro.

```
  \begingroup
```

`\ccInherit` The inherit mechanism is dynamic, i.e., we can load multiple type declarations from multiple containers at once.

`{#1}` is a comma-separated list of Types that should be inherited
`{#2}` is a comma-separated list of Container names which the Types should be inherited from

```
    \def\ccInherit##1##2{\cc@inherit{##1}{##2}{#1}\ignorespaces}%
```

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

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

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

`\ccDeclareEnv` Each container usually is realised as a LaTeX environment. The `\ccDeclareEnv` macro is used to set up this environment. Usually, the environment has *the same name as the Container*.

`[#1]` overrides the environment's name. However, keep in mind that the Container's name is not changed by renaming the corresponding environment.
`{#2}` is used for the stuff done at the environment's beginning
`{#3}` is the stuff done at the environment's end

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

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

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

```
    \ccDeclareType{Attributes}{}%
    #2%
  \endgroup
\ignorespaces}
```

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

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

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

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

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

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

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

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

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

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

`{#1}` is a comma-separated list of things to be inherited
`{#2}` is the Container-list that should be inherited from
`{#3}` is the name of the inherting Container

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

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

```
\def\cc@parse@inherit #1,#2,\@nil #3,#4,\@nil #5\@@nil{%
  \let\next\relax
  \if!#1!\else
    \if!#3!\else
      \cc@do@inherit{#1}{#3}{#5}%
      \def\@argii{#2}\def\@argiv{#4}%
      \ifx\@argii\@empty
        \ifx\@argiv\@empty\else
          \def\next{\cc@parse@inherit #1,,\@nil #4,\@nil #5\@@nil}%
        \fi
      \else
        \ifx\@argiv\@empty
          \def\next{\cc@parse@inherit #2,\@nil #3,,\@nil #5\@@nil}%
        \else
          \def\next{%
            \cc@parse@inherit #1,,\@nil #4,\@nil #5\@@nil
            \cc@parse@inherit #2,\@nil #3,#4,\@nil #5\@@nil
          }%
        \fi\fi\fi\fi
  \next}
```

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

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

```
\def\cc@do@inherit#1#2#3{%
  \ccDebugMsg[inheritance]{#3 inherits #1 from #2.}%
  \ccCheckParent{#2}{#3}%
  \expandafter\ifx\csname cc@type@#1@#2\endcsname\relax
    \ccPackageError{Kernel}{Type}{Type `#1' was not declared}{Type `#1' was not declared for
        Container `#2'.}%
  \else
    \edef\x{\noexpand\csgappto{cc@type@#1@#3}}%
    \expandafter\x\expandafter{\csname cc@type@#1@#2\endcsname}%
    \ccDebugMsg[inheritance]{value cc@type@#1@#3:^^J \csmeaning{cc@type@#1@#3}}%
  \fi
  \ignorespaces}
```

5 Components

5.1 Simple Components

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

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

`[#1]` is the Component's identifier. If omitted, `{#1}` is the same as `{#2}`.
`{#2}` is the Component's name
`{#3}` is code that is executed *before* assignment of the user's value
`{#4}` is code that is executed *after* assignment of the user's value

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

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

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

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

`\ccCompWarning` issues a warning if a value is assigned to a non-existing Component. `{#1}` is the name of the unknown Component.

```
\def\ccCompWarning#1{\ccPackageWarning{Kernel}{}{Assigning value to previously undeclared^^J%
    \cc@warning@spaces Component `#1'. Declaring now.}}
```

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

`{#1}` is the Component's name
`{#2}` is the Instance value.

```
\long\protected\def\ccComponent#1#2{%
  \ifx\cc@is@counted\relax
    \ifcsdef{cc@\cc@cur@cont @#1}{}
      {\ccCompWarning{#1}\cc@def@counted@comp{\cc@counted@comp@scheme{#1}}{#1}{}{}}%
    \csgdef{cc@\cc@cur@cont @\cc@counted@comp@scheme{#1}}{#2}%
  \else
    \ifcsdef{cc@\cc@cur@cont @#1}{}
      {\ccCompWarning{#1}\ccDeclareComponent{#1}{}{}}%
    \csdef{cc@\cc@cur@cont @#1}{#2}%
  \fi\ignorespaces}
```

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

`{#1}` is the Component's name
`{#2}` is the Instance value.

```
\long\protected\global\def\ccGlobalComponent#1#2{%
  \ifx\cc@is@counted\relax
    \ifcsdef{cc@\cc@cur@cont @#1}{}
      {\ccCompWarning{#1}\cc@def@counted@comp{\cc@counted@comp@scheme{#1}}{#1}{}{}}%
    \csgdef{cc@\cc@cur@cont @\cc@counted@comp@scheme{#1}}{#2}%
  \else
    \ifcsdef{cc@\cc@cur@cont @#1}{}
       {\ccCompWarning{#1}\ccDeclareGlobalComponent{#1}{}{}}%
    \csgdef{cc@\cc@cur@cont @#1}{#2}%
  \fi\ignorespaces}
```

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

```
\long\protected\def\ccComponentEA#1#2{%
  \def\x{\ccComponent{#1}}\expandafter\x\expandafter{#2}%
  \ignorespaces}
```

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

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

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

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

\ccgdefFromComp is the global variant of \ccdefFromComp.

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

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

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

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

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

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

{#1}	is a scope quantifier (either 'e' or 'x')
{#2}	is a CS token
{#3}	is the name of a component

```
\long\def\cc@store@comp#1#2#3{%
  \edef\@tempa{\expandonce{\csname protected@#1def\endcsname}\noexpand#2}%
  \protected@edef\@tempb{\csname cc@\cc@cur@cont @#3\endcsname}%
   \ifx\@tempb\relax
     \let#2\relax
   \else
     \expandafter\@tempa\expandafter{\@tempb}%
  \fi
  \ignorespaces}
```

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

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

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

```
\def\ccGetComp{\@ifstar\cc@sget@comp\cc@get@comp}
\def\cc@get@comp#1{%
  \ccWhenComp{#1}
    {\ccWhenAlly{\ccaStructStart{P}}%
     \ccUseComp{#1}%
     \ccWhenAlly{\ccaStructEnd{P}}%
     \par}}
```

`\ccGetComp*` The starred version of `\ccGetComp` supresses automated tagging for that Component (only if accessibility features are active).

```
\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.

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

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

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

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

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

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

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

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

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

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

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

`{#1}` is the name of the Container
`{#2}` is the name of the Component
`{#3}` is the comparison value
`{#4}` is the `then` branch
`{#5}` is the `else` branch

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

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

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

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

```
\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

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

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

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

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

```
\def\cc@check@empty{\cc@opt@empty\@cc@check@empty}%]
\def\@cc@check@empty[#1]#2#3#4{%
  \ccIfComp{#4#3}
    {\ccIfCompEmpty{#4#3}
      {\expandafter\global\expandafter\let\csname cc@#2@#4#3\endcsname\relax}
      {}}
    {\ccIfComp{#1#3}
      {\expandafter\expandafter\expandafter\let\expandafter\csname cc@#2@#4#3\expandafter\
          endcsname\csname cc@#2@#1#3\endcsname}
      {}}}
```

5.2 Counted Components

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

Component Groups

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

```
\def\ccDeclareComponentGroup{\cc@opt@empty\cc@declare@component@group}%
\def\cc@declare@component@group[#1]#2#3{%
 \csnumgdef{cc#2Cnt}{\z@}%
 \def\@argi{#1}\ifx\@argi\@empty
  \csgdef{cc@type@Attributes@\cc@cur@cont @cc#2}{}%
 \else
  \csgappto{cc@type@Attributes@\cc@cur@cont @cc#2}{#1}%
 \fi
 \long\csdef{\ccPrefix#2}{\cc@opt@empty{\csname cc@group@#2\endcsname}}%
 \long\csdef{cc@group@#2}[##1]{%
  \def\cc@cnt@grp{cc#2}%
  \csxdef{cc#2Cnt}{\expandafter\the\expandafter\numexpr\csname cc#2Cnt\endcsname+\@ne\relax}%
  \if!##1!\else
   \ccEvalAttributes[\cc@cur@cont @cc#2]{##1}%
   \csgdef{cc@\cc@cur@cont @#2-\csname cc#2Cnt\endcsname @attrs}{##1}%
```

```
    \fi
    #3%
    \csname cc@group@#2@hook\endcsname
    \ignorespaces
  }%
  \csdef{end\ccPrefix#2}{{\ccToggleCountedConditionals\csuse{cc@compose@group@#2}}\aftergroup\
      ignorespaces}%
}
```

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

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

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

```
\def\ccDeclareGroupHandler#1#2{%
  \ifcsdef{cc@group@#1}
    {\ifcsdef{cc@compose@group@#1}
      {\csgappto{cc@compose@group@#1}{#2}}
      {\csgdef{cc@compose@group@#1}{#2}}}
    {\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.}}%
\ignorespaces}
```

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

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

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

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

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

```
\def\ccUsePropFrom#1#2#3{%
  \begingroup
    \@tempcnta\numexpr#2\relax
    \letcs\ccTotalCount{cc#1Cnt}%
    \def\cc@cnt@grp{cc#1}%
    \ccToggleCountedConditionals
    \csnumdef{cc#1Cnt}{\the\@tempcnta}%
    \ccCurCount=\the\@tempcnta\relax%
    \csname cc@\cc@cur@cont @#3\endcsname%
  \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.

```
\newcount\ccCurCount
```

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

```
\def\cc@assign@res#1{%
  \ifx\cc@iterate@res\@undefined
    \cslet{#1}\relax
  \else
    \ifx\cc@iterate@res\relax
      \cslet{#1}\relax
    \else
      \expandafter\csname #1\expandafter\endcsname\expandafter{\cc@iterate@res}%
    \fi
  \fi
  \global\let\cc@iterate@res\relax
}
```

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

```
\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}.

```
\def\ccComposeCollection#1#2#3{%
  \ccIfComp{#3}
    {\cslet{cc@used@#3@override}\@empty}
    {\cc@compose@collection{#1}{#2}%
     \cc@assign@res{\ccPrefix#3}}}
```

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

```
\def\ccApplyCollection#1#2#3{%
  \ccIfComp{#3}
    {\cslet{cc@used@#3@override}\@empty}
    {\cc@apply@collection{#1}{#2}%
     \cc@assign@res{\ccPrefix#3}}}
```

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

```
\def\cc@compose@collection#1#2{%
  \ccaProtect
  \expandafter\ifnum\csname cc#1Cnt\endcsname > \z@\relax
    \edef\cc@iterate@res{%
```

```
      \noexpand\bgroup
        \noexpand\def\noexpand\ccTotalCount{\csname cc#1Cnt\endcsname}%
        \noexpand\ccToggleCountedConditionals
        \noexpand\def\noexpand\cc@cur@cont{\cc@cur@cont}%
        \noexpand\def\noexpand\cc@cnt@grp{cc#1}}%
      \expandafter\@tempcntb=\csname cc#1Cnt\endcsname\relax
      \cc@iterate{\@tempcnta}{\@ne}{\@tempcntb}{%
        \edef\@tempb{%
          \noexpand\begingroup
            %% top-level counter for user interaction
            \noexpand\ccCurCount=\the\@tempcnta\relax
            %% evaluating the current group's Attributes
            \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}
              {\noexpand\ccEvalAttributes[\cc@cur@cont @cc#1]
                {\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}%
              %% internal counter for macro grabbing
              \noexpand\csnumdef{cc#1Cnt}{\ccCurCount}%
              \noexpand\ccUseProperty{#2}%
              \noexpand\endgroup
          }%
          \expandafter\expandafter\expandafter\def
          \expandafter\expandafter\expandafter\cc@iterate@res
          \expandafter\expandafter\expandafter{\expandafter\cc@iterate@res\@tempb}%
        }%
        \expandafter\def\expandafter\cc@iterate@res\expandafter{\cc@iterate@res\egroup}%
  \fi
  \ccaEnable}
```

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

```
\def\cc@apply@collection#1#2{%
  \ccaProtect
  \begingroup
    \global\let\cc@iterate@res\relax
    \letcs\ccTotalCount{cc#1Cnt}%
    \cc@iterate{\@tempcnta}{\@ne}{\ccTotalCount}{%
      \bgroup
        \ccToggleCountedConditionals
        \def\cc@cnt@grp{cc#1}%
        \csnumdef{cc#1Cnt}{\the\@tempcnta}%
        \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}
          {\ccEvalAttributes[\cc@cur@cont @cc#1]
            {\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}%
        \ccCurCount=\the\@tempcnta
        \protected@xdef\@tempb{\csname cc@\cc@cur@cont @#2\endcsname}%
        \@temptokena \expandafter{\@tempb}%
        \def\@tempc{\csgappto{cc@iterate@res}}%
        \expandafter\@tempc\expandafter{\@tempb}%
      \egroup
    }%
  \endgroup
  \ccaEnable
}
```

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

`{#1}`	is a prefix to the def command, e.g., `\global` or `\protected`
`{#2}`	is a CS token
`{#3}`	is the Name of the Counted Component

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

```
\def\cc@comp@edef{\cc@opt@empty\@cc@comp@edef}
\def\@cc@comp@edef[#1]#2#3#4{%
  \cc@apply@collection{#3}{#4}%
  \ifx\cc@iterate@res\relax
    #1\let#2\relax%
  \else
    \def\@tempa{#1\def#2}%
    \cc@assign@res{@tempa}%
  \fi
}
```

```
\def\cc@comp@def{\cc@opt@empty\@cc@comp@def}
\def\@cc@comp@def[#1]#2#3#4{%
  \cc@compose@collection{#3}{#4}%
  \ifx\cc@iterate@res\relax
    #1\let#2\relax%
  \else
    \def\@tempa{#1\def#2}%
    \cc@assign@res{@tempa}%
  \fi
}
```

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

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

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

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

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

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

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

Declaring Counted Component

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

{#1} the name of the Counted Component.

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

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

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

```
\def\ccDeclareCountedComponent#1{%
  \cc@def@counted@comp
    {\cc@counted@comp@scheme{#1}}
    {#1}
    {}
    {\expandafter\global}%
  \ignorespaces}
```

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

`{#1}` is the internal name of the Component which is composed out of the group name, the value of the group counter and the user-level macro name `{#2}`
`{#2}` is the name of the Counted Component
`{#3}` is some custom code passed to the second argument of `\ccDeclareComponent`
`{#4}` is a modifier to the internal macro definition.

```
\def\cc@def@counted@comp#1#2#3#4{%
  \ccDeclareComponent[#1]{#2}
    {\bgroup#3\expandafter\global}
    {\def\@tempa{{@cc@reset@components@\cc@cur@cont}}%
     \edef\@tempb{\noexpand\csgundef{cc@\noexpand\cc@cur@cont @#1}}%
     \expandafter\expandafter\expandafter\csgappto\expandafter\@tempa\expandafter{\@tempb}%
     \egroup}%
   \ignorespaces
   #4\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#2\endcsname{\csname cc@\
        cc@cur@cont @#1\endcsname}%
   \ignorespaces}
```

Resetting Counted Component

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

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

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

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

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!

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

```
  \long\def\ccIfComp##1{%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
         csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@secondoftwo\else\expandafter\
         @firstoftwo\fi%
  }%
  \long\def\ccWhenComp##1{%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
        csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@gobble\else\expandafter\
        @firstofone\fi%
  }%
  \long\def\ccUnlessComp##1{%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\ifx\
        csname cc@\cc@cur@cont @##1\endcsname\relax\expandafter\@firstofone\else\expandafter\
        @gobble\fi%
  }%
  \long\def\ccIfCompEmpty##1{%
    \expandafter\expandafter\expandafter\ifx\csname cc@\cc@cur@cont @##1\endcsname\cc@long@empty
        \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}%
  \ccToggleCountedConditionalsHook% legacy
}
```

6 Properties

6.1 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.

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

`\ccAppToProp` can be used add material to the *end* of an existing Property value.

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

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

`\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 the previous value of that Property

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

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

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

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

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

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

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

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

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

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

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

6.2 Using Properties

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

```
\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`.

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

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

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

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

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

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

```
\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 instrunctions (see below).

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

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

```
\def\ccIfStrEqual#1#2{%
  \edef\@argi{#1}\edef\@argii{#2}%
  \expandafter\expandafter\expandafter\ifstrequal
    \expandafter\expandafter\expandafter{\expandafter\@argi\expandafter}%
      \expandafter{\@argii}}
```

Local Property Overrides

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

`{#1}` is the CS token representing a method to alter the property (`\ccSetProperty`, `\ccAppToProp`, or `\ccPreToProp`)
`{#2}` is the name of the Property to be altered
`{#3}` is the new (or added) Value

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

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

They all take two arguments:

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

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

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

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

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

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

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

6.3 Processing Instructions

In general, processing instructions are commands that are only visible to a specific process and ignored by others. In CoCoTEX, 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.

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

6.4 Property Conditionals

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

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

\ccWhenProp is the same as \ccIfProp but omits the else-branch.

```
\long\def\ccWhenProp#1#2{%
  \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\relax\else
    \expandafter\ifx\csname cc@\cc@cur@cont @#1\endcsname\cc@long@empty\else#2\fi
  \fi
\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!

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

7 Helper macros

7.1 Handling of Optional Arguments

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

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

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

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

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

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

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

7.2 Iterators

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

```
\long\def\cc@iterate{\@ifnextchar[{\@cc@iterate}{\@cc@iterate[\@ne]}}%]
\long\def\@cc@iterate[#1]#2#3#4#5{%
```

```
  #2=#3\relax%
  \expandafter\ifnum#2>#4\relax%
  \else
    #5%
    \advance#2 by #1\relax
    \cc@iterate[#1]{#2}{\the#2}{#4}{#5}%
  \fi}%
```

7.3 Attributes

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

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

```
\begingroup
\catcode`"=12
```

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

`{#1}` is the domain of the attribute
`{#2}` is the raw attribute list

```
\gdef\ccParseAttributes#1#2{%
  \if!#1!\else
    \if!#2!\else
      \def\cc@cur@domain{#1}%
      \cc@parse@attributes #2,,\@nil
    \fi\fi}
```

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

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

```
\gdef\cc@parse@attributes #1,#2,\@nil{%
  \if!#1!\else
    \cc@parse@kv#1==\@nil
    \if!#2!\else
      \cc@parse@attributes#2,\@nil
    \fi\fi}
```

and

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

```
\gdef\cc@parse@kv#1=#2=#3\@nil{%
  \edef\@argii{#2}%
  \ifx\@argii\@empty
    \expandafter\let\csname cc@\cc@cur@domain @attr@#1\endcsname\@empty%
  \else
    \ifx #2 =\else
      \expandafter\def\csname cc@\cc@cur@domain @attr@#1\endcsname{#2}%
    \fi
  \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`.

```
\gdef\cc@parse@csv#1#2{%
  \if!#1!\else
    \let\cc@parser@callback#1%
    \edef\cc@tempa{\csname #2\endcsname}%
    \ifx\cc@tempa\@empty\else
      \expandafter\cc@@parse@csv\cc@tempa,,\@nil
    \fi
  \fi}
```

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

```
\gdef\cc@@parse@csv #1,#2,\@nil{%
  \if!#1!\else
    \cc@parser@callback{#1}%
  \fi
  \if!#2!\else
    \cc@@parse@csv#2,\@nil
  \fi
  \ignorespaces}
\endgroup
```

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

`[#1]`	is the Attribtue Domain (defaults to the current Container name)
`{#2}`	is the Container Instance's raw Attribute list.

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

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

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

If so, we parse the Attribute list.

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

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

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

`{#1}`	is the name of the attribute (i. e., the part before the '=')
`[#2]`	is code that is called when the Attribute *does not* occur in the Attribute list `{#1}`
`{#3}`	is code that is called when the Attribute *does* occur in the Attribute list `{#1}`.

```
    \def\ccDeclareAttributeHandler{%
      \let\cc@is@starred\@undefined
      \@ifstar
      {\let\cc@is@starred\relax\cc@declare@attribute@handler}
      {\cc@declare@attribute@handler}}%
  \def\cc@declare@attribute@handler##1{\cc@opt@empty{\@cc@declare@attribute@handler{##1}}}%
  \def\@cc@declare@attribute@handler##1[##2]##3{%
    \let\ccAttrVal\relax
    \ifx\cc@is@starred\relax
      \ccIfAttrIsSet{#1}{##1}{##3}{##2}%
    \else
      \ccIfAttr{#1}{##1}
        {\letcs\ccAttrVal{cc@#1@attr@##1}##3}
        {##2}%
    \fi\ignorespaces
  }%
```

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

```
  \ccEvalType[#1]{Attributes}%
\else
```

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

```
  \if!#2!\else
```

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

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

`\ccGetAttribute` returns the value of an attribute.

`{#1}` is the attribute domain
`{#2}` is the attribute name

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.4 Style Classes

Style Classes are locally usable sub-Containers.

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

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

```
\long\def\ccDeclareClass{\@ifnextchar [{\@cc@set@class}{\@cc@set@class[default]}}%]
\long\def\@cc@set@class[#1]#2{\cc@opt@empty{\cc@set@class[#1]{#2}}}%
\long\gdef\cc@default@class@default{}
\long\def\cc@set@class[#1]#2[#3]#4{%
  \def\@argii{#2}\ifx\@argii\@empty\let\@argii\cc@str@default\fi%
  \if!#3!\else
    \expandafter\long\expandafter\def\csname cc@#1@class@\@argii @parent\endcsname{#3}%
  \fi
  \expandafter\long\expandafter\def\csname cc@#1@class@\@argii\endcsname{#4}%
\ignorespaces}
```

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

`{#1}`	is the Style Class name
`{#2}`	is the Container name

```
\def\ccUseStyleClass#1#2{%
  \expandafter\ifx\csname cc@#2@class@#1\endcsname\relax
    \expandafter\ifx\csname cc@default@class@#1\endcsname\relax
      \PackageError{cocotex.cls}{Class `#1' with scope `#2' not defined!}{Please declare the
          class `#1'!}%
    \else
      \PackageInfo{cocotex.cls}{Class `#1' with scope `#2' not defined; using unscoped class.}%
    \fi
  \fi
  \csname cc@default@class@#1\endcsname%
  \expandafter\ifx\csname cc@#2@class@#1@parent\endcsname\relax\else
    \expandafter\ccUseStyleClass\expandafter{\csname cc@#2@class@#1@parent\endcsname}{#2}%
  \fi
  \csname cc@#2@class@#1\endcsname\ignorespaces}
```

7.5 The CoCoTEX Logo

`\CoCoTeX` the CoCoTEX Logo.

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

```
</kernel>
```

Module 3
coco-common.dtx

```
<*common>
```

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

```
%%
%% module for CoCoTeX that provides some commonly used base macros.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-common}
   [2024/12/13 v0.5.0 CoCoTeX common module]
```

Default hook label for CoCoTEX modules:

```
\SetDefaultHookLabel{cc}
```

1 Package options

1.1 Accessibility Features

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

```
\def\cc@str@cmyk{cmyk}
\def\cc@str@rgb{rgb}
\def\cc@str@gray{gray}
\def\cc@str@natural{natural}
\let\cc@color@enc\cc@str@cmyk
\ExplSyntaxOn
\keys_define:nn { cocotex/common }
{
  color-enc .choice:,
  color-enc / srgb .code:n = { \global\let\cc@color@enc\cc@str@rgb },
  color-enc / rgb .code:n = { \global\let\cc@color@enc\cc@str@rgb },
  color-enc / gray .code:n = { \global\let\cc@color@enc\cc@str@gray },
  color-enc / grey .code:n = { \global\let\cc@color@enc\cc@str@gray },
  color-enc / cmy .code:n = { \global\let\cc@color@enc\cc@str@cmyk },
  color-enc / cmyk .code:n = { \global\let\cc@color@enc\cc@str@cmyk },
  color-enc / natural .code:n = { \global\let\cc@color@enc\cc@str@natural },
  color-enc / none .code:n = { \global\let\cc@color@enc\relax },
  color-enc .initial:n = cmyk
}
```

1.2 Output Options

The silent options attempts to supress most unneccessary messages sent to the shell.

```
\newif\if@cc@silent \@cc@silentfalse
\keys_define:nn { cocotex/common }
{
  silent .code:n = { \global\@cc@silenttrue }%
}
```

Process the options

```
\ProcessKeyOptions[cocotex/common]
```

First, we set up the requested color model with both the newer LaTeX Kernel and the xcolor package.

```
\ifx\cc@color@enc\relax\else
  \tl_gset:Nn \l_color_fixed_model_tl {\cc@color@enc}%
  \PassOptionsToPackage{\cc@color@enc}{xcolor}%
\fi%
\ExplSyntaxOff
```

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

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

2 Commonly Used Low-Level Macros and Registers

If CoCoTEX is used in conjunction with xerif[1], 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.

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

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

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

2.1 Hard Dependencies

Hard requirements for all CoCoTEX modules:

```
\RequirePackage{xcolor}
```

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

```
\RequirePackage{graphicx}
\if@cc@silent\let\Gin@log\@gobble\fi
\DeclareGraphicsRule{.EPS}{eps}{.EPS}{}
```

[1] See https://github.com/transpect/xerif/

2.2 Common Variables

String Variables for Value Comparisions

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

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

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

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

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

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

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

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

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

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

Box Registers

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

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

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

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

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

Temporary Length and Skip Registers

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

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

2.3 Helper macros

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

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

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

```
\def\cc@afterbox{%
  \everypar{%
    \if@nobreak
      \@nobreakfalse
      \clubpenalty \@M
      \if@afterindent \else
        {\setbox\z@\lastbox}%
        \everypar{}%
      \fi
    \else
      \clubpenalty \@clubpenalty
      {\setbox\z@\lastbox}%
      \everypar{}%
    \fi}}
```

`\ccSanitizeStr` is used to clean macro content from any non-expansible markup. It is intended to sanitize strings before they are sent to the PDF writer, like DocumentInfo strings or alternative texts.

`{#1}` is the macro that is defined to hold the sanitized text
`{#2}` is the content to be sanitized

```
\ExplSyntaxOn
\cs_new:Npn\ccSanitizeStr #1 #2
  {
    \edef #1 { \text_purify:n { #2 } }
  }
\ExplSyntaxOff
```

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.

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

`\hackfor` intended to hide line breaking macros.

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

`\Hack` intended to mask page breaking macros.

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

`\Hackfor` intended to hide page breaking macros.

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

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

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

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

```
\def\ccGobble{%
  \renewcommand\footnote[2][\the\c@footnote]{\def\@thefnmark{##1}\@makefnmark}%
  \renewcommand\index[2][]{}%
  \renewcommand\marginpar[2][]{}%
  \renewcommand\glossary[2][]{}%
  \let\hypertarget\@gobbletwo
  \let\label\@gobble
  \@cc@is@finalfalse
}%
```

2.5 Arithmetics

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

```
\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.

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

`\minusvspace` Counterpart to LaTeX's `\addvspace`: if the value of `\minusvspace` is larger than `\lastskip`, `\lastskip` is used. Otherwise, the value of `\minusvspace` is used.

```
\def\@xminusvskip{%
  \ifdim\lastskip<\@tempskipb
  \else
    \ifdim\lastskip<\z@
    \else
      \ifdim\@tempskipb<\z@
        \advance\@tempskipb\lastskip
      \fi
      \vskip-\lastskip
      \vskip \@tempskipb
    \fi
  \fi}
\def\minusvspace#1{%
  \ifvmode
     \if@minipage\else
       \ifdim \lastskip =\z@
```

Compatibility to texlive pre 2020:

```
         \ifx\@vspace@calcify\@undefined
           \vskip #1\relax
         \else
           \@vspace@calcify{#1}%
         \fi
       \else
       \setlength\@tempskipb{#1}%
```

```
      \@xminusvskip
     \fi
    \fi
  \else
   \@noitemerr
  \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 LaTeX to evaluate it independently from the actual manner of counting.

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

```
\def\the@cc@thispage{}%
```

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

```
\newcount\cc@abspage \cc@abspage\z@
```

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

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

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

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

The absolute page counter is injected directly into LaTeX's output routine:

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

We split the testing mechanism into two parts.

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

```
\def\ccTestPage{%
  \expandafter\ifx\csname the@cc@thispage\endcsname\@empty
    \gdef\the@cc@atthispage{1}%
  \else
    \expandafter\ifnum \the@cc@thispage=\cc@abspage%
      \begingroup
        \@tempcnta\the@cc@atthispage\relax
        \advance\@tempcnta\@ne\relax
        \xdef\the@cc@atthispage{\the\@tempcnta}%
      \endgroup
    \else
      \gdef\the@cc@atthispage{1}%
    \fi
  \fi
  \xdef\the@cc@thispage{\the\cc@abspage}%
  \let\@cc@currpage\relax
  \expandafter\ifx\csname \cc@cur@cont-\the@cc@thispage-\the@cc@atthispage\endcsname\relax
    \ifodd\cc@abspage\relax\@cc@oddtrue\else\@cc@oddfalse\fi
  \else
    \edef\@cc@currpage{\expandafter\expandafter\expandafter\@firstofone\csname \cc@cur@cont-\
         the@cc@thispage-\the@cc@atthispage\endcsname}%
    \ifodd\@cc@currpage\relax\@cc@oddtrue\else\@cc@oddfalse\fi
  \fi
}
```

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

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

3 Re-Thinking LATEX Core Functions

3.1 Keeping .aux-Files Up-to-Date

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

```
\DeclareRobustCommand*{\ccBreak}{\hfill\break}
\ifx\ccPrefix\@empty\else\cslet{\ccPrefix break}\ccBreak\fi
```

3.2 Content lists

Default LATEX Content Lists

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

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

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

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

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

```
\def\cc@init@l@{\cc@opt@empty\@cc@init@l@}%
\def\@cc@init@l@[#1]#2#3#4{%
  \expandafter\ifx\csname c@#2depth\endcsname\relax
    \expandafter\global\expandafter\newcount\csname c@#2depth\endcsname
    \expandafter\global\csname c@#2depth\endcsname=0\relax
  \fi
  \expandafter\ifx\csname cc@#2@extract@data\endcsname\relax
    \expandafter\let\csname cc@#2@extract@data\endcsname\cc@extract@generic
  \fi
  \expandafter\ifx\csname cc@#2@print@entry\endcsname\relax
    \expandafter\let\csname cc@#2@print@entry\endcsname\cc@print@generic
  \fi
  \expandafter\long\expandafter\gdef\csname cc@l@#4\endcsname##1##2{%
    \ifLuaTeX\suppresslongerror=1\fi
    \expandafter\ifnum \csname c@#2depth\endcsname<#3\relax
    \else
      \bgroup
```

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

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

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

`\ccContentsline` is our version of LaTeX's `\contentsline`.

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

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

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

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

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

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

```
\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 LaTeX headings.

`{#1}`	is the content of the `cc@l@`-Macro

{#2} is the name of the handling Container
{#3} is the Component prefix
{#4} is the name of the Content component

```
\def\cc@expand@l@contents#1#2#3#4{%
  \global\let\cc@tempa\relax
  \sbox\z@{\def\numberline##1{\xdef\cc@tempa{\noexpand\csdef{cc@#2@#3Number}{##1}}}#1}%
  \ifdim\wd\z@>\z@
    \let\numberline\@gobble%
    \protected@csedef{cc@#2@#3#4}{#1}%
    \cc@tempa
  \else
    #1%
  \fi
  \global\let\cc@tempa\relax
}
```

Custom Content Lists

\ccDeclareContentList provides an interface for additional content lists.

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

```
\def\ccDeclareContentList#1#2{%
  \def\cc@add@extra@cl##1{%
    \expandafter\ifx\csname cc@##1@extra@cl\endcsname\relax
      \csgdef{cc@##1@extra@cl}{#1}%
    \else
      \csgappto{cc@##1@extra@cl}{,#1}%
    \fi}%
  \edef\@argii{#2}%
  \cc@parse@csv\cc@add@extra@cl{@argii}%
  \expandafter\newwrite\csname cc@cl@#1\endcsname\relax
}
```

\ccCreateContentListEntries creates entries for Custom Content Lists. It is called during the proccessing 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

```
\def\ccCreateContentListEntries#1#2#3#4#5{%
  \def\cc@add@extra@cl##1{%
    \expandafter\protected@write\csname cc@cl@##1\endcsname
      {\ccGobble\ccaProtect}%
      {\protect\ccContentsline{#2}{#3}{#4}{#5}\protected@file@percent}\relax
    \expandafter\protected@write\csname cc@cl@##1\endcsname
      {\ccaEnable}{}%
  }%
  \ifcsdef{cc@#1@extra@cl}{%
    \cc@parse@csv\cc@add@extra@cl{cc@#1@extra@cl}}{}%
}
```

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 indention 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 cotainer 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 LaTeX runtime. That way, for the next LaTeX 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 LaTeX 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.

```
\def\cc@store@latest#1#2{%
  \expandafter\ifx\csname cc-\cc@cur@cont-#1\endcsname\relax
    \csxdef{cc-\cc@cur@cont-#1}{#2}%
  \else
    \expandafter\ifdim\csname cc-\cc@cur@cont-#1\endcsname<#2\relax
      \csxdef{cc-\cc@cur@cont-#1}{#2}%
    \fi
  \fi
  \expandafter\ifx\csname cc-\cc@cur@cont-#1-local\endcsname\relax
    \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
  \else
    \expandafter\ifdim\csname cc-\cc@cur@cont-#1-local\endcsname<#2\relax
      \csxdef{cc-\cc@cur@cont-#1-local}{#2}%
    \fi
  \fi
```

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

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

```
  \ifcsdef{cc-\cc@cur@cont-#1-stored-trigger}{}
    {\edef\@tempa{%
       \noexpand\immediate\noexpand\write\noexpand\@auxout{%
         \noexpand\string\noexpand\csgdef{cc-\cc@cur@cont-#1}{%
           \noexpand\csname cc-\cc@cur@cont-#1-local\noexpand\endcsname}}}%
     \expandafter\AtEndDocument\expandafter{\@tempa}%
     \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.

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

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

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

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

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

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

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

`{#1}` is the internal Property prefix

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

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

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

Then, we check for `#1indent`.

```
  \ccIfProp{#1indent}
    {\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).

```
      \ccSetProperty{#1hang-number}{%
        \hskip\ccUseProperty{#1indent}%
        \hbox to -\ccUseProperty{#1indent}{%
          \ccIfPropVal{#1number-align}{left}{}{\hss}%
          \ccUseProperty{#1number-format}%
          \ccIfPropVal{#1number-align}{right}{}{\hss}}}%
    \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.

```
\def\cc@calc@margin@left#1#2{%
  \@tempcnta\numexpr#2-\@ne\relax
  \expandafter\ifx\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname\relax
    \@tempdima=-\ccUseProperty{#1indent}\relax%
  \else
    \@tempdima=\dimexpr\csname cc-\cc@cur@cont-#1\the\@tempcnta-margin-left\endcsname-\
         ccUseProperty{#1indent}\relax
  \fi
  \cc@store@latest{#1#2-margin-left}{\the\@tempdima}%
  \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.

```
\def\cc@get@indent{\@ifnextchar[{\@cc@get@indent}{\@cc@get@indent[\cc@calc@margin@left]}}
\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 ultimatly being passed to `\hskip`.

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

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

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

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

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

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

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

```
  {\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.

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

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

```
      {\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.

```
      {\ccIfProp{int-#2indent}
        {\ccSetPropertyX{#2indent}{\ccUseProperty{int-#2indent}}}
        {\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.

```
     #1{#2}{#3}}
```

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

```
    {\ccIfProp{int-#2margin-left}
      {\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.

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

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

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

If `margin-left` is not set,

```
      {\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.

```
        {\ccPropertyLetX{#2margin-left}{#2number-width-level-max}%
         \ccSetPropertyX{#2indent}{-\ccUseProperty{#2number-width-level-max}}}
        {\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.

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

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

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

3.4 Labelling and Cross referencing

CoCoTEX 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.

```
\AtBeginDocument{%
```

Storing the final definitions of \label

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

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

\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!

```
\def\ccCreateLabel#1{%
  \ifx\Hy@MakeCurrentHrefAuto\@undefined\else
    \Hy@MakeCurrentHrefAuto{cc:#1}%
    \Hy@raisedlink{\hyper@anchorstart{\@currentHref}\hyper@anchorend}%
  \fi
  \let\cc@ref@label\relax
  \ccWhenComp{RefLabel}
    {\ccgdefFromComp\cc@ref@label{RefLabel}%
     \expandafter\cc@create@label\expandafter{\cc@ref@label}}%
  \ccIfAttr{\cc@cur@cont}{label}
    {\cc@parse@csv\cc@create@label{cc@\cc@cur@cont @attr@label}}%
    {\ifx\cc@ref@label\relax\cc@create@label{\@currentHref}\fi}}
```

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

{#1} is the label ID

```
\def\cc@create@label#1{%
  \ccIfComp{Number}
  {\ifx\cc@labelname@comp\@undefined
      \def\cc@labelname@comp{Title}%
    \fi
    \begingroup
      \ccGobble\ccaProtect
      \ccgdefFromComp\@currentlabel{Number}%
      \ccgdefFromComp\@currentlabelname{\cc@labelname@comp}%
    \endgroup}%
  {\cc@fallback@anchor}%
  %% leaving this will generate lots of "duplicate destination"
  %% messages from pdfbackend
  %\expandafter\hypertarget\expandafter{#1}{}%
  \@tempswafalse\ifx\cc@saved@ccaStructStart\ccaStructStart\@tempswatrue\fi
  \ccaProtect
  \expandafter\label\expandafter{#1}%
  \if@tempswa\ccaEnable\else\ccaDisable\fi
}
\def\cc@fallback@anchor{\phantomsection}%
```

3.5 Linguistic Name generation and selection

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

`{#1}`	is the language
`{#2}`	is the internal reference name
`{#3}`	is the language specific label

```
\def\ccSetBabelLabel#1#2#3{%
  \def\ccc@lang{#1}%
  \expandafter\def\expandafter\ccc@tempa\expandafter{\expandafter\def\csname #2name\endcsname
      {#3}}%
  \ifdefstring\ccc@lang{german}{%
    \expandafter\addto\expandafter\captionsgerman\expandafter{\ccc@tempa}%
    \expandafter\addto\expandafter\captionsngerman\expandafter{\ccc@tempa}%
  }\relax%
  \ifdefstring\ccc@lang{english}{%
    \expandafter\addto\expandafter\captionsbritish\expandafter{\ccc@tempa}%
    \expandafter\addto\expandafter\captionsUKenglish\expandafter{\ccc@tempa}%
    \expandafter\addto\expandafter\captionsenglish\expandafter{\ccc@tempa}%
    \expandafter\addto\expandafter\captionsamerican\expandafter{\ccc@tempa}%
    \expandafter\addto\expandafter\captionsUSenglish\expandafter{\ccc@tempa}%
  }\relax%
}
```

3.6 Link Generation

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

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

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

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

```
</common>
```

Module 4
coco-accessibility.dtx

This file provides code for the interaction between the CoCoTeX framwork and the `ltpfdfa` 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

```
<*a11y-sty>
```

1.1 General Processing

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

```
%%
%% Accessibility features for \textit{xerif} projects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% requires LuaLaTeX and ltpdfa!
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-accessibility}
    [2024/12/13 v0.5.0 CoCoTeX accessibility module]
```

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.

`\if@cc@do@ally` is a simple switch that indicates active (`\relax`) or inactive (everything else) accessibility features.

```
\newif\if@cc@do@ally \@cc@do@allyfalse
\ExplSyntaxOn
\keys_define:nn { cocotex/a11y }
{
  init .code:n = {\global\@cc@do@allytrue},
```

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

```
  nodetree .code:n = {\let\cca@do@nodetree\relax},
```

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

```
  show-spaces .code:n = {\let\cca@do@showspaces\relax},
```

`\cca@do@dospaces` if `\relax`, add ASCII space characters to the PDF. LaTeX 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-dependend width of that space to keep LaTeX's typeface intact. This is activated by default.

```
}
\let\cca@do@dospaces\relax
\keys_define:nn { cocotex/a11y }
{
  no-spaces .code:n = {\let\cca@do@dospaces\@undefined}
}
```

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

```
\let\cca@do@doparas\relax
\keys_define:nn { cocotex/a11y }
{
  no-paras .code:n = {\let\cca@do@doparas\@undefined}
}
```

Processing the options.

```
\ProcessKeyOptions[cocotex/a11y]
```

`\cca@patch@error` is a generic error message that is thrown whenever a LaTeX 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.

```
\def\cca@patch@error{\cc@patch@error{a11y}}
```

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.

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

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

```
\def\ccWhenAlly{\if@cc@do@ally\expandafter\@firstofone\else\expandafter\@gobble\fi}
```

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

```
\def\ccUnlessAlly{\if@cc@do@ally\expandafter\@gobble\else\expandafter\@firstofone\fi}
```

1.3 Accessibility-specific additions

Loading Further Dependencies

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

```
\ccWhenAlly{%
  \ifluatex\else
    \ccPackageError{a11y}{engine}
      {accessibility features require lualatex!}
      {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.}
  \fi
  \RequirePackage{luatexbase-attr}
  \RequirePackage{atveryend}
```

Additonal Hyperref Setup

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

```
  \AtBeginDocument{%
    \hypersetup{%
      % pdfa=true% already set elsewhere
      ,unicode=true%
      ,pdfinfo={}%
      % ,pdfpagelabels=true% already set elsewhere
      ,pageanchor=true%
    }%
    \Hy@pdfatrue
  }
```

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:

```
  \directlua{ltpdfa = require('ltpdfa')}
  \directlua{ltpdfa.config.final = true}
  \ifinlist{ltpdfa}\debug@domain@list
    {\directlua{ltpdfa.config.debug = true}}
    {\directlua{ltpdfa.config.debug = false}}
  \directlua{ltpdfa.config.nodetree = \ifx\cca@do@nodetree\relax true\else false\fi}
  \directlua{ltpdfa.config.showspaces = \ifx\cca@do@showspaces\relax true\else false\fi}
  \directlua{ltpdfa.config.dospaces = \ifx\cca@do@dospaces\relax true\else false\fi}
  \directlua{ltpdfa.config.doparas = \ifx\cca@do@doparas\relax true\else false\fi}
```

We implement our own auto-close mechanism to end document partitions, so we disable ltpdfa's native one:

```
  \directlua{ltpdfa.tagger.doautoclose = false}
```

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

```
  \directlua{ltpdfa.config.driver = "\luaescapestring{pdftex}"}
```

\cca@lang@id is the IETF language tag for the document's main language.

```
  \AtBeginDocument{%
    \edef\cca@lang@id{\localeinfo*{tag.bcp47}}% tag.bcp47
    \directlua{ltpdfa.config.lang = '\luaescapestring{\cca@lang@id}'}%
    \directlua{ltpdfa.init()}%
  }%
```

Initial setup of ltpdfa

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

We need the absolute last page of the document

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

1.4 Generic Macro to Declare Accessibility Features

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

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

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

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

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

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

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

```
\def\cca@protected@defs{}
```

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

`\ccaDisable` disables all ltpdfa commands

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

and

`\ccaEnable` enables all ltpdfa commands.

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

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

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

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

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

\DeclareAccessibilityCommand is the wrapper for our interface macros. It has the same argument signature as LaTeX's \newcommand*, albeit without the whole checking for already defined control sequences.

```
\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.

```
\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:

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

```
\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.

```
\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.

```
  \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.

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

```
  \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.

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

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

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

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

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

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

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

Some macros from `ltpdfa.sty`:

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

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

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

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

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

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

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

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

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

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

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

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

\ccaPstructEnd ends an unattributed tagging area.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\ccaAddNumbering` ???

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

1.5 Language Tagging

`\ccaAddLang` adds a /Lang(uage) attribute to the current node with the value `{#1}`.

```
\DeclareAccessibilityCommand{\ccaAddLang}[1]{\directlua{ltpdfa.tagger.setLang('\luaescapestring
    {#1}')}}
```

The following code patches the unstarred `\foreignlanguage` command in such a way that it automatically tags its content with a `</Span/>` and adds a `/Lang` attribute to the Tag that is colleted from the language definition file of the currently activated language.

```
\ccWhenAlly{%
  \AddToHook{babel/*/foreign}{%
    \pretocmd\BabelText
      {\ccaVstructStart{Span}\ccaAddLang{\localeinfo{tag.bcp47}}}
      {}{\cca@patch@error\BabelText}%
    \apptocmd\BabelText
      {\ccaVstructEnd{Span}}
      {}{\cca@patch@error\BabelText}%
    }%
  \AtBeginDocument{\ActivateGenericHook{babel/*/foreign}}%
}% /ccWhenAlly
```

1.6 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:

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

1.7 Alternative Text Elements

Some structure elements in the PDF/UA standards require alternative texts.

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

It is assumed that the input is always UTF-8, so we convert the value in `{#1}` to PDF encoding using the `utf8toPDFenc` method from `ltpdfa`. If the entire string contains only the 256 ASCII characters, the string is written as-is into the PDF. However, if it also contains characters further down the Unicode range, the entire string is converted to hexadecimal UTF-16.

```
\DeclareAccessibilityCommand{\ccaAddAltText}[1]{\if@cc@is@final\directlua{ltpdfa.tagger.
    addAltText(ltpdfa.metadata.utf8toPDFenc('\luaescapestring{#1}'))}\fi}
```

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

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

1.8 Hyperlink handling

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

```
\DeclareAccessibilityCommand{\ccaAddLastLink}{\if@cc@is@final\directlua{ltpdfa.tagger.
    addLastLink()}\fi}
```

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

```
\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.

```
\begingroup
\@makeother\#
\ccWhenAlly{%
\AtBeginDocument{%
   \patchcmd\Hy@StartlinkName
     {\pdfstartlink}
     {\Hy@pstringdef\@argii{#2}\ccaStructStart{Link}\ccaAddAltText{#2}\edef\@ltpdfmy@parent{\
         ccaGetStructParent}%
      \pdfstartlink}
     {}{\cca@patch@error\Hy@StartlinkName}
```

and the parent node inside the link attribute:

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

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

```
    \patchcmd\hyper@linkurl
      {\pdfstartlink}
      {\Hy@pstringdef\@argii{#2}\ccaStructStart{Link}\ccaAddAltText{#2}\edef\@ltpdfmy@parent{\
          ccaGetStructParent}%
       \pdfstartlink}
      {}{\cca@patch@error\hyper@linkurl}
```

and secondly for the Parent:

```
    \patchcmd\hyper@linkurl
      {/C[\@urlbordercolor]%
        \fi
      }
      {/C[\@urlbordercolor]%
        \fi
        /StructParent \@ltpdfmy@parent%
        /Contents(\@argii)
      }{}{\cca@patch@error\hyper@linkurl}
```

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

```
    \patchcmd\close@pdflink
      {\pdfendlink}
      {\pdfendlink
       \ccaAddLastLink\ccaStructEnd{Link}}
      {}{\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:

```
    \let\cca@hy@setref\@setref
    \let\@setref\real@setref
    \patchcmd\@setref
      {\else}
      {\else\ccaStructStart{Reference}}
      {}{\cca@patch@error\orig@setref@new}%
    \patchcmd\@setref
      {\fi}
      {\ccaStructEnd{Reference}\fi}
      {}{\cca@patch@error\orig@setref@new}%
```

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

```
    \let\real@setref\@setref
    \let\@setref\cca@hy@setref
    \patchcmd\@setref
      {\expandafter\Hy@setref@link}
      {\ccaStructStart{Reference}\expandafter\Hy@setref@link}
      {}{\cca@patch@error\@setref}
    \patchcmd\@setref
      {{#2}}
      {{#2}\ccaStructEnd{Reference}}
      {}{\cca@patch@error\@setref}
    }% /AtBeginDocument
}% /ccWhenAlly
\endgroup
```

1.9 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.

```
\DeclareAccessibilityCommand{\ccaPagestyleArtifacts}{%
  \ifx\@oddhead\@empty\else
    \pretocmd\@oddhead{\ccaStructStart[Document]{header}}{}{}%
    \apptocmd\@oddhead{\ccaStructEnd{header}}{}{}%
  \fi
  \ifx\@evenhead\@empty\else
    \pretocmd\@evenhead{\ccaStructStart[Document]{header}}{}{}%
    \apptocmd\@evenhead{\ccaStructEnd{header}}{}{}%
  \fi
  \ifx\@oddfoot\@empty\else
    \pretocmd\@oddfoot{\ccaStructStart[Document]{footer}}{}{}%
    \apptocmd\@oddfoot{\ccaStructEnd{footer}}{}{}%
  \fi
  \ifx\@evenfoot\@empty\else
    \pretocmd\@evenfoot{\ccaStructStart[Document]{footer}}{}{}%
    \apptocmd\@evenfoot{\ccaStructEnd{footer}}{}{}%
  \fi}
```

The standard pagestyles from the LaTeX kernel are patched by the module.

```
\AtBeginDocument{%
  \apptocmd\ps@empty{\ccaPagestyleArtifacts}{}{}%
  \apptocmd\ps@plain{\ccaPagestyleArtifacts}{}{}%
  \apptocmd\ps@headings{\ccaPagestyleArtifacts}{}{}%
  \apptocmd\ps@myheadings{\ccaPagestyleArtifacts}{}{}%
}
```

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

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

1.10 generic artifacts

```
  \ccaAddToConfig{artifact}{leaders={Type:Layout}}
  \ccaAddToConfig{artifact}{footnoterule={Type:Layout}}
  \ccaAddToConfig{artifact}{Rule={Type:Layout}}
  \ccaAddToConfig{artifact}{Artifact={Type:Layout}}
  \ccaAddToConfig{artifact}{Background={Type:Background}}
}
```

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

```
\def\ccaArtifact{\ccaStructStart[Document]{Artifact}\ccaDisable}
```

`\endccaArtifact` ends an Artifact environment.

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

1.11 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.

```
\DeclareAccessibilityCommand{\ccaAddFigure}[7]{\directlua{ltpdfa.tagger.addFigure(
    '\luaescapestring{#1}',
    '\luaescapestring{#2}',
    '\luaescapestring{#3}',
    '\luaescapestring{#4}',
    '\luaescapestring{#5}',
    '\luaescapestring{#6}',
    '\luaescapestring{#7}')}}
```

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

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

`\ccaFigureEnd` injects the ending tag for images

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

which we add to the beginning and the end of `graphics` package’s `\Ginclude@graphics` macro, respectively:

```
\AtBeginDocument{%
  \let\ltx@Ginclude@graphics\Ginclude@graphics
  \def\Ginclude@graphics#1{\if@cc@is@final\ccaFigureStart{}\fi\ltx@Ginclude@graphics{#1}\
      if@cc@is@final\ccaFigureEnd{}\fi}%
}
```

```
\apptocmd\Ginclude@@pdftex{\if@cc@is@final%
  \def\@tempa{!}%
  \ccaAddFigure{\Gin@llx}{\Gin@lly}{\Gin@urx}{\Gin@ury}
    {\ifx\Gin@scalex\@tempa\else \Gin@scalex\fi}
    {\ifx\Gin@scaley\@tempa\else \Gin@scaley\fi}
    {\ifGin@clip true\else false\fi}\fi}%rwi/rhi
    {}{}
\AtBeginDocument{%
  \@ifpackageloaded{htmltabs}{%
    \let\ltx@ht@valign@box\ht@valign@box
    \def\ht@valign@box{\if@ht@final@render\@cc@is@finaltrue\fi\ltx@ht@valign@box}
    \let\ltx@ht@RenderCell\ht@RenderCell
    \def\ltx@ht@RenderCell{\@cc@is@finalfalse\ltx@ht@RenderCell}}{}}
```

Tagging for Tables

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

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

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

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

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

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

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

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

1.12 Transformation of Typographic Unicode characters

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

```
\ifx\pdfextension\@undefined\else
\protected\def\pdfglyphtounicode{\pdfextension glyphtounicode}
\input glyphtounicode
\edef\pdfgentounicode{\pdfvariable gentounicode}
\pdfgentounicode = 1
\fi
```

\ccaConvertPdfString takes a utf-16 string as {#1} and converts it to utf-8. This is intended to resolve octal tokens in the output of hyperref's \pdfstringdef, which ltpdfa's ltpdfa.setDocInfo() method does not seem to handle well.

```
\DeclareAccessibilityCommand{\ccaConvertPdfString}[1]{\directlua{tex.print(ltpdfa.metadata.utf16
    ToUtf8('\luaescapestring{#1}'))}}
```

1.13 Automatic PDF Tagging

Document Root Node

The following code causes the ltpdfa package to tag the *document* environmant as the structural representation's root node:

```
\ccWhenAlly{%
```

cca/document/begin is a hook to add accessibility specific declarations right before begin document.

```
  \NewHook{cca/document/begin}%
  \AtBeginDocument{%
    \directlua{ltpdfa.beginDocument('\luaescapestring{\ltpdfa@last@page}')}
    \UseHook{cca/document/begin}%
    \ccaVstructStart{Document}%
```

`\cca@id@document` is the internal ID of the document root node.

```
    \ccaSaveCurStruct{cca@id@document}%
```

`\cca@id@document@dummy` is the internal ID of a placeholder tag node inserted immediately after the root tag node is opened. This can be used to be replaced by the document's main title tag node using the `\ccaReplaceStruct` command, which should get this macro as its sole argument.

```
    \ccaStructStart{dummy}%
      \ccaSaveCurStruct{cca@id@document@dummy}%
    \ccaStructEnd{dummy}%
```

`\PDF@FinishDoc` is re-defined to disable `hyperref` filling the PDF's DocumentInfo with empty strings, as they are written by `ltpdfa`, instead.

```
    \renewcommand\PDF@FinishDoc{%
      \Hy@DisableOption{pdfauthor}%
      \Hy@DisableOption{pdftitle}%
      \Hy@DisableOption{pdfsubject}%
      \Hy@DisableOption{pdfcreator}%
      \Hy@DisableOption{pdfcreationdate}%
      \Hy@DisableOption{pdfmoddate}%
      \Hy@DisableOption{pdfproducer}%
      \Hy@DisableOption{pdfkeywords}%
      \Hy@DisableOption{pdftrapped}%
      \Hy@DisableOption{pdfinfo}%
    }%
  }%
  \AtEndDocument{%
    \ccDebugMsg[a11y]{Final AutoClose. \meaning\ccPrevSecLevel, \meaning\ccCurSecLevel}%
    \cchResetNesting
    \ccDebugMsg[a11y]{End Document}%
    \if@backmatter\ccaVstructEnd{Backmatter}\fi
    \ccaVstructEnd{Document}
    \ifarticle\else
      \ccaAddRolemap{Frontmatter}{Part}%
      \ccaAddRolemap{Mainmatter}{Part}%
      \ccaAddRolemap{Backmatter}{Part}%
    \fi
    \directlua{ltpdfa.endDocument()}%
  }%
}
```

1.14 Math Tagging

```
\AtBeginDocument{%
  \apptocmd{\(}{\ccaStructStart{Formula}}{}{\cca@patch@error\(}
  \pretocmd{\)}{\ccaStructEnd{Formula}}{}{\cca@patch@error\)}
  \apptocmd{\[}{\ccaVstructStart{P}\ccaStructStart{Formula}}{}{\cca@patch@error\[}
  \pretocmd{\]}{\ccaStructEnd{Formula}\ccaVstructEnd{P}}{}{\cca@patch@error\]}
  \@ifpackageloaded{amsmath}{%
    \pretocmd{\start@align}{\ccaVstructStart{P}\ccaStructStart{Formula}}{}{\cca@patch@error\
        start@align}%
    \apptocmd{\endalign}{\ccaStructEnd{Formula}\ccaVstructEnd{P}}{}{\cca@patch@error\endalign}%
  }{}%
}
```

1.15 Default Role Mapping

Note that this section contains only the role mappings that didn't thematically fit into other CoCoTEX modules.

```
% none so far.
```

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

```
\ccWhenAlly{\AddToHook{shipout/before}{\directlua{ltpdfa.pageprocessor(tex.box["ShipoutBox"])}}}
    %
```

End of TEX source code.

```
</a11y-sty>
```

2 Lua code

```
<*a11y-lua>
```

2.1 Local Variables and Tables

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

```
local ltpdfa = require('ltpdfa')
```

2.2 Meta Data Extraction

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

```
meta = {
  Author = '',
  Title = '',
  Creator = '',
  Producer = '',
  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).

```
  extract = function(self)
    local xmpfile = ltpdfa.metadata.xmphandler.fromFile(ltpdfa.config.metadata.xmpfile)
    local f = io.open(xmpfile, "rb")
    local content = f:read("*all")
    f:close()
```

First, we extract the document title.

```
    if (content:find('<dc:title>')) then
      local title = content:gsub('.*<dc:title>[^<]*<rdf:Alt>[^<]*<rdf:li[^>]*>(.*)</rdf:li
          >[^<]*</rdf:Alt>[^<]*</dc:title>.*', "%1")
      self.Title = title
    end
```

Then, we extract the authors from the `dc:creator` list.

```
    local authors
    local author = {}
    if (content:find('<dc:creator>')) then
      authors = content:gsub('.*<dc:creator>[^<]*<rdf:Seq>(.*)</rdf:Seq>[^<]*</dc:creator>.*', "
          %1")
      for k in string.gmatch(authors, "<rdf:li>([^>]+)</rdf:li>") do
        table.insert(author , k)
      end
      self.Author = table.concat(author, '\\and ')
    end
```

Then, we extract the keywords from the `dc:subject` list. If that doesn't exist, we try to extract the keywords from the `pdf:Keywords` Element, instead.

```
    local keywords
    local keyword = {}
    if (content:find('dc:subject')) then
      keywords = content:gsub('.*<dc:subject>[^<]*<rdf:Bag>(.*)</rdf:Bag>[^<]*</dc:subject>.*', "
          %1")
      for k in string.gmatch(keywords, "<rdf:li>([^>]+)</rdf:li>") do
        table.insert(keyword , k)
      end
      self.Keywords = table.concat(keyword, '\\and ')
    elseif (content:find('pdf:Keywords')) then
      local keyword = content:gsub('.*<pdf:keywords>(.*)</pdf:Keywords>.*', "%1")
      self.Keywords = keyword
    end
```

Then, we extract the PDF producer fom the `pdf:Producer` element, if it exists:

```
    if (content:find('<pdf:Producer>')) then
      local prod = content:gsub('.*<pdf:Producer>(.*)</pdf:Producer>.*', "%1")
      self.Producer = prod
    end
```

Finally, we extract the PDF CreatorTool fom the `xmp:CreatorTool` element, if it exists:

```
    if (content:find('<xmp:CreatorTool>')) then
      local creatortool = content:gsub('.*<xmp:CreatorTool>(.*)</xmp:CreatorTool>.*', "%1")
      self.Creator = creatortool
    end
  end
}
```

2.3 Public Methods

`cocotex` is the base table that contains all public methods and sub-tables available in the CoCoTEX framework. Here, it is defined unless it is already defined elsewhere.

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

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

```
cocotex.ally = {
  meta = meta
}
```

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

```
return cocotex.ally
```

no more lua code.

```
</a11y-lua>
```

Module 5 coco-meta.dtx

```
<*meta>
```

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

File preamble

```
%%
%% module for CoCoTeX that provides handling of a document's meta data.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-meta}
   [2024/12/13 v0.5.0 CoCoTeX meta module]
\RequirePackage{coco-common}
```

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

```
\ccDeclareContainer{CommonMeta}{%
  \ccDeclareType{Components}{%
    \ccDeclareRole[author]{Author}%
    \ccm@declare@comp
    \ccm@extended@common@macros
    \ccm@declare@affils
  }%
  \ccDeclareType{Properties}{}%
}
```

1 Counted Container Handlers

1.1 Generic Blocks

\ccm@generic@comp is used to define a generic meta data block. It provides two Components for each instance, one for the block's Heading or Label, and one for its Content.

```
\def\ccm@generic@comp{%
```

CC is the formatted list of all GenericMeta components.

```
  \ccDeclareComponent{GenericMetaBlock}{\expandafter\global}{}%
```

```
  \ccDeclareComponentGroup{GenericMeta}{%
```

`Heading` is the label of a generic meta datum

```
    \ccDeclareCountedComponent{Heading}%
```

`Heading` is the content of a generic meta datum

```
    \ccDeclareCountedComponent{Content}%
```

```
  }}
```

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

```
\def\ccm@generic@eval{{%
  \def\cc@cur@cont{titlepage}%
  \ccComposeCollection{GenericMeta}{generic-meta-format}{GenericMetaBlock}
}}
```

1.2 Contributor Roles

Contributors are Counted Containers that represent the meta data of people that share a role in contributing content to a document. Examples for such roles are an article/chapter/book's authors, or a collection/series' editors.

1.3 Declaring Contributor Role Blocks

`\ccDeclareRoleBlock` is used to create a new Collection Container (named `\ccPrefix#2#3`) for a Role with the name `{#2}`. A Role Block is a Component of the Parent Container, which contains a formatted list of certain Components of all members of the Role. Format and selection of the utilised Components are specified via the Property given in `{#4}`. Role Blocks can also be directly used inside the Parent Container as Overrides.

The optional argument `[#1]` tells the evaluator in the Parent Container's `end` macro how the collector is to be composed. Valid values are `compose` (*default*), which uses `\ccComposeCollection` to compose the Collection Component, or `apply`, which uses `\ccApplyCollection`, instead.

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

`\ccm@role@eval` creates the name lists for the role. `{#1}` is the name of the role.

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

`\ccm@eval@{#1}e` is the name of the macro used to compose the Collection (either `\ccComposeCollection`, or `\ccApplyCollection`)
`{#2}` is the name of the role
`{#3}` is the name of the list.

The access Component is `\ccPrefix #2#3`, i.e., the prefix and both argumets together.

```
\def\ccm@eval@role#1#2#3{%
```

First, we check if the Collection Component has already been set in the input. If so, we set an internal flag to indicate that the Collection Component has been filled manually.

```
  \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.

```
    \ifcsdef{cc#2Cnt}
      {\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`.

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

`\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.

```
\def\ccm@role@apply#1#2{\ccm@eval@role\ccApplyCollection{#1}{#2}}
```

`\ccm@role@compose` #1 is the name of the role and #2 is the name of the composition. This stores the *unexpaded* contents of the `\ccm@role@\cc@cur@cont @#1@#2` Property in the `#1#2` Component.

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

Declaring Contributor Roles

`\ccDeclareRole` is used to declare the Components that belong to each member of a contributor role.

`[#1]` is the internal name of the Role's formatting Property. If omitted, it is the same as `{#2}`
`{#2}` is the name of the role

The output of all members of a role is controlled by a Component called "`<role>NameList`" that is formatted according to the `<role>-format` Property. For reasons of naming conventions, the role names for a Component and its respective Property do not necessarily need to be identical.

```
\def\ccDeclareRole{\cc@opt@second\cc@declare@role}%
\def\cc@declare@role[#1]#2{%
  \ccDeclareComponentGroup[%
    {\ccDeclareAttributeHandler*{corresp}{\ccSetProperty{is-corresp}{true}}}%
  ]{#2}{%
```

Contributor Role Components

`FullNameOR` is the full name of the Role member. If omitted, it is calculated by the `role-full-name-format` Property.

```
    \ccDeclareCountedComponent{FullName}%
```

`CiteNameOR` is the Full Role member name that is used for citation advices. If omitted, it is calculated by the `role-cite-name-format` Property.

```
    \ccDeclareCountedComponent{CiteName}%
```

ShortCiteNameOR is a shortened version of the CiteName Component. If omitted, it is calculated by the role-short-cite-name-format Property.

```
    \ccDeclareCountedComponent{ShortCiteName}%
```

PDFInfoNameOR is the version of the Role member name that is used in the PDF Info dictionary. If omitted, it is calculated by the role-pdfinfo-name-format Property.

```
    \ccDeclareCountedComponent{PDFInfoName}%
```

InitialOR holds the initials of the Role member. If omitted, the initials are attempted to be calculated from the FirstName and MidName Components via the initials-format Property.

```
    \ccDeclareCountedComponent{Initial}%
```

LastName is the surname of the Role member.

```
    \ccDeclareCountedComponent{LastName}%
```

FirstName is the first name of the Role member.

```
    \ccDeclareCountedComponent{FirstName}%
```

MidName is/are the middle name(s) of the Role member.

```
    \ccDeclareCountedComponent{MidName}%
```

Honorific is a other honorific title for the Role member.

```
    \ccDeclareCountedComponent{Honorific}%
```

Lineage is the name suffix, typically something like “jr.” or “the 3rd”.

```
    \ccDeclareCountedComponent{Lineage}%
```

ORCID is the ORCID (Open Researcher and Contributor ID) of the Role member. Depending on the publisher style, this can be a full URL or just the identifier.

```
    \ccDeclareCountedComponent{ORCID}%
```

AffilRef is the ID of an entry in the Affil Component Group.

```
    \ccDeclareCountedComponent{AffilRef}% for references to the Affil Group
```

Affiliation is the Affiliation of the Role member.

Note that only one AffilRef or Affiliation should be used for any Role member, not both at the same time.

```
    \ccDeclareCountedComponent{Affiliation}% for affiliations as direct Author meta data
```

Email is the email/contact address of the Role member.

```
    \ccDeclareCountedComponent{Email}%
```

CorrespondenceAsOR is how the Role member is to be addressed when she is the corresponding Role member. If omitted, it is calculated by the role-correspondence-as-format Property.

```
  \ccDeclareCountedComponent{CorrespondenceAs}%
```

Contributor Role Group Handlers

The Group Handlers fill the previously defined Override Components when they are not explicitly given inside the Component Group.

```
}%
\ccDeclareGroupHandler{#2}{%
  \ccUnlessComp{FullName}{\ccComponent{FullName}{\ccUseProperty{#1-full-name-format}}}%
  \ccUnlessComp{Initial}{\ccComponent{Initial}{\ccUseProperty{initials-format}}}%
  \ccUnlessComp{CiteName}{\ccComponent{CiteName}{\ccUseProperty{#1-cite-name-format}}}%
  \ccUnlessComp{ShortCiteName}{\ccComponent{ShortCiteName}{\ccUseProperty{#1-short-cite-name-
      format}}}%
  \ccUnlessComp{PDFInfoName}{\ccComponent{PDFInfoName}{\ccUseProperty{#1-pdfinfo-name-format
      }}}%
  \ccUnlessComp{CorrespondenceAs}{\ccComponent{CorrespondenceAs}{\ccUseProperty{#1-
      correspondence-as-format}}}%
  \ccWhenComp{AffilRef}{\ccWhenComp{Affiliation}{%
      \ccPackageError{Meta}{Ambiguity}
        {You cannot use both Containers AffilRef and Affiliation in the same `\ccPrefix#2' Sub-
            Container}
        {At least one `\ccPrefix#2' Sub-Container contains both AffilRef and Affiliation. This
            is not allowed. Please decide for one affiliation strategy: Either two lists with
            cross-references, or affiliations directly as an author's meta-data.}}}%
}%
```

Declaring the Contributor Role's Collection Components

Recall that the Collection Component's name are all prefixed by the Role's name, e.g., the actual Collection Component NameList of a Role named "Author" is accessed by calling the AuthorNameList Component.

NameListCL is the formatted list of all Role member's FullName Components according to the role-block-print-format Property.

```
\ccDeclareRoleBlock{#2}{NameList}{#1-list-print-format}%
```

CitationListCL is the formatted list of all Role member's CiteName according to the role-block-cite-format Property.

```
\ccDeclareRoleBlock{#2}{CitationList}{#1-list-cite-format}%
```

ShortCitationListCL is the formatted list of all Role member's ShortCiteName Component role-block-short-cite-format Property.

```
\ccDeclareRoleBlock{#2}{ShortCitationList}{#1-list-short-cite-format}%
```

PDFInfoCL is the formatted string that is sent to the PDF's Info dictionary. Its format is determined by the role-block-pdfinfo-format Property.

```
\ccDeclareRoleBlock[apply]{#2}{PDFInfo}{#1-list-pdfinfo-format}%
```

`CorrespondenceCL` is the list of all Role member's `CorrespondanceAs` Components according to the `role-block-correspondence-format` Property.

```
  \ccDeclareRoleBlock{#2}{Correspondence}{#1-list-correspondence-format}%
```

```
}
```

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

TODO
make into LaTeX kernel hook

```
\def\ccAddToRole#1#2{\csgappto{cc@group@#1@hook}{#2}}
```

2 Labeled Components

Labeled Components are two Components, one for the Content and one for the Label.

`\ccDeclareLabeledComp` declares two Components: one named `\ccPrefix #2` for the value, and another one named `\ccPrefix #2Label` for its corresponding label. #3 is used for Property overrides. The optional Argument #1 allows to set a default value for the Label.

```
\def\ccDeclareLabeledComp{\cc@opt@empty\cc@declare@labeled@comp}
\def\cc@declare@labeled@comp[#1]#2#3{%
  \ccDeclareComponent{#2}{\expandafter\global}{}%
  \ccDeclareComponent{#2Label}{\expandafter\global}{}%
  \csxdef{labeled-meta-property-infix-\cc@cur@cont-#2}{#3}%
  \if!#1!\else
    \long\csgdef{cc@\cc@cur@cont @#2Label}{#1}%
  \fi\ignorespaces}
```

`\ccUseLabeledComp` returns the Labeled Component with its label. The starred version omits automatic Tagging if the `coco-accessibility` module is active.

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

`\ccCurInfix` stores the currently active Property infix for the Labeled Component. Is is used to call the right format Property for the Labeled Component, which defaults to `labeled-meta-[ccCurInfix]-format`. If this Property doesn't exists, formatting falls back to `labeled-meta-format`.

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

`\ccCurComp` stores the currently active name of the Labeled Component, which is used in the generic `labeled-meta-format` Property.

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

```
    \ifx\ccm@no@tag\relax\else
      \ccaStructStart{MetaDatum}%
      \ccaAddToStruct{\cch@id@cur@meta}%
    \fi
    \ccIfProp{labeled-meta-\ccCurInfix-format}
      {\ccUseProperty{labeled-meta-\ccCurInfix-format}}
      {\ccUseProperty{labeled-meta-format}}%
    \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatum}\fi
  }\global\let\ccm@no@tag\@undefined}
```

3 Meta Data Rolemaps for Tagged PDFs

Role mapping for accessibility tagging:

```
\ccaAddRolemap{Authors}{P}
\ccaAddRolemap{Editors}{P}
\ccaAddRolemap{SeriesEditors}{P}
\ccaAddRolemap{Affiliations}{P}
\ccaAddRolemap{MetaDatum}{Div}
\ccaAddRolemap{MetaDatumLabel}{P}
\ccaAddRolemap{MetaDatumValue}{P}
\ccaAddRolemap{MetaDatumBlock}{Div}
\ccaAddRolemap{Abstract}{Div}
\ccaAddRolemap{AbstractLabel}{P}
\ccaAddRolemap{AbstractText}{Div}
\ccaAddRolemap{Keywords}{Div}
\ccaAddRolemap{KeywordsLabel}{P}
\ccaAddRolemap{KeywordsText}{P}
```

4 Common Meta Data

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

```
\def\ccm@declare@comp{%
```

`Copyright` holds the Copyright notice.

```
    \ccDeclareComponent{Copyright}{\expandafter\global}{}% Copyright text
```

`LicenceLogo` is a component for a license logo. This usually contains an `\includegraphics`.

```
    \ccDeclareComponent{LicenceLogo}{}{}%
```

`LicenceName` is the name of the license.

```
    \ccDeclareComponent{LicenceName}{}{}%
```

```
}%
```

`article-meta` is an abstract Container that holds meta data specific to a journal's Article.

```
  %% for single articles
\ccDeclareContainer{article-meta}{%
  \ccDeclareType{Components}{%
```

`StartPage` is the number of the starting page of an article

```
    \ccDeclareGlobalComponent{StartPage}
```

`EndPage` is the number of the ending page of an article

```
    \ccDeclareGlobalComponent{EndPage}
```

`CiteAs` holds a string as to how the article should be cited in other publications.

```
    \ccDeclareLabeledComp[Cite as]{CiteAs}{cite-as}
```

`Submitted` holds the date when the article was submitted to the journal.

```
    \ccDeclareLabeledComp[Submitted]{Submitted}{sumbitted}
```

`Received` holds the date when the article was received by the journal

```
    \ccDeclareLabeledComp[Received]{Received}{received}
```

`Revised` holds the date when the article was revised by its author(s)

```
    \ccDeclareLabeledComp[Revised]{Revised}{revised}
```

`Reviewed` holds the date when the article was reviewed by the editors or reviewers.

```
    \ccDeclareLabeledComp[Reviewed]{Reviewed}{reviewed}
```

`Accepted` holds the date when the article was accepted for publication by the journal.

```
    \ccDeclareLabeledComp[Accepted]{Accepted}{accepted}
```

`Published` holds the date when the article is due to be published.

```
    \ccDeclareLabeledComp[Published]{Published}{published}
```

`COIStatement` holds the author's Conflict of Interest statement

```
    \ccDeclareLabeledComp[Conflict of Interest]{COIStatement}{coi-statement}
```

```
  }}
```

`\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.

```
\def\ccm@extended@common@macros{%
```

`Abstract` holds the contribution's abstract or content summary.

```
  \ccDeclareLabeledComp[Abstract]{Abstract}{abstract}%
```

`Keywords` holds a list of keywords related to the contribution.

```
  \ccDeclareLabeledComp[Keywords]{Keywords}{keyword}%
```

DOI holds the Digital Object Identifier. Depending on the Publisher style, this may be the full URI, or just the identifier.

```
  \ccDeclareLabeledComp{DOI}{doi}%
```

TitleEn holds the English title of the publication when the contribution's main language is *not* english.

```
  \ccDeclareLabeledComp{TitleEn}{title-en}%
```

```
  \ccm@generic@comp
}
```

4.1 Affiliations

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

```
\def\ccm@declare@affils{%
```

AffilBlockCC is the Collection Component for the contribition's Affiliations list. **Note** that the AffilBlock itself is not generated in this module. The two modules coco-headings and coco-title that both depend on the coco-meta module have their own mechanisms to build their respective AffilBlock Collection Components.

```
  \ccDeclareComponent{AffilBlock}{\expandafter\global}{}%
```

```
  \ccDeclareComponentGroup{Affil}{%
```

AffiliationOR is the fully formatted Affiliation string. If omitted, the Component is built using the affiliation-format Property.

```
    \ccDeclareCountedComponent{Affiliation}%
```

Address is the address where the Role member is working.

```
    \ccDeclareCountedComponent{Address}%
```

Institute is the name of the university, department or institution where the Role member is working

```
    \ccDeclareCountedComponent{Institute}%
```

Country is the country where the institution is locaed in.

```
    \ccDeclareCountedComponent{Country}%
```

Department is the department where the Role member is working.

```
    \ccDeclareCountedComponent{Department}%
```

AffilIDOR is the internal identifier that is referenced by the Role member's AffilRef Component. If omitted, the ID is the value of an automatic counter that is incremented by one at the beginning of each Affil Group Container counter in the same Parent Container.

```
    \ccDeclareCountedComponent{AffilID}%
```

```
  }%
  \ccDeclareGroupHandler{Affil}{%
    \ccUnlessComp{AffilID}{\ccComponentEA{AffilID}{\ccAffilCnt}}%
    \ccUnlessComp{Affiliation}{\ccComponent{Affiliation}{\ccUseProperty{affiliation-format}}}%
  }%
}
```

5 Meta Data Properties

```
\ccAddToType{Properties}{CommonMeta}{%
```

5.1 Initials

`initials-format` `<any>` generates an Role member's initials from the FirstName and MidName Components.

```
  \ccSetProperty{initials-format}{%
    \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
        cc@long@empty\else
      \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\the\ccCurCount\endcsname\
          relax\else
        \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-FirstName-\
            the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
      \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
          cc@long@empty\else
        \expandafter\ifx\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\the\ccCurCount\endcsname\
            relax\else
          \ccUseProperty{initials-sep}%
          \expandafter\expandafter\expandafter\@car\csname cc@\cc@cur@cont @\cc@cnt@grp-MidName-\
              the\ccCurCount\endcsname\relax\@nil\ccUseProperty{initials-period}%
        \fi\fi
    \fi\fi
  }
```

`initials-sep` `<any>` is the separator between two Initials.

```
  \ccSetProperty{initials-sep}{~}
```

`initials-period` `<any>` is the symbol that is inserted at the end of each Initial.

```
  \ccSetProperty{initials-period}{.}
```

5.2 Member Role Composition Properties

5.3 Overrides Within a Role Counted Component

The next Properties control how the Compoent Overrides within a single Role Counted Component are composed.

`role-full-name-format` `<any>` how the FullName Component for each Role member is generated.

```
  \ccSetProperty{role-full-name-format}{%
    \if\ccUseComp{Honorific}\relax
    \else
     \ccUseComp{Honorific}\space
    \fi
    \ccUseComp{FirstName}\space
    \if\ccUseComp{MidName}\relax
    \else
     \ccUseComp{MidName}\space
    \fi
    \ccUseComp{LastName}%
    \if\ccUseComp{Lineage}\relax
    \else
     \space\ccUseComp{Lineage}%
    \fi%
  }%
```

role-cite-name-format <any> how the CiteName for each Role member is formatted.

```
\ccSetProperty{role-cite-name-format}{\ccIfComp{LastName}{\ccUseComp{LastName},~\ccUseComp{
    Initial}}{\ccUseComp{FullName}}}% How CiteName for each name is built
```

role-short-cite-name-format <any> how the ShortCiteName Component of a Role member is formatted

```
\ccSetProperty{role-short-cite-name-format}{\ccUseComp{LastName}}%
```

role-pdfinfo-name-format <any> how the PDFInfoName of a Role member is formatted

```
\ccPropertyLet{role-pdfinfo-name-format}{role-cite-name-format}%
```

role-correspondence-as-format <any> How the CorrespondenceAs string of a Role Member is formatted

```
\ccSetProperty{role-correspondence-as-format}{\ccUseComp{Email}}%
```

5.4 Format of Single Role Collection Component Items

the next properties control how single items in the Parent container's Collection Components are to be formatted.

role-block-print-format <any> How a single entry in the NameList is formatted.

```
\ccSetProperty{role-block-print-format}{\ccUseComp{FullName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}%
```

role-block-cite-format <any> how a single entry in the CitationList is formatted

```
\ccSetProperty{role-block-cite-format}{\ccUseComp{CiteName}\ifnum\ccCurCount<\ccTotalCount\
    ccUseProperty{counted-name-sep}\fi}%
```

role-block-short-cite-format <any> how a single entry in the ShortCitationList is formatted

```
\ccSetProperty{role-block-short-cite-format}{\ccUseComp{ShortCiteName}\ifnum\ccCurCount<\
    ccTotalCount\ccUseProperty{counted-name-sep}\fi}%
```

role-block-pdfinfo-format <any> how a single entry in the PDFInfo Component is formatted

```
\ccSetProperty{role-block-pdfinfo-format}{\ccUseComp{PDFInfoName}\ifnum\ccCurCount<\
    ccTotalCount\and\fi}% How each item in the Component <Role>PDFInfo is formatted
```

`role-block-correspondence-format` <any> how a single entry in the `Correspondence` Component is formatted

```
\ccSetProperty{role-block-correspondence-format}{%
  \ccIfPropVal{is-corresp}{true}
    {\ifx\is@first@corresp\relax
       \ccUseProperty{corresp-sep}%
     \else
       \global\let\is@first@corresp\relax
     \fi
     \ccUseComp{CorrespondenceAs}%
   }{}}%
```

`counted-name-sep` <any> how single entries in `NameList` are separated

```
\ccSetProperty{counted-name-sep}{,\space}%
```

`name-and` <any> is a Property that can be used when composing Role specific Collection Components. Is is usually used between the penultimate and the last entry in the Collection Component.

```
\ccSetProperty{name-and}{\space and\space}%
```

`name-etal` <any> is a Property that can be used when composing Role specific Collection Components. Is is usually used after the first entry in the list, when the total number of entries is too large.

```
\ccSetProperty{name-etal}{\space et~al.}%
```

`name-sep` <any> is the default separator between entries in a Role specific Collection Component.

```
\ccSetProperty{name-sep}{,\space}%
```

`keywords-sep` <any> is the default separator between Entries in the Keywords list.

```
\ccSetProperty{keywords-sep}{,\space}%
```

`corresp-mark` <any> is the default marker for the “Correspondence” Role member, i.e., the Role member who is designated the primary contact person of a contribution.

```
\ccSetProperty{corresp-mark}{*}%
```

`corresp-sep` <any> is the default seperator between entries in the `Correspondence` Collection Component.

```
\ccSetProperty{corresp-sep}{,\space}%
```

5.5 Collection Component Properties Specific to Author Role

The Properties defined here are mostly aliases of the more generic Properties defined above.

Author Instance Override Properties

`author-cite-name-format` <any> how an Author’s `CiteName` is formatted.

```
\ccPropertyLet{author-cite-name-format} {role-cite-name-format}%
```

`author-short-cite-name-format` <any> how an author’s `ShortCiteName` is formatted.

```
\ccPropertyLet{author-short-cite-name-format} {role-short-cite-name-format}%
```

author-full-name-format `<any>` how an author's FullName Component is composed

```
\ccPropertyLet{author-full-name-format} {role-full-name-format}%
```

author-pdfinfo-name-format `<any>` how an author's PDFInfoName Component is composed

```
\ccPropertyLet{author-pdfinfo-name-format} {role-pdfinfo-name-format}%
```

author-correspondence-as-format `<any>` how an author's CorrespondenceAs entry is to be formatted

```
\ccPropertyLet{author-correspondence-as-format} {role-correspondence-as-format}%
```

Author-Specific Collcection Component Override Properties

author-list-print-format `<any>` is the format of each entry in the AuthorNameList Component.

```
\ccPropertyLet{author-list-print-format} {role-block-print-format}%
```

author-list-cite-format `<any>` is the format of each entry in the AuthorCitationList Component.

```
\ccPropertyLet{author-list-cite-format} {role-block-cite-format}%
```

author-list-short-cite-format `<any>` is the format of each entry in the AuthorShortCitationList Component.

```
\ccPropertyLet{author-list-short-cite-format} {role-block-short-cite-format}%
```

author-list-pdfinfo-format `<any>` is the format of each entry in the AuthorPDFInfo Component.

```
\ccPropertyLet{author-list-pdfinfo-format} {role-block-pdfinfo-format}%
```

author-list-correspondence-format `<any>` is the format of the AuthorContribution Collection Component.

```
\ccPropertyLet{author-list-correspondence-format} {role-block-correspondence-format}%
```

5.6 Format of Affiliation Lists

affiliation-format `<any>` is the format of the Affiliation Component for each Affil Instance.

```
\ccSetProperty{affiliation-format}{%
  \ccWhenComp{Institute}{\ccUseComp{Institute}}%
  \ccWhenComp{Department}{, \ccUseComp{Department}}%
  \ccWhenComp{Address}{, \ccUseComp{Address}}%
}%
```

affil-sep `<any>` is the separator between the entries of the AffilBlock Collection Component.

```
\ccSetProperty{affil-sep}{\par}
```

affil-block-item-face `<any>` are the font parameters used to print each entry in the AffilBlock Collection Component.

```
\ccSetProperty{affil-block-item-face}{}%
```

affil-block-item-format i s the format of each entry in the AffilBlock list

```
\ccSetProperty{affil-block-item-format}{%
  \textsuperscript{\ccUseComp{AffilID}}%
  \bgroup
    \ccUseProperty{affil-block-item-face}%
    \ccUseComp{Affiliation}
  \egroup%
  \ifnum\ccCurCount<\ccTotalCount\relax\ccUseProperty{affil-sep}\fi%
}
```

affil-block-face <any> is the font used to print the AffilBlock Collection Component.

```
\ccSetProperty{affil-block-face}{\small\normalfont}%
```

affil-block-format <any> prints the AffilBlock Collection Component.

```
\ccSetProperty{affil-block-format}{%
  \ccWhenComp{AffilBlock}
    {\bgroup
       \ccUseProperty{affil-block-face}%
       \ccUseComp{AffilBlock}%
     \egroup
     \par
   }}
```

5.7 Properties for Labeled Componetns

labeled-meta-format <any> is the generic Property that determins how Labeled Components are composed. It checks for implicit formatting properties speific to each labeled Component and falls back to generic defaults if those are not defined by the user or publisher style.

```
\ccSetProperty{labeled-meta-format}{%
  \ccIfProp{labeled-meta-before-\ccCurInfix}
    {\ccUseProperty{labeled-meta-before-\ccCurInfix}}
    {\ccUseProperty{labeled-meta-before}}%
  \bgroup
    \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumLabel}\fi
    \ccIfProp{labeled-meta-\ccCurInfix-face}
      {\ccUseProperty{labeled-meta-\ccCurInfix-face}}
      {\ccUseProperty{labeled-meta-face}}%
    \ccIfProp{labeled-meta-\ccCurInfix-label-format}
      {\ccUseProperty{labeled-meta-\ccCurInfix-label-format}}
      {\ccUseProperty{labeled-meta-label-format}}%
    \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumLabel}\fi
    \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatumValue}\fi
    \ccUseComp{\ccCurComp}%
    \ifx\ccm@no@tag\relax\else\ccaStructEnd{MetaDatumValue}\fi
  \egroup
  \ccIfProp{labeled-meta-after-\ccCurInfix}
    {\ccUseProperty{labeled-meta-after-\ccCurInfix}}
    {\ccUseProperty{labeled-meta-after}}%
}
```

labeled-meta-label-format <any> is the generic format of the label of a Labeled Component.

```
\ccSetProperty{labeled-meta-label-format}{%
  \ccWhenComp{\ccCurComp Label}{%
    \bgroup
```

```
      \ccUseProperty{labeled-meta-before-\ccCurInfix-label}%
      \ccIfProp{labeled-meta-\ccCurInfix-label-face}
        {\ccUseProperty{labeled-meta-\ccCurInfix-label-face}}
        {\ccUseProperty{labeled-meta-label-face}}%
      \ccUseComp{\ccCurComp Label}%
      \ccIfProp{labeled-meta-\ccCurInfix-label-sep}
        {\ccUseProperty{labeled-meta-\ccCurInfix-label-sep}}
        {\ccUseProperty{labeled-meta-label-sep}}%
    \egroup
  }}
```

`labeled-meta-label-face` `<any>` is the font setting for the Label of a Labeled Component.

```
  \ccSetProperty{labeled-meta-label-face}{\bfseries}
```

`labeled-meta-label-sep` `<any>` is the default and fallback separator between the Labeled Component's `Label` and its value.

```
  \ccSetProperty{labeled-meta-label-sep}{:\enskip}
```

`labeled-meta-face` `<any>` is the face of a Labeled Component. It applies to bothe the Label and the Value, but can be locally overridden by the `labeled-meta-label-face` Property.

```
  \ccSetProperty{labeled-meta-face}{}
```

`labeled-meta-before` `<any>` is the code expanded before the Labeled Component is printed.

Note that the Property is expanded *outside* the local group of the Labeled Compoent.

```
  \ccSetProperty{labeled-meta-before}{}
```

`labeled-meta-after` `<any>` is the code expanded after the Labeled Component is printed.

Note that the Property is expanded *outside* the local group of the Labeled Compoent.

```
  \ccSetProperty{labeled-meta-after}{\par}
```

```
}
```

```
</meta>
```

Module 6

coco-headings.dtx

```
<*headings>
```

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

```
%%
%% module for CoCoTeX that extends heading objects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-headings}
   [2024/12/13 v0.5.0 CoCoTeX headings module]
\RequirePackage{coco-meta}
```

Headings are handled differently with `cocotex.cls` compared to standard 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 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:

```
\RequirePackage{bookmark}%
```

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

```
\hypersetup{bookmarksdepth=-999}%
```

1 Facility for declaring heading levels and their layouts

`Heading` is an abstract parent Container for headings. It inherits from `CommonMeta`.

```
\ccDeclareContainer{Heading}{%
  \ccInherit{Components,Properties}{CommonMeta}%
  \ccDeclareType{Parent}{}%
  \ccDeclareType{Components}{%
```

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

```
    \cch@provide@authors%
```

The remaining Components are built as usual.

`Title` is the main title of the heading.

```
    \cch@provide@comp{Title}%
```

`Subtitle` is an optional second-level title of the heading.

```
    \cch@provide@comp{Subtitle}%
```

`Number` is the heading's counter.

```
    \cch@provide@comp{Number}%
```

`RefLabel` is a unique ID of an heading. It is targeted by cross references and replaces LaTeX's `\label` command.

```
    \ccDeclareComponent{RefLabel}{}{}%
```

```
    \cch@provide@quotes
  }%
  \ccDeclareType{Attributes}{%
```

`class <string>` is the style class of the heading.

```
    \ccDeclareAttributeHandler{class}{%
      \let\cch@current@class\ccAttrVal
      \expandafter\ccUseStyleClass\expandafter{\ccAttrVal}{Heading}%
    }%
```

`notag` is a flag that if set tags the entire heading and its Components as Artifacts. Content after the heading is *not* tagged as its own `<Sect/>`, but belongs to the `<Sect/>` of the last, non-flagged, heading.

```
    \ccDeclareAttributeHandler*{notag}{%
      \let\ccaEnable\relax
      \let\ccaProtect\ccaDisable
      \global\let\cch@notag\relax
      \global\advance\cch@total@nesting@level\m@ne\relax
      \ccaArtifact\ccaDisable
    }%
```

```
  }
  \ccDeclareType{Properties}{}%
  \ccDeclareEnv{\cch@heading}{\cch@end@heading}%
}
```

`\ccDeclareHeading` is the user-level macro to declare new headings. There also exists a *starred* version of this macro, which exempts the declared heading from auto-tagging.

#1	(optional) inherit-from: load all properties from that heading level, first.
#2	level: used for toc entries. -1 for part, 0 for chapter, 1 for section, etc.
#3	name: part, chapter, section, etc, to be used in toc, head lines, bookmarks, etc.
#4	Property definitions and switches

```
\long\def\ccDeclareHeading{\global\let\cch@star@hdg\@undefined\@ifstar{\global\let\cch@star@hdg\
    relax\cc@declare@heading}{\cc@declare@heading}}
\long\def\cc@declare@heading{\cc@opt@empty\@cc@declare@heading}
\long\def\@cc@declare@heading[#1]#2#3#4{%
```

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

```
\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.

```
  \ccPackageInfo{Headings}{}{Appending to `#3'}%
  \ifcsstring{cch@#3@level}{#2}{}{%
      \ccPackageError{Headings}
        {Level Mismatch}
        {Level of heading `#3' cannot be altered!}
        {The already existing heading `#3' has toc level `\csname cch@#3@level\endcsname', but
             your^^J%
         re-declaration states `#2'.^^J%
         ^^J%
         Consider declaring a new heading alltogether with `#3' as parent,^^J%
         or add Properties to `#3' using \string\ccAddToType\string{Properties\string}\string
              {#3\string}.}%
     }%
```

we also check the parent.

```
  \if!#1!\else
    \ifcsstring{cc@parent@#3}{#1}{}{%
      \ccPackageError{Headings}
        {Parent Mismatch}
        {Parent of heading `#3'^^J cannot be altered!}
        {The already existing heading `#3' inherits from `\csname cc@parent@#3\endcsname',^^J%
         but your re-declaration sets Parent to `#1'.^^J%
         ^^J%
         Consider declaring a new heading alltogether with `#1' as parent.}%
    }%
  \fi
```

and finally pass the new Properties to the existing heading.

```
  \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.

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

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

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

```
    \csgdef{cch@#3@level}{#2}%
```

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

```
    \ifx\@argi\@empty
      \ccInherit{Components,Properties}{Heading}%
    \else
      \ccInherit{Components,Properties,Parent}{#1}%
    \fi
```

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

```
    \ccDeclareType{Properties}{%
      #4%
    }%
```

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

```
    \ccDeclareType{Init}{%
      \cch@init@hooks{#3}%
      \let\@cch@cur@cont\cc@cur@cont
      \def\cc@cur@cont{Heading}%
      \cc@init@l@{toc}{#2}{#3}%
      \let\cc@cur@cont\@cch@cur@cont
      \cch@init@cnt{#3}%
    }%
```

Unlike other Sub-Containers, headings form no own LaTeX 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.

```
    \cch@declare@heading{#2}{#3}%
   }% \ccDeclareContainer{#3}
   \ccWhenAlly{\ccaAddRolemap{#3}{Sect}}%
  }% \ifcsdef cc@container@#3 fi
}% \cc@declare@heading
```

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

```
\def\cch@create@parent#1#2{%
  \def\ccCurSecName{#2}%
  \if!#1!\else
    \ccCheckParent{#1}{#2}%
  \fi%
}
```

`\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.

```
\def\cch@declare@heading#1#2{%
  \ccEvalType{Parent}%
  \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.

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

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

```
  \ccSetContainer{Heading}%
  \@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.

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

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

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

Processing the Quote Group Container, if any.

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

Hyperref related stuff.

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

Call the mechanism to calculate the heading's counter.

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

Here, the actual construction of the heading begins.

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

Add vertical space before the heading

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

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

```
  \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.

```
  \ccIfProp{after-skip}{\expandafter\global\expandafter\@tempskipa\expandafter=\ccUseProperty{
      after-skip}\relax}{\global\@tempskipa=1sp\relax}%
```

```
  \cch@use@hook{print/before}{#2}%
  \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.

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

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

```
    \ccCreateLabel{#2}% label facility
    \leftskip\ccUseProperty{margin-left}%
    \rightskip\ccUseProperty{margin-right}%
    \bgroup
```

If Accessibility features are active, we add the start Tag for `<SectMeta>` that contains all meta data belonging to the heading (i.e. all heading Components sans the title, which is made into the head of the section). We immediately retrieve the Tag's ID since we will move it, later.

```
      \ccWhenAlly{%
        \ccaStructStart{SectMeta}%
        \ccaSaveCurStruct{cch@id@cur@meta}%
      }%
      \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 progammers need to make sure that no page breaks are allowed within the `heading-block`!

```
      \ccIfPropVal{no-toc}{true}{}{\cch@make@toc}% ToC entries
      \ccIfPropVal{no-BM}{true}{}{\cch@make@bookmarks}% Bookmarks
      \ccUseProperty{toc-hook}%
      \ccIfProp{extended}{\ccUseProperty{extended-heading}}{}%
```

Here, we end the `</SectMeta>` tag.

```
      \ccWhenAlly{\ccaStructEnd{SectMeta}}%
    \egroup%
    \cch@make@run% Running headers
    \cch@use@hook{run/after}{#2}%
    \ccUseProperty{after-heading-block}%
    \cch@use@hook{after}{#2}%
  }%
```

Finally, we decide whether the printable material we stored in `\@svsec` is to be rendered as a block or inline. This is adopted from LaTeX'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.

```
  \ifdim\@tempskipa <\z@\relax
    \cch@make@inline%
  \else
    \cch@make@block%
  \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 environemnt is closed before we actually print the fully composed heading. The definition of `\next` happens in either `\cch@make@inline` or `\cch@make@block`.

```
  \aftergroup\next%
 }%
}
```

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

```
\def\cch@use@hook#1#2{%
  \expandafter\ifx\csname cc@parent@#2\endcsname\relax\else
    \letcs\@cch@parent{cc@parent@#2}%
    \cch@use@hook{#1}{\csname cc@parent@#2\endcsname}%
  \fi
  \UseHook{cc/headings/#2/#1}%
  \ignorespaces}
```

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

```
\def\cch@add@before@skip{%
  \setlength\@tempskipa{\ccUseProperty{before-skip}}%
  \ifdim\@tempskipa<\z@\relax
    \def\do@skip{\minusvspace{-\@tempskipa}}%
  \else
    \def\do@skip{\addvspace{\@tempskipa}}%
  \fi%
  \if@nobreak
    \everypar{}%
    \do@skip
  \else
    \addpenalty\@secpenalty
    \do@skip
  \fi}
```

1.1 Initializers for New Heading Levels

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

```
\def\cch@init@hooks#1{%
```

cc/headings/[level]/toc/before is exanded before the ToC entry is printed

```
  \NewHook{cc/headings/#1/toc/before}%
```

cc/headings/[level]/toc/after is exanded after the ToC entry is printed

```
  \NewHook{cc/headings/#1/toc/after}%
```

cc/headings/[level]/before is expanded before the before-heading property called

```
  \NewHook{cc/headings/#1/before}%
```

cc/headings/[level]/after is expanded after the after-heading-block property was called.

```
  \NewHook{cc/headings/#1/after}%
```

cc/headings/[level]/print/before is expanded just before \@svsec is locally defined.

```
  \NewHook{cc/headings/#1/print/before}%
```

cc/headings/[level]/run/after is expanded after the local RunTitle has been generated

```
  \NewHook{cc/headings/#1/run/after}%
```

env/Heading/[level]/begin is the hook that is called at the begin of each defined heading level with the name [name]. It is called at the beginning of every Heading environment whose mandatory argument matches [name] immediately before the Instance's Attributes are evaluated.

```
  \NewHook{env/\ccPrefix Heading/#1/begin}%
```

If the current heading is derived from a parent, we want the parent's hooks to also apply to the child:

```
  \ifcsname cc@parent@#1\endcsname
    \AddToHook{env/\ccPrefix Heading/#1/begin}{\UseHook{env/\ccPrefix Heading/\csname cc@parent@
         #1\endcsname/begin}}%
  \fi
}
```

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

```
\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.

```
\def\cch@auto@number#1#2{%
  \ccIfPropVal{numbering}{auto}
    {\expandafter\ifx\csname c@#2\endcsname\relax\cch@init@cnt{#2}\fi
     \ccIfAttrIsSet{Heading}{nonumber}{}
       {\ccIfComp{Number}
         {}
         {\ifnum #1>\c@secnumdepth\relax\else
            \stepcounter{#2}%
            \edef\@tempa{\csname the#2\endcsname}%
            \ccComponentEA{Number}{\@tempa}%
          \fi}}
     }{}}
```

2 Externalisation of Heading Compoents

Components of headings may be used far away from the heading itself. Since, by design, Components are defined strictly local within their containers, those externale usages demand special treatment.

2.1 Common Stuff

\cch@set@author@name@list sets the #1AuthorNameList Component.

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

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

```
  \ccUnlessComp{#1AuthorNameList}{%
```

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

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

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

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

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

```
      \ifnum\ccAuthorCnt>\z@
        \ccdefFromCountedComp\cch@tempa{Author}{author-list-print-format}%
        \ifx\cch@tempa\relax\else
          \ccComponent{#1AuthorNameList}{\cch@tempa}%
        \fi
      \fi
    \fi
  }}%
```

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.

```
\def\cch@make@toc{%
  \cc@check@empty{Heading}{Title}{Toc}%
  \cc@check@empty{Heading}{Number}{Toc}%
  \cc@check@empty{Heading}{Subtitle}{Toc}%
  \cch@set@author@name@list{Toc}%
  \ccIfAttrIsSet{Heading}{notoc}{}
    {\protected@edef\cch@toc@entry{%
       \ccIfComp{TocTitle}{\string\ccComponent{TocTitle}{\string\ignorespaces\space\expandonce{\
           cc@Heading@TocTitle}}}{}%
       \ccIfComp{TocNumber}{\string\ccComponent{TocNumber}{\string\ignorespaces\space\expandonce
           {\cc@Heading@TocNumber}}}{}%
       \ccIfComp{TocAuthorNameList}{\string\ccComponent{TocAuthorNameList}{\string\ignorespaces\
           space\expandonce{\cc@Heading@TocAuthorNameList}}}{}%
       \ccIfComp{TocSubtitle}{\string\ccComponent{TocSubtitle}{\string\ignorespaces\space\
           expandonce{\cc@Heading@TocSubtitle}}}{}%
     }%
     \ccIfProp{toc-level}
       {\edef\cch@toc@sec@name{\ccUseProperty{toc-level}}}
       {\let\cch@toc@sec@name\ccCurSecName}%
     \protected@write\@auxout
       {\ccGobble\ccaProtect}%
       {\string\@writefile{toc}{\protect\ccContentsline{\cch@toc@sec@name}{\cch@toc@entry}{\
           thepage}{\@currentHref}\protected@file@percent}}\relax
     \ccCreateContentListEntries{Heading}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage}{\
         @currentHref}%
     \ccCreateContentListEntries{\cch@toc@sec@name}{\cch@toc@sec@name}{\cch@toc@entry}{\thepage
         }{\@currentHref}%
     \protected@write\@auxout{\ccaEnable}{}%
    }}
```

\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.

```
\def\cc@toc@extract@data#1#2#3#4{%
  \ccSetContainer{Heading}%
  \ccEvalType[#2]{Properties}%
  \ccDeclareComponent{TocPage}{}{}%
  \ccComponent{TocPage}{\ccUseProperty{toc-page-face}#4}%
  \ccDeclareComponent{TocTitle}{}{}%
  \ccDeclareComponent{TocSubtitle}{}{}%
  \ccDeclareComponent{TocNumber}{}{}%
  \ccDeclareComponent{TocAuthorNameList}{}{}%
  \cc@expand@l@contents{#3}{Heading}{Toc}{Title}%%
  \cc@format@number{toc-}{Toc}{#1}%
}
```

`\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.

```
\def\cc@toc@print@entry#1{%
  \bgroup
    \cch@use@hook{toc/before}{#1}%
    \ccUseProperty{toc-before-entry}%
    \ccUseProperty{toc-format}%
    \cch@use@hook{toc/after}{#1}%
    \ccUseProperty{toc-after-entry}%
  \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.

```
\def\cch@make@run{%
  \cc@check@empty{Heading}{Title}{Run}%
  \cc@check@empty{Heading}{Number}{Run}%
  \cc@check@empty{Heading}{Subtitle}{Run}%
  \cch@set@author@name@list{Run}%
  \ccUseProperty{running-extra}%
  \ccIfProp{running-level}
    {\letcs\cch@mark@name{\ccUseProperty{running-level}mark}}
    {\letcs\cch@mark@name{\ccCurSecName mark}}%
    \letcs\cch@parent{cc@parent@\ccCurSecName}%
    \ifx\cch@mark@name\@undefined
      \ifx\cch@parent\relax\else
        \letcs\cch@mark@name{\cch@parent mark}%
      \fi
    \fi
  \ifx\cch@mark@name\@undefined\else
    \begingroup
      \ccGobble
      \protected@edef\@tempa{\csname cc@Heading@running-heading\endcsname}%
      \expandafter\cch@mark@name\expandafter{\@tempa}%
    \endgroup
  \fi
}
```

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.

```
\def\cch@make@bookmarks{%
  \cc@check@empty[Toc]{Heading}{Title}{BM}%
  \cc@check@empty[Toc]{Heading}{Number}{BM}%
  \cc@check@empty[Toc]{Heading}{AuthorNameList}{BM}%
  \cc@check@empty[Toc]{Heading}{Subtitle}{BM}%
  \ccIfAttrIsSet{Heading}{noBM}{}
    {\ccIfProp{bookmark-level}{\edef\Hy@toclevel{\ccUseProperty{bookmark-level}}}{}%
     \begingroup
       \ccGobble
       \protected@edef\@tempa{\csname cc@Heading@bookmark\endcsname}%
       \bookmark[level=\Hy@toclevel,dest=\@currentHref]{\expandonce{\@tempa}}%
     \endgroup
   }}
```

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.

```
\newbox\cch@inline@sec@box
\def\cch@make@inline{%
  \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
  \global\setbox\cch@inline@sec@box\hbox{\ifvoid\cch@inline@sec@box\else\unhbox\
      cch@inline@sec@box\ccUseProperty{interline-para-sep}\fi\@svsec}%
  \ccaEnable
  \@nobreakfalse
  \global\@noskipsectrue
  \gdef\next{%
    \global\everypar{%
      \if@noskipsec
        \global\@noskipsecfalse
        {\setbox\z@\lastbox}%
        \clubpenalty\@M
        \begingroup
          \unhbox\cch@inline@sec@box
        \endgroup
        \unskip
        \hskip -\@tempskipa
      \else
        \clubpenalty \@clubpenalty
        \global\setbox\cch@inline@sec@box\box\voidb@x
        \everypar{}%
      \fi}%
    \ignorespaces}}
```

3.2 Block Headings

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

```
\def\cch@make@block{%
  \@svsec
  \ccUseProperty{after-heading-par}%
  \ccIfProp{after-indent}{\global\@afterindenttrue}{\global\@afterindentfalse}%
  \gdef\next{%
    \ifdim\parskip>\z@\relax\advance\@tempskipa-\parskip\relax\fi
    \vskip \@tempskipa
    \@afterheading
    \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.

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

Some LaTeX kernel macros are saved, the namespace is set and counted groups from previous headings are reset.

```
  \cch@reserve
```

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

```
  \xdef\ccCurSecName{#2}%
  \ccEvalAttributes[Heading]{#1}%
```

After opening the environment, if accessibility Features are enabled, we check the current absolute nesting level and close all open Sectioning tags if the nominal level of the current heading is lower than the *nominal* level of the last opened heading. *Nominal* in this case refers the level given to the heading when it was defined in contrast to the actual, *absolute*, nesting level in the tex document.

```
  \ifx\cch@notag\relax\else
    \ccWhenAlly{%
```

`\ccPrevSecLevel` is the previously opened, *nomimal*, heading level.

```
      \global\let\ccPrevSecLevel=\ccCurSecLevel\relax
```

`\ccCurSecLevel` stores the nominal level of the current heading.

```
      \xdef\ccCurSecLevel{\csname cch@#2@level\endcsname}%
```

Now, we call the auto-close mechanism defined below in Sect. 6,

```
      \cchAutoClose%
      \ccDebugMsg[a11y]{Level after close: \the\cch@total@nesting@level}}%
  \fi
```

Then, we call the heading level specific `env/Heading/[level]/begin` hook.

```
  \UseHook{env/\ccPrefix Heading/#2/begin}%
  \ccEvalType[#2]{Components}%
  \ignorespaces
}
```

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

```
\def\cch@end@heading{%
  \expandafter\ifx\csname ccUseHeading\ccCurSecName\endcsname\relax
    \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.}%
  \else
    \csname ccUseHeading\ccCurSecName\endcsname%
  \fi
  \cch@reset
}
```

4.2 Content Handlers

\cch@reserve re-directs some of LaTeX's kernel macros and makes sure that some other macros have their default values:

```
\def\cch@reserve{%
  \ccSetContainer{Heading}%
  \let\cch@ltx@dbl@backslash\\%
  \letcs\\{\ccPrefix Break}%
  \let\cc@ltx@label\label%
  \def\ccAuthorCnt{\z@}%
  \def\ccAffilCnt{\z@}%
  \cc@reset@components{\cc@cur@cont}%
  \let\cch@current@class\relax
  \global\let\cch@id@cur@meta\@undefined
  \global\let\cch@id@cur@head\@undefined
  \global\let\cch@no@tag\@undefined%
  \global\let\cch@ccaEnable\ccaEnable
  \global\let\cch@ccaProtect\ccaProtect
  \ignorespaces}
```

\cch@reset restores LaTeX's default definitions (however, this should be unnecessary since Heading is an environment and therefore constitutes a closed group).

```
\def\cch@reset{%
  \let\cc@cur@cont\relax
  \let\\\cch@ltx@dbl@backslash
  \let\label\cc@ltx@label
  \let\ccCurSecName\relax
  \global\let\ccaEnable\cch@ccaEnable
  \global\let\ccaProtect\cch@ccaProtect
  \ifx\cch@notag\relax\endccaArtifact\fi
  }
```

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

```
\def\cch@provide@quotes{%
```

QuoteBlockCL is the Collection Component for one or more Quote Component Groups.

```
  \ccDeclareComponent{QuoteBlock}{}{}%
```

QuoteGC is a Component Group for quotes that belong to a heading.

```
  \ccDeclareComponentGroup{Quote}{%
```

QuoteTextCC is the quotation text

```
    \ccDeclareCountedComponent{QuoteText}%
```

QuoteSourceCC is the source of the quotation.

```
    \ccDeclareCountedComponent{QuoteSource}%
  }%
}
```

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

```
\def\cch@provide@authors{%
  \ccAddToRole{Author}{%
```

AuthorContactCC holds the contact information of an author.

```
    \ccDeclareCountedComponent{AuthorContact}%
  }%
```

AuthorContactBlockCL is the Collection Component for the Counted Component AuthorContact.

```
  \ccDeclareRoleBlock{Author}{ContactBlock}{author-contact-block-format}%
```

```
  \ccDeclareGroupHandler{Author}{%
    \ccIfComp{AuthorContact}{}{\ccComponent{AuthorContact}{\ccUseProperty{author-contact-format
        }}}{}%
  }%
```

AuthorNameListCL is the Collection Component for the Author names.

```
  \cc@provide@overrides{AuthorNameList}%
}
```

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

```
\def\cch@provide@comp#1{%
  \ccDeclareComponent{#1}{}{}%
  \cc@provide@overrides{#1}%
}
```

\cc@provide@overrides declares the Component macros for a Heading Component's overrides. #1 is the Component name. The overrides allow a four-way distinction between *i* the data printed in-situ (#1), *ii* data sent to toc (Toc#1), (iii) data sent to the page styles (Run#1), and (iv) the data sent to the PDF bookmarks (BM#1).

```
\def\cc@provide@overrides#1{%
  \ccDeclareComponent{Toc#1}{}{}% toc overrides
  \ccDeclareComponent{Run#1}{}{}% running overrides
  \ccDeclareComponent{BM#1}{}{}% bookmark overrides
}
```

5 Defaults

```
\ccAddToProperties{Heading}{%
```

`interline-para` `[true|<empty>]` is a switch that if non-empty prevents two adjacent inline headings from being set in the same paragraph.

```
  \ccSetProperty{interline-para}{}%
```

`interline-para-sep` `<any>` is the material that is printed between to adjacent inline headings.

```
  \ccSetProperty{interline-para-sep}{\space}
```

`heading-par` `<any>` is the material added to the very beginning of a heading.

```
  \ccSetProperty{heading-par}{%
    \ccWhenProp{interline-para}{\if@noskipsec \leavevmode \fi}%
    \par
    \global\@afterindenttrue
  }%
```

`after-heading-par` `<any>` is expanded at the very end of non-inline headings.

```
  \ccSetProperty{after-heading-par}{\par \nobreak}%
```

`before-heading` `<any>` is expanded immediately before any vertical skips of a heading are inserted, but after the `begin-hook`.

```
  \ccSetProperty{before-heading}{}%
```

`title-face` `<any>` is the style of the heading's main title.

```
  \ccSetProperty{title-face}{\bfseries}%
```

`subtitle-face` `<any>` is the style of the heading's subtitle.

```
  \ccSetProperty{subtitle-face}{\normalfont}%
```

`author-face` `<any>` is the face of the heading's printed `Author` Component.

```
  \ccSetProperty{author-face}{\normalfont}%
```

`quote-face` `<any>` is the style of a quotation.

```
  \ccSetProperty{quote-face}{\raggedleft}%
```

`quote-source-face` `<any>` is the style of a quotation's source line.

```
  \ccSetProperty{quote-source-face}{}%
```

quote-block-format <any> is the format of a single quotation. By default, it uses the QuoteText and QuoteSource Components.

```
\ccSetProperty{quote-block-format}{%
  \bgroup
    \ccUseProperty{quote-face}%
    \ccUseComp{QuoteText}\par
    \ccIfComp{QuoteSource}{{\ccUseProperty{quote-source-face}--\space\ccUseComp{QuoteSource}}\
        par}{}%
  \egroup}
```

heading-block <any> is the format of the main heading. It uses the Subtitle, AuthorNameList, QuoteBlock and AffilBlock Components.

```
\ccSetProperty{heading-block}
  {\ccUseProperty{main-title-format}%
   \ccaStructStart{Div}%
   \ccWhenComp{Subtitle}{{\ccUseProperty{subtitle-face}\ccaStructStart{P}\ccUseComp{Subtitle}\
        ccaStructEnd{P}}\par\nobreak}%
   \ccWhenComp{AuthorNameList}{{\ccUseProperty{author-face}\ccaStructStart{P}\ccUseComp{
        AuthorNameList}\ccaStructEnd{P}}\par\nobreak}%
   \ccWhenComp{QuoteBlock}{\ccaStructStart{Div}\ccUseComp{QuoteBlock}\ccaStructStart{Div}}%
   \ccWhenComp{AffilBlock}{{\ccUseProperty{affil-block-face}\ccaStructStart{Div}\ccUseComp{
        AffilBlock}\ccaStructStart{Div}}\par}%
   \ccaStructEnd{Div}%
  }%
```

main-title-format <any> is the format of the heading's main title. It should also enclose the heading's Number and Title Components with Tags that are mapped to <H/> or <Hn/> with $1 < n < 6$. The number is tagged as <Lbl/> if present.

```
\ccSetProperty{main-title-format}{%
  \ccUseProperty{title-face}%
  \cchHeadTagStart
  \ccIfComp{Number}%
    {\ccaStructStart{Lbl}\ccUseProperty{hang-number}\ccaStructEnd{Lbl}}%
    {\leftskip0pt}%
  \ccUseComp{Title}%
  \cchHeadTagEnd
  \par\nobreak
}
```

extended-heading <any> is the format of extended headings whcih incorporates the Abstract and Keywords Labeled Components. Requires the extended Property to be non-empty.

Abstract and Keywords are tagged with <Abstract/> and <Keyword/>, their labels with <AbstractLabel/> and <KeywordsLabel/>, and their values with <AbstractText/> and <KeywordsText/>, respectively.

```
\ccSetProperty{extended-heading}{%
  \ccWhenComp{Abstract}{%
    \par\vskip\baselineskip
    \ccaStructStart{Abstract}%
    \bgroup
      \bfseries
      \ccaStructStart{AbstractLabel}%
      \ccIfComp{AbstractLabel}
        {\ccUseComp{AbstractLabel}}
        {Abstract}%
      \ccaStructEnd{AbstractLabel}%
    \egroup
```

```
      \par\nobreak
      \bgroup
        \itshape\small
        \ccaStructStart{AbstractText}%
        \ccUseComp{Abstract}%
        \ccaStructEnd{AbstractText}%
      \egroup
      \par
      \ccaStructEnd{Abstract}%
    }%
    \ccWhenComp{Keywords}{%
      \par\vskip\baselineskip
      \ccaStructStart{Keywords}%
      \bgroup
        \bfseries
        \ccaStructStart{KeywordsLabel}%
        \ccIfComp{KeywordsLabel}
          {\ccUseComp{KeywordsLabel}}
          {Keywords}%
        \ccaStructEnd{KeywordsLabel}%
      \egroup%
      \par\nobreak
      \bgroup
        \itshape\small
        \ccaStructStart{KeywordsText}%
        \ccUseComp{Keywords}%
        \ccaStructEnd{KeywordsText}%
        \par%
      \egroup
      \ccaStructEnd{Keywords}%
    }}%
```

`before-skip` `<skip>` the vertical space before heading. Positive values are set with LaTeX's `\addvspace`, while negative values are set with coco-common's .

`\minusvspace`

TODO
values $< 0pt$ use \minusvspace, else \addvspace. LaTeX's default behaviour of \@afterindent is relocated to the after-indent property.

```
\ccSetProperty{before-skip}{\z@skip}%
```

`after-heading-block` `<any>` is expanded at the very end of the printed heading.

```
\ccSetProperty{after-heading-block}{}%
```

`before-heading-block` `<any>` is expanded at the very beginning of `@svsec`.

```
\ccSetProperty{before-heading-block}{\parindent\z@ \parskip\z@}%
```

`toc-hook` `<any>` is called after ToC and Bookmark entries are written and allows for material to be added to the toc file.

```
\ccSetProperty{toc-hook}{}% Called, after ToC and BM entries have been written to the .aux file
```

after-indent `<any>` if non-empty, the first paragraph after the heading will be indented.

```
\ccSetProperty{after-indent}{}%
```

margin-left `[auto|<dimen>|<empty>]` is the left margin of the heading. Its value can either be a fixed dimension, the string `auto`, or empty. If the Property is set to `auto` or an empty string, the margin is calculated from the indent (see below). Otherwise the fix value is used.

```
\ccSetProperty{margin-left}{}%
```

margin-right `<skip>` is the right margin of the heading block.

```
\ccSetProperty{margin-right}{\@flushglue}%
```

after-skip `<skip>` is the vertical space after the heading block. If the value is greater than or equal to 0pt, the heading is formatted in block, while it is formatted as inline heading if the value is negative.

```
\ccSetProperty{after-skip}{1sp}%
```

indent `[auto|auto-global|<dimen>]` is the offset of the first line of the heading relative to margin-left.

If the value is auto, the indent of the heading is the width of the widest Number Component of *all headings with the same level*.

If the value is `auto-global`, the indent is the width of the widest Number component across *all heading levels*. Both `auto` and `auto-global` require at least two LaTeX runs. See Sect. 3.3 in Module Module 3 for more details.

```
\ccSetProperty{indent}{auto}%
```

number-width `<dimen>` is the (actula) width of the `Number` component.

```
\ccSetProperty{number-width}{}%
```

number-sep `<any>` Is the separator between the `Number` and the `Title` components

```
\ccSetProperty{number-sep}{\space}%
```

number-align `[left|center|right]` is the horizontal alignment of the `Number` component inside its surrounding `\hbox`.

```
\ccSetProperty{number-align}{left}%
```

number-format `<any>` is the format of a heading's counter. It prints the Number component and the number-sep Property, and stylizes them both with the title-face *and* number-face Properties.

```
\ccSetProperty{number-format}{%
  \bgroup
    \ccUseProperty{title-face}%
    \ccUseProperty{number-face}%
    \ccUseComp{Number}%
    \ccUseProperty{number-sep}%
  \egroup}
```

numbering `[auto|<any>]` if non-auto, headings are not numbered automatically if no Number component is given. This property can be overridden in a local instance with the nonumber Attribute.

```
\ccSetProperty{numbering}{auto}%
```

`running-level` `<name>` is an override that allows the heading's running title to appear as another level's running title. Usually, the `RunTitle` Component is passed to `\<level>mark` for the page header, but if this Property is non-empty, the heading will be passed to `\<runnning-level>mark`, instead.

```
\ccSetProperty{running-level}{}% override level for running title, name
```

`running-heading` `<any>` is the format of the material passed to the `\<level>mark` or `\<running-level>mark` command. It uses the `RunTitle` and `RunAuthorNameList` Components.

```
\ccSetProperty{running-heading}{%
  \ccIfComp{RunAuthorNameList}{\ccUseComp{RunAuthorNameList}:\space}{}%
  \ccUseComp{RunTitle}%
}%
%% ToC
```

`no-toc` `[true|false]` whether or not the heading does *not* create an entry in the table of contents (`true` means no toc entry, `false` means toc entry).

```
\ccSetProperty{no-toc}{false}%
```

`no-BM` `[true|false]` whether or not the heading does *not* create a bookmark (`true` means no bookmark, `false` means bookmark).

```
\ccSetProperty{no-BM}{false}%
```

`toc-margin-top` `<skip>` vertical space before the ToC entry.

```
\ccSetProperty{toc-margin-top}{\z@}%
```

`toc-margin-bottom` `<skip>` vertical space after the ToC entry.

```
\ccSetProperty{toc-margin-bottom}{\z@}%
```

`toc-margin-left` `[auto|<dimen>]` left margin of the toc entry. See `margin-left` for the meaning of auto.

```
\ccSetProperty{toc-margin-left}{auto}%
```

`toc-margin-right` `<dimen>` right margin of the ToC entry.

```
\ccSetProperty{toc-margin-right}{\@pnumwidth}%
```

`toc-title-face` `<any>` style of the title in the ToC entry.

```
\ccSetProperty{toc-title-face}{}%
```

`toc-indent` `[auto|auto-global|<dimen>]` offset of the ToC entry's first line relative to margin-left. See `indent`.

```
\ccSetProperty{toc-indent}{auto}%
```

`toc-number-width` `<dimen>` the actual width of the TocNumber Component.

```
\ccSetProperty{toc-number-width}{}%
```

`toc-number-align` `[left|center|right]` the alignment of the TocNumber within the surrounding `\hbox`.

```
\ccSetProperty{toc-number-align}{left}%
```

`toc-number-face` `<any>` style of the TocNumber component.

```
\ccPropertyLet{toc-number-face}{toc-title-face}%
```

toc-number-sep <any> separator between the TocNumber and TocTitle Components

```
\ccSetProperty{toc-number-sep}{\enskip}%
```

toc-number-format <any> is the format of the TocNumber Component, using the toc-number-face and toc-number-sep Properties.

```
\ccSetProperty{toc-number-format}{%
  \bgroup
    \ccUseProperty{toc-number-face}%
    \ccUseComp{TocNumber}%
    \ccUseProperty{toc-number-sep}%
  \egroup}
```

toc-page-sep <any> separator between the TocTitle and the page counter. The dotted line is tagged as <leaders/>, which is mapped to be an artifact of type <leaders/>.

```
\ccSetProperty{toc-page-sep}{\ccaAddKeep\ccaStructStart[Document]{leaders}\dotfill\
    ccaStructEnd{leaders}}%
```

toc-page-face <any> style of the page counter

```
\ccSetProperty{toc-page-face}{}%
```

toc-page-format <any> format of the page counter using the toc-page-sep and toc-page-face Properties. The number itself is tagged as a <Span/>.

```
\ccSetProperty{toc-page-format}{%
  \ccUseProperty{toc-page-sep}%
  \bgroup
    \ccUseProperty{toc-page-face}%
    \ccaStructStart{Span}\ccUseComp{TocPage}\ccaStructEnd{Span}%
  \egroup}%%
```

toc-level <name> name of another heading level as which the ToC entry should be rendered.

```
\ccSetProperty{toc-level}{}%
```

toc-before-entry <any> is expanded before any ToC entry is rendered. Should setup margins, alignment, line-breaking rules, etc.

```
\ccSetProperty{toc-before-entry}{%
  \addvspace{\ccUseProperty{toc-margin-top}}%
  \parindent \z@
  \let\\\@centercr
  \hyphenpenalty=\@M
  \rightskip \ccUseProperty{toc-margin-right} \@plus 1fil\relax
  \parfillskip -\rightskip
  \leftskip\ccUseProperty{toc-margin-left}%
}%
```

toc-after-entry <any> is expanded at the very end of a ToC entry. By default, it sets the skip after the entry to toc-margin-bottom.

```
\ccSetProperty{toc-after-entry}{\par\addvspace{\ccUseProperty{toc-margin-bottom}}}%
```

toc-format <any> format of the ToC entry itself. It uses the toc-title-face, toc-hang-number and toc-page-format Properties to print the TocNumber, TocAuthorNameList, TocTitle, and TocPage Components. Tagging should incorporate the <TOCI/>, as well as for numbered ToC entries the <P/> and <Reference/> tags that span the entire entry, as well as <Lbl/> for the TocNumber, and <Span/> for the remaining Components of the entry. Unnumbered Entries are simply <TOCI/> with <Spans/> for the various Components.

```
\ccSetProperty{toc-format}{%
  \ccUseProperty{toc-title-face}%
  \ccaVstructStart{TOCI}%
  \ccWhenComp{TocNumber}{\ccaStructStart{P}\ccaStructStart{Reference}}%
  \ccTocLink{%
    \ccIfComp{TocNumber}
      {\ccaStructStart{Lbl}\ccUseProperty{toc-hang-number}\ccaStructEnd{Lbl}}
      {\leftskip0pt\leavevmode}%
    \ccWhenComp{TocAuthorNameList}{\ccaStructStart{Span}\ccUseComp{TocAuthorNameList}:\space\
        ccaStructEnd{Span}}%
    \ccaStructStart{Span}\ccUseComp{TocTitle}\ccaStructEnd{Span}%
    \ccUseProperty{toc-page-format}%
  }%
  \ccWhenComp{TocNumber}{\ccaStructEnd{Reference}\ccaStructEnd{P}}%
  \ccaVstructEnd{TOCI}%
}%
```

bookmark-level <num> number(!) of the heading level as which the Bookmark entry should be rendered.

```
\ccSetProperty{bookmark-level}{}%
```

bookmark <any> is the format of the bookmark, which by default is built only from the BMNumber and BMTitle Components.

```
\ccSetProperty{bookmark}{%
  \ccIfComp{BMNumber}{\ccUseComp{BMNumber}\space}{}%
  \ccUseComp{BMTitle}%
}%
```

orcid-link <any> how an ORCID link is rendered.

```
\ccSetProperty{orcid-link}{%
  \ccIfComp{ORCID}{\ccCompLink{ORCID}{\includegraphics[height=1em]{logos/ORCID.pdf}}}{}%
}%
```

author-contact-format <any> how a single Author Component's contact information should be rendered. By default, it uses the Author's FullName, the value of the AffilRef component as superscript, and the orcid-link Property.

```
%% a single Author's contact infomration block
\ccSetProperty{author-contact-format}{%
  \ccUseComp{FullName}\ccWhenComp{RefAffil}{\textsuperscript{\ccUseComp{AffilRef}}}%
  \ccUseProperty{orcid-link}%
}%
```

author-list-format <any> how a single entry in the AuthorNameList Collection Component should be rendered.

```
\ccPropertyLet{author-list-format}{author-list-print-format}%
```

author-contact-block-format <any> is the Collection Property for the AuthorContactBlock Collection Component and sets how each single entry in the Collection should be formatted. By default, it uses the AuthorContact Counted Component and appends the counted-name-sep to all instance of that Component but the last.

```
\ccSetProperty{author-contact-block-format}{%
  \ccUseComp{AuthorContact}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi
}}
```

6 Accessibility Features

`\cch@total@nesting@level` stores the absolute nesting level opened with each heading.

```
\newcount\cch@total@nesting@level \cch@total@nesting@level=\z@\relax
```

`\cch@max@nesting@level` stores the highest absolute nesting level throughout the document.

```
\newcount\cch@max@nesting@level \cch@max@nesting@level=\z@\relax
```

Since PDF 1.7 allows only `<H1/>`–`<H6/>`, more nesting levels need to be mapped to one of the Standard tags. The Tagged PDF Best Practice Guide recommends mapping higher levels to `<P/>`, which we do at the end of the document. All this is not necessary if PDF/UA-2 is used, since first it allows any Hn for as long as Hns are nested sequentially, and second, because then we can use the `<Title/>` tag instead of `<Hn/>`.

```
\ccWhenAlly{\if@cc@pdf@two\else\AtEndDocument{\cch@add@h@rolemap}\fi}%
```

`\cch@add@h@rolemap` adds the Rolemap entries for Hn, where $n > 6$.

```
\def\cch@add@h@rolemap{%
  \@tempcnta=7\relax
  \loop
  \unless\ifnum\@tempcnta>\cch@max@nesting@level\relax
    \ccaAddRolemap{H\the\@tempcnta}{P}%
    \advance\@tempcnta\@ne\relax
  \repeat
}
```

`\cchAutoClose` checks if any open `<Sect/>` tags (or Tagging mapped to `<Sect/>`) need to be closed and closes them if applicable.

For that, we check if the value of `\ccPrevSecLevel` (i.e., the *nominal* heading level of the last opened heading) is larger or equal to the *nominal* level of the currently opened heading. If so, we add closing `</Sect>` tags until either the *absolute* nesting level is 1 or an decrementing counter that starts at `\ccPrevSecLevel` gets less than `\ccCurSecLevel`. At each step, we also decrease the `\cch@total@nesting@level` counter by one.

```
\DeclareAccessibilityCommand\cchAutoClose{%
  \let\cca@next\relax
  \ccDebugMsg[a11y]{Calling AutoClose for \ccCurSecName\space (\ccCurSecLevel).}
  \ifx\ccPrevSecLevel\@undefined\else
    \ifx\ccCurSecLevel\@undefined\else
      \ccDebugMsg[a11y]{prev: \ccPrevSecLevel, cur: \ccCurSecLevel, abs: \the\
          cch@total@nesting@level.}
      \ifnum\ccPrevSecLevel=\ccCurSecLevel\relax
        \ccDebugMsg[a11y]{Closing \the\cch@total@nesting@level.}
        \global\advance\cch@total@nesting@level\m@ne\relax
        \ccaVstructEnd{\csname cch@sec@\the\cch@total@nesting@level @name\endcsname}%
      \else
        \ifnum\ccPrevSecLevel<\ccCurSecLevel\relax
          \csxdef{cch@prev@parent@level@\ccCurSecLevel}{\ccPrevSecLevel}%
      \else
        \ifnum\ccPrevSecLevel>\ccCurSecLevel\relax
          \ccDebugMsg[a11y]{Closing (inner) \the\cch@total@nesting@level.}
          \global\advance\cch@total@nesting@level\m@ne\relax
          \ccaVstructEnd{\csname cch@sec@\the\cch@total@nesting@level @name\endcsname}%
          \ifnum\cch@total@nesting@level=\@ne\relax\else
            \xdef\ccPrevSecLevel{\csname cch@prev@parent@level@\ccPrevSecLevel\endcsname}%
            \let\cca@next\cchAutoClose
          \fi
        \fi
      \fi
    \fi
  \fi
\fi
\cca@next
}
```

`\cchResetNesting` resets the *absolute* heading nesting level to 0 by closing all currently open `</Sect>` tags. This macro is intended to be used at the very end of the document or at major document partitions (e.g., `\mainmatter`, `\appendix`, etc.)

```
\DeclareAccessibilityCommand\cchResetNesting{%
  \ccWhenAlly{%
    \ifnum\cch@total@nesting@level>\z@\relax
      \loop
        \ccaVstructEnd{\csname cch@sec@\the\cch@total@nesting@level @name\endcsname}%
        \global\advance\cch@total@nesting@level\m@ne\relax
      \ifnum\cch@total@nesting@level>\z@\relax
      \repeat
    \fi
  }}
```

The advancement of the `\cch@total@nesting@level` counter is done in the `env/Heading/before` hook.

```
\AddToHook{env/\ccPrefix Heading/before}{%
  \ccDebugMsg[a11y]{Processing heading}%
  \global\advance\cch@total@nesting@level\@ne\relax
}
```

In the `env/Heading/after` hook, first, the `</SectMeta>` Tag is closed. Only then, we open the tag for actual `<Sect>`. Here, we also define the name of the Tag to be re-used later in the `\cchAutoClose` mechanism. Finally, we move the `<Hn/>` Tagging node (which tags the section's title) and the `<SectMeta/>` node (which contains the tagged meta data realized by the `Heading` Container's other Components) into the `<Sect/>` node.

```
\AddToHook{env/\ccPrefix Heading/after}{%
  \ifx\cch@notag\relax
    \global\let\cch@notag\@undefined
  \else
    \ccaVstructStart{\ccCurSecName}%
    \csxdef{cch@sec@\the\cch@total@nesting@level @name}{\ccCurSecName}%
```

For inline headings, the actual title is not yet printed by the time the heading Container's environment is closed. In this case, we don't move the node. For one, because it is likely not yet set, and second because it most likely already is in the "right" spot, i.e. inside the `<Sect/>` tag.

```
    \ifx\cch@id@cur@head\@undefined\else
      \ccaMoveStruct{\cch@id@cur@head}%
    \fi
```

If \cch@id@cur@meta is undefined when the Heading container's environment is closed, then this indicates an inline heading. In this case, we don't need to move the <SectMeta/> node, as it most likely already is in the right place.

```
    \ifx\cch@id@cur@meta\@undefined\else
      \ccaMoveStruct{\cch@id@cur@meta}%
    \fi%
  \fi
}
```

\cchHeadTagStart is used inside Property declarations to insert the start tag of the current section

```
\def\cchHeadTagStart{%
  \if@cc@pdf@two
    \ccaVstructStart{Title}%
  \else
    \ccaVstructStart{H\the\numexpr\cch@total@nesting@level\relax}%
    \ifnum\cch@total@nesting@level>\cch@max@nesting@level
      \global\cch@max@nesting@level=\cch@total@nesting@level\relax
    \fi
  \fi
  \ccaSaveCurStruct{cch@id@cur@head}%
}
```

\cchHeadTagEnd is used inside Property declarations to insert the end tag of the current section, unless the heading level was declared with the starred version of \ccDeclareHeading.

```
\def\cchHeadTagEnd{%
  \if@cc@pdf@two
    \ccaVstructEnd{Title}%
  \else
    \ccaVstructEnd{H\the\numexpr\cch@total@nesting@level\relax}%
  \fi}
```

The <SectMeta/> tag is mapped to a simple <Div/>.

```
\ccaAddRolemap{SectMeta}{Div}
```

7 Miscellaneous

7.1 Alternative paragraph separation

\ccNewPar is a user-level macro to have a vertical skip between two local paragraphs and no indent in the second one. The amount of vertical space between the paragraphs can be adjusted with the optional argument. If #1 is omitted, \ccnewparskip is inserted, which defaults to 1\baselineskip if the dimension isn't set to something other than 0pt in the preamble. This macro is intended to be used at the end of the first of the paragraphs.

```
\newdimen\ccnewparskip \AtBeginDocument{\ifdim\ccnewparskip=\z@\relax \ccnewparskip=1\
    baselineskip\relax\fi}
\def\ccNewPar{\@ifnextchar[{\cc@newpar}{\cc@newpar[\the\ccnewparskip]}}%]
\def\cc@newpar[#1]{%
  \ifhmode\par\fi
  \vskip#1\relax
  \@afterheading
}
\cslet{\ccPrefix NewPar}\ccNewPar
```

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

`\TitleBreak`

```
\letcs\TitleBreak{\ccPrefix Break}
```

```
</headings>
```

Module 7

coco-notes.dtx

```
<*endnotes>
```

This file contains the code for foot- and endnote handling. It provides a switch between endnotes and footnotes as well as options to handle the resetting of footnote/endnote counters.

```
%%
%% module for CoCoTeX that handles footnote/endnote switching.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-notes}
   [2024/12/13 v0.5.0 le-tex coco notes module]
```

1 Internal Switches and Package Options

1.1 Package Switches

`\if@ccn@use@en` is an internal switch for endnotes (`\ccn@use@entrue`) or footnotes (`\ccn@use@enfalse`, default).

```
\newif\if@ccn@use@en \@ccn@use@enfalse
```

1.2 Package Options

The `endnotes` option causes all footnotes to be rendered as endnotes.

```
\ExplSyntaxOn
\keys_define:nn { cocotex/notes }
{
  endnotes .code:n = { \global\@ccn@use@entrue },
```

The option `ennotoc` prevents headings in the Notes section from creating entries in the Table of Contents.

```
  ennotoc .code:n = { \global\let\ccn@en@no@toc\relax },
```

The option `resetnotesperchapter` resets foot- and endnote counters at the start of each `chapter` level heading. If omitted (default) foot- or endnotes are numbered throughout the whole document

```
  resetnotesperchapter .code:n = { \global\let\ccn@reset@notes@per@chapter\relax },
```

The option `endnoteswithchapters` implies `endnotes` and causes chapter headings to be repeated in the printnotes chapter as sections.

```
  endnoteswithchapters .code:n =
  {
    \global\@ccn@use@entrue
    \global\let\ccn@en@with@chapters\relax
  },
```

The option `endnotelinks` is now defunct, because back-linking is necessary for tagging.

```
  endnotelinks .code:n = {}
}
\ProcessKeyOptions[cocotex/notes]
\ExplSyntaxOff
```

1.3 Hard Requirements

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

```
\RequirePackage{footnote}
```

2 Endnote Handling

`\if@enotesopen` is a switch from the `endnotes` package. but since the package is loaded only with the `endnotes` options set, we need to define the conditional, anyhow.

```
\newif\if@enotesopen
```

`\ccn@parindent` is the par indent used in the endnotes section. It defaults to the value of parindent at the very end of the LaTeX Preamble.

```
\AtBeginDocument{\edef\ccn@parindent{\the\parindent}}
```

`\enindent` is the left margin and hanging indent of the endnotes section.

```
\newdimen\enindent \enindent=2em\relax
```

If endnotes are activated via a Package option, we include the `endnotes` package.

```
\if@ccn@use@en
  \RequirePackage{endnotes}
```

`\ccn@use@TeX@heading` is a switch that defines itself when the CoCoTeX Headings module is loaded.

```
  \@ifpackageloaded{coco-headings}{\let\ccn@use@TeX@heading\relax}{}
```

`\@endnotemark` is re-defined when endnotes should back-reference. In this case, we insert a LaTeX `\label` for later referencing.

TODO
This macro should be patched, not re-defined!

```
\global\newcount\endnoteLinkCnt \global\endnoteLinkCnt\z@
\def\@endnotemark{%
  \leavevmode
  \ifhmode\edef\@x@sf{\the\spacefactor}\nobreak\fi
  \phantomsection%
  \label{endnote-\the\endnoteLinkCnt}%
  \ccaVstructStart{FootnoteMark}\ccaVstructStart{Lbl}%
  \hyperref[endnotetext-\the\endnoteLinkCnt]{\makeenmark}%
  \ccaVstructEnd{Lbl}\ccaVstructEnd{FootnoteMark}%
  \ifhmode\spacefactor\@x@sf\fi%
  \relax%
}
%\fi
```

`\footnote` is re-defined to be an alias of the `\endnote` macro.

```
\def\footnote{\cc@opt@empty\ccn@endnote}
\long\def\ccn@endnote[#1]#2{%
  \ccaStructStart{Footnote}%
  \ccaSaveCurStruct{abs@enote@\the\endnoteLinkCnt}%
  \def\@argi{#1}\ifx\@argi\@empty
    \endnote{#2}%
  \else
    \endnote[#1]{#2}%
  \fi
  \ccaStructEnd{Footnote}}
```

`\enotesize` holds the font size of the endnotes section.

```
\def\enotesize{\normalsize}%
```

`\enoteformat` is the format of an endnote. We create the label right at the start of the endnote text to prevent erroneous pointing to the next page.

```
\def\enoteformat{%
  \phantomsection%
  \label{endnotetext-\currentEndnote}%
  \noindent
  \leavevmode
  \hskip-\enindent\hb@xt@\enindent{%
    \ccaVstructStart{Lbl}%
    \hyperref[endnote-\currentEndnote]{\@theenmark}\hss%
    \ccaVstructEnd{Lbl}%
  }%
  \expandafter\parindent\ccn@parindent\relax\expandafter%
}%
```

`\enoteheading` is a macro that is expanded at the beginning of the endnotes section. Originally, it was intended to hold the endnote section's heading, we mis-use it to set the leftskip. Apparently, the intention is to re-define the macro style-wise if needed...

```
\gdef\enoteheading{%
  \leftskip\enindent
}%
```

`\printnotes` is the macro that eventually prints the endnote section in its stead.

```
  \def\printnotes{%
    \ifx\ccn@en@with@chapters\relax
      \ccn@end@enotes
    \fi
    \if@enotesopen
      \ifx\ccn@reset@notes@per@chapter\relax
        \global\c@endnote\z@%
      \fi
      \bgroup
      \parskip\z@
      \theendnotes
      \egroup
    \fi}
\else
```

\c@endnote is defined to ensure upward-compatibility.

```
  \newcount\c@endnote \c@endnote\z@
  \let\printnotes\relax
\fi
```

3 Processing Package Options

3.1 Endnotes With Chapters

\ccn@end@enotes stores the number of endnotes in a chapter in a generic macro \ccn@enotes@in@\the\realchap.

```
\def\ccn@end@enotes{\csxdef{ccn@enotes@in@\the\realchap}{\the\endnoteLinkCnt}}
```

\realchap is a counter that increases by one with each (coco-headings) chapter.

```
\newcount\realchap \realchap\z@
```

If endnotes are printed chapter-wise, we need to hook into the chapter heading level using cc/headings/chapter/print/before. There, we check if the last chapter did actually contain endnotes. If yes, we pass the chapter's Title and RunTitle components into the endnote temporary .ent file as a section heading.

```
\AtBeginDocument{%
  \AddToHook{cc/headings/chapter/print/before}{%
    \ifx\ccn@en@with@chapters\relax
      \ccn@end@enotes
      \global\advance\realchap\@ne
      \ifx\ccn@reset@notes@per@chapter\relax\global\c@endnote\z@\fi
      \def\ccn@par@number{\ccIfComp{TocNumber}{\ccUseComp{TocNumber}}{\ccUseComp{Number}}}%
      \def\ccn@par@title{\ccIfComp{TocTitle}{\ccUseComp{TocTitle}}{\ccUseComp{Title}}}%
      \def\ccn@par@runtitle{\ccIfComp{RunTitle}{\ccUseComp{RunTitle}}{\ccUseComp{Title}}}%
      \addtoendnotes{%
        \noexpand\ifnum\noexpand\csname ccn@enotes@in@\the\realchap\endcsname>\noexpand\csname
            ccn@enotes@in@\the\numexpr\the\realchap-\@ne\relax\endcsname\relax
          \noexpand\edef\noexpand\prev@leftkip{\noexpand\the\noexpand\leftskip}%
          \noexpand\leftskip\noexpand\z@
          \noexpand\begin{\ccPrefix Heading}\noexpand[notag\ifx\ccn@en@no@toc\relax, notoc\fi\
              noexpand]{section}%
            \noexpand\ccComponent{Number}{\ccn@par@number}%
            \noexpand\ccComponent{Title}{\ccn@par@title}%
          \noexpand\ccComponent{RunTitle}{\ccn@par@runtitle}%
          \noexpand\end{\ccPrefix Heading}%
        \noexpand\leftskip\noexpand\prev@leftkip\noexpand\relax%
        \noexpand\fi}%
    \else
```

3.2 Chapter-wise Resetting

If we don't use endnotes with chapters, we check if the `resetnotesperchapter` option is set and, if it is set, we instead set both counters for endnotes and footnotes to zero.

```
      \ifx\ccn@reset@notes@per@chapter\relax
        \global\c@footnote\z@
        \global\c@endnote\z@
      \fi
    \fi
    }%
  }
```

3.3 Back-Referencing Endnotes

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

```
\global\newif\if@haveenotes
\long\def\@endnotetext#1{%
  \global\@haveenotestrue
  \if@enotesopen \else \@openenotes \fi
  \immediate\write\@enotes{%
    \string\def\string\currentEndnote{\the\endnoteLinkCnt}%
    \noexpand\ccaVstructStart{FootnoteText}%
    \noexpand\expandafter\noexpand\ccaAddToStruct\noexpand\expandafter{\noexpand\csname
        abs@enote@\the\endnoteLinkCnt\noexpand\endcsname}%
    \@doanenote{\@theenmark}%
  }%
  \begingroup
     \def\next{#1}%
     \newlinechar='40
     \immediate\write\@enotes{\meaning\next}%
  \endgroup
  \immediate\write\@enotes{\noexpand\ccaAddID{auto}\noexpand\ccaVstructEnd{FootnoteText}\
      @endanenote}%
  \global\advance\endnoteLinkCnt\@ne%
}
```

3.4 Allow Non-Numerical Endnote Counters

`\@xendnote` is an override of endnote's macro of the same name to account for manual entnote counters that include non-numerical symbols.

```
\ifdefined\@xendnote
  \patchcmd\@xendnote
    {\c@endnote=#1\relax
     \unrestored@protected@xdef\@theenmark{\theendnote}}
    {\sbox\z@{\@tempcnta0#1\relax}%
     \ifdim\wd\z@>\z@\relax
```

```
      %\global\advance\c@endnote\@ne\relax
      \unrestored@protected@xdef\@theenmark{#1}%
    \else
      \c@endnote=#1\relax
      \unrestored@protected@xdef\@theenmark{\theendnote}%
    \fi}
   {}{\cc@patch@error{notes}\@xendnote}%
\fi
```

4 Adjusting Regular Footnotes

4.1 Allowing Multiple Paragraphs in Footnotes

First, we make a small adjustment to the `\fn@fntext` macro from the `footnote` package by making it `\long` and therefore allowing `\par` inside its argument.

```
\long\def\fn@fntext#1{%
  \ifx\ifmeasuring@\@@undefined%
    \expandafter\@secondoftwo\else\expandafter\@iden%
  \fi%
  {\ifmeasuring@\expandafter\@gobble\else\expandafter\@iden\fi}%
  {%
    \global\setbox\fn@notes\vbox{%
      \unvbox\fn@notes%
      \fn@startnote%
      \@makefntext{%
        \rule\z@\footnotesep%
        \ignorespaces%
        #1%
        \@finalstrut\strutbox%
      }%
      \fn@endnote%
    }%
  }%
}
```

4.2 Allowing Non-Numeric Footnote Counters

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

```
\def\fn@getmark@i#1[#2]{%
  \sbox\z@{\@tempcnta0#2\relax}%
  \ifdim\wd\z@>0\p@\relax
    \def\thempfn{#2}%
    \fn@getmark@iii%
  \else
    \csname c@\@mpfn\endcsname#2%
    \fn@getmark@ii%
  \fi
}
\def\fn@getmark@iii#1{%
  \unrestored@protected@xdef\@thefnmark{\thempfn}%
  \endgroup%
  #1%
}
```

And the same for plain LaTeX:

```
\long\def\@xfootnote[#1]#2{%
  \ccaStructStart{Footnote}%
  \begingroup
    \sbox\z@{\@tempcnta0#1\relax}%
    \ifdim\wd\z@>0\p@\relax
      \unrestored@protected@xdef\@thefnmark{#1}%
    \else
      \csname c@\@mpfn\endcsname #1\relax
      \unrestored@protected@xdef\@thefnmark{\thempfn}%
    \fi
  \endgroup
  \@footnotemark\@footnotetext{#2}%
  \ccaStructEnd{Footnote}%
}
```

5 Tagging Footnotes

Adding artifact tagging to the footnoterule:

```
\pretocmd\footnoterule{\ccaVstructStart[Document]{footnoterule}}{}{\cca@patch@error\footnoterule
    }
\apptocmd\footnoterule{\ccaVstructEnd{footnoterule}}{}{\cca@patch@error\footnoterule}
```

patching `\@footnotemark` to introduce the `<FootnoteMark/>` tag which will be mapped to the `<Reference/>` tag, later.

```
\pretocmd\@footnotemark{%
  \protected@xdef\@lt@fn@parent{\ccaGetCurStruct{idx}}%
  \ccaStructStart{FootnoteMark}%
}{}{\cca@patch@error\@footnotemark}
\apptocmd\@footnotemark{%
  \ccaStructEnd{FootnoteMark}%
}{}{\cca@patch@error\@footnotemark}
```

patching `\@makefnmark` for the `<Lbl/>` tag both in the text body and in the footnote insert.

```
\pretocmd\@makefnmark{%
  \ccaStructStart{Lbl}%\addAltText{\@thefnmark}
}{}{\cca@patch@error\@makefnmark}
\apptocmd\@makefnmark{%
  \ccaStructEnd{Lbl}%\addAltText{\@thefnmark}
}{}{\cca@patch@error\@makefnmark}
```

patching `\@makefntext` to introduce the `<FootnoteText/>` tag, which will be mapped to `<Note/>`, below.

```
\pretocmd\@makefntext{%
  \ccaStructStart{FootnoteText}%
  \ifx\@lt@fn@parent\@empty\relax\else\ccaAddToStruct{\@lt@fn@parent}\fi%
}{}{\cca@patch@error\@makefntext}
\apptocmd\@makefntext{%
  \ccaAddID{auto}\ccaStructEnd{FootnoteText}%
}{}{\cca@patch@error\@makefntext}
```

Finally, we add the <FootnoteMark/> and <FootnoteText/> PDF tags to the rolemap.

```
\ccaAddRolemap{Footnote}{Span}
\ccaAddRolemap{FootnoteMark}{Reference}
\ccaAddRolemap{FootnoteText}{\if@cc@pdf@two Aside\else Note\fi}
```

```
</endnotes>
```

Module 8
coco-script.dtx

```
<*script>
```

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

```
%% module for CoCoTeX that handles script switching.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% requires LuaLaTeX!
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-script}
   [2024/12/13 v0.5.0 CoCoTeX script module]
```

The argument of the `usescript` option is a list of script systems that are used in the document. It is used to determine the additional fonts that are to be loaded via the babel package.

```
\RequirePackage{coco-kernel}
\let\usescript\relax
\ExplSyntaxOn
\keys_define:nn { cocotex/script }
{
  usescript .code:n = { \gdef\usescript{#1} }
}
\ProcessKeyOptions[cocotex/script]
\ExplSyntaxOff
\RequirePackage[quiet]{fontspec}
\def\parse@script#1,#2,\relax{%
  \ccs@callback{#1}%
  \edef\@argii{#2}%
  \let\next\relax
  \ifx\@argii\@empty\else
    \def\next{\parse@script#2,\relax}%
  \fi\next}
\ifx\usescript\relax\else
  \def\ccs@callback#1{\expandafter\global\expandafter\let\csname use@script@#1\endcsname\@empty}
      %
  \expandafter\parse@script\usescript,,\relax
\fi
\ccPackageInfo{Script}{Info}{Fonts loaded: \meaning\usescript}
```

If babel's bidirectional feature is loaded without need, there might be errors, so we do some checks first:

```
\def\cc@bidi{}
\@tempswafalse
\ifx\usescript\relax\else
  \expandafter\ifx\csname use@script@arabic\endcsname\@empty\@tempswatrue
  \else\expandafter\ifx\csname use@script@hebrew\endcsname\@empty\@tempswatrue
    \else\expandafter\ifx\csname use@script@amharic\endcsname\@empty\@tempswatrue
      \else\expandafter\ifx\csname use@script@ethiop\endcsname\@empty\@tempswatrue
      \else\expandafter\ifx\csname use@script@syriac\endcsname\@empty\@tempswatrue
        \fi\fi\fi\fi\fi\fi
\if@tempswa\def\cc@bidi{,bidi=basic}\fi
\RequirePackage[silent\cc@bidi]{babel}
```

1 Fallback fonts

`\ccsTestFont` is used to test the currently active font family. For that, we compare the base name of `\f@family` with the (fully expanded) value given in the first argument.

`{#1}` is the comparison value that is testes against LaTeX's `\f@family`.
`{#2}` is the `true` branch, executed if the base name of `{#1}` matches against the base name `\f@family`.
`{#3}` is the `else` branch.

```
\def\ccsTestFont#1#2#3{\edef\@argi{#1}\expandafter\expandafter\expandafter\ccs@testfont\
    expandafter\f@family\expandafter.\expandafter\@nil\@argi.\@nil{#2}{#3}\@nil}
\def\ccs@testfont #1.#2\@nil#3.#4\@nil#5#6\@nil{\expandafter\ifnum\pdf@strcmp{#1}{#3}=\z@\relax
    #5\else#6\fi}
```

Comparison values of the three basic font families of LateX (`roman`, `sans-serif` and `monospace`) are defined after all packages are loaded. Macros for comparing Custom font families need to be defined manually.

```
\AddToHook{begindocument}{%
```

`\ccs@rmdefault` holds the default roman font

```
  \begingroup\rmfamily\xdef\ccs@rmdefault{\f@family}\endgroup%
```

`\ccs@sfdefault` holds the default sans-serif font

```
  \begingroup\sffamily\xdef\ccs@sfdefault{\f@family}\endgroup%
```

`\ccs@ttdefault` holds the default monospace font

```
  \begingroup\ttfamily\xdef\ccs@ttdefault{\f@family}\endgroup%
}
```

1.1 Default Fallback Font: Noto

The default fall backfont is the NotoSans Font Family

```
\newfontfamily\fallbackfont{NotoSerif-Regular.ttf}%
[BoldFont = NotoSerif-Bold.ttf,%
 ItalicFont = NotoSerif-Italic.ttf,%
 BoldItalicFont = NotoSerif-BoldItalic.ttf,%
 Path = ./fonts/Noto/Serif/,%
 WordSpace = 1.25]
\newfontfamily\sffallbackfont{NotoSans-Regular.ttf}%
[BoldFont = NotoSans-Bold.ttf,%
 ItalicFont = NotoSans-Italic.ttf,%
 BoldItalicFont = NotoSans-BoldItalic.ttf,%
 Path = ./fonts/Noto/Sans/,%
 WordSpace = 1.25]
```

```
\newfontfamily\ttfallbackfont{NotoSansMono-Regular.ttf}%
[BoldFont = NotoSansMono-Bold.ttf,%
 ItalicFont = NotoSansMono-Light.ttf,%
 BoldItalicFont = NotoSansMono-SemiBold.ttf,%
 Path = ./fonts/Noto/Mono/,%
 WordSpace = 1.25]
\DeclareTextFontCommand\textfallback{\fallbackfont}
\DeclareTextFontCommand\textsffallback{\sffallbackfont}
\DeclareTextFontCommand\texttfallback{\ttfallbackfont}
```

1.2 Emojis

A font and a text command for using plain, black emojis.

```
\newfontfamily\emojifont{NotoEmoji-Regular.ttf}%
[BoldFont = NotoEmoji-Bold.ttf,%
 Path = ./fonts/Noto/Emoji/]
\DeclareTextFontCommand\textemoji{\emojifont}
```

1.3 Support for medieval scripts and special characters

Warning: Junicode provides supports only for the `rm` font family!

```
\babelfont{mdv}[%
Path=fonts/Junicode/,%
ItalicFont = Junicode-Italic.ttf,%
BoldFont = Junicode-Bold.ttf,%
BoldItalicFont = Junicode-BoldItalic.ttf,%
]{Junicode.ttf}
\def\mdvfont#1{{\mdvfamily#1}}
```

1.4 International Phonetic Alphabet

Since the IPA character inventory seems to be included in the Noto fonts, we simply define the `\ccTextipa` as a context-sensitive alias.

`\ccTextipa` is a text command for the International Phonetic Alphabet. By default, the global Noto fallback font is used and sensitive to `rm`, `sf`, and `tt` contexts.

Warning! Noto has been chosen because it contains all IPA symbols, but there are some short-commings: in italic contexts, [a] and [] become indistinguishable for roman and hardly distinguishable in sans-serif contexts. Also, the bow used to bind vowels is misplaced in NotoSerif. We therefore provide an easy way to locally re-define alternative IPA fonts:

`\ccIpaFont` for roman contexts,

```
\let\ccIpaFont\fallbackfont
```

`\ccIpaSfFont` for sans-serif contexts, and

```
\let\ccIpaSfFont\sffallbackfont
```

`\ccIpaTtFont` for monospace contexts.

```
\let\ccIpaTtFont\ttfallbackfont
```

Those macros are used to select the appropriate IPA font inside the `\ccTextipa` text command:

```
\DeclareTextFontCommand\ccTextipa{%
  \ccsTestFont{\ccs@rmdefault}
    {\ccIpaFont}
    {\ccsTestFont{\ccs@sfdefault}
      {\ccIpaSfFont}
      {\ccIpaTtFont}}}%
```

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}`

```
\def\ccDeclareBabelFont{\cc@opt@empty\ccs@declare@babel@font}%
\def\ccs@declare@babel@font[#1]#2#3#4{%
  \expandafter\ifx\csname use@script@#2\endcsname\@empty
    \babelprovide[#1]{#2}%
    \message{^^J [coco-script Loaded Script: #2]^^J}%
    %%
    \expandafter\gdef\csname ccs@babel@rm@font@#2\endcsname{#3}%
    \expandafter\gdef\csname ccs@babel@sf@font@#2\endcsname{#4}%
    \if!#2!\else
      \def\ccs@tempa{\babelfont[#2]{rm}}%
      \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@rm@font@#2\endcsname
    \fi
    \if!#3!\else
      \def\ccs@tempa{\babelfont[#2]{sf}}%
      \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@sf@font@#2\endcsname
    \fi
  \fi
}
```

Top level macro to declare a font alias.

`#1`	font family alias
`#2`	font family fallback

```
\def\ccBabelAlias#1#2{%
  \ifx\usescript\relax\else
    \def\ccs@callback##1{%
      \expandafter\ifx\csname ccs@no@fallback@##1\endcsname\relax
        \expandafter\ifx\csname ccs@babel@#2@font@##1\endcsname\relax
          \PackageError
            {coco-script.sty}
            {\expandafter\string\csname #2family\endcsname\space for Language `##1' was not
                declared!}
            {You attempted to declare an alias towards a font family that has not been declared
                for the language `##1', yet.}%
        \else
          \def\ccs@tempa{\babelfont[##1]{#1}}%
        \expandafter\expandafter\expandafter\ccs@tempa\csname ccs@babel@#2@font@##1\endcsname
      \fi
    \else
      \PackageInfo{coco-script.sty}{^^J\space\space\space\space No fallback for `##1';^^J\space
          \space\space\space Skipping font family `#1'->`#2'}%
    \fi}%
  \expandafter\parse@script\usescript,,\relax
\fi}
```

3 Predefined script systems

3.1 Support for Armenian script

```
\ccDeclareBabelFont{armenian}{[%
    Path=./fonts/Noto/Armenian/,
    BoldFont = NotoSerifArmenian-Bold.ttf,%
    WordSpace = 1.25]{NotoSerifArmenian-Regular.ttf}}
  {[%
    Path=./fonts/Noto/Armenian/,
    BoldFont = NotoSansArmenian-Bold.ttf,%
    WordSpace = 1.25]{NotoSansArmenian-Regular.ttf}%
  }
```

Legacy and backwards compatibility:

```
\def\armenian{\foreignlanuage{armenian}}
```

3.2 Support for Chinese script

```
\ccDeclareBabelFont{chinese}{[%
    Path=./fonts/Noto/Chinese/,
    BoldFont = NotoSerifSC-Bold.otf,%
    WordSpace = 1.25]{NotoSerifSC-Regular.otf}}
  {[%
    Path=./fonts/Noto/Chinese/,
    BoldFont = NotoSansSC-Bold.otf,%
    WordSpace = 1.25]{NotoSansSC-Regular.otf}%
  }
```

3.3 Support for Japanese script

```
\ccDeclareBabelFont{japanese}{[%
    Path=./fonts/Noto/Japanese/,
    BoldFont = NotoSerifJP-Bold.otf,%
    WordSpace = 1.25]{NotoSerifJP-Regular.otf}
  }{[%
    Path=./fonts/Noto/Japanese/,
    BoldFont = NotoSansJP-Bold.otf,%
    WordSpace = 1.25]{NotoSansJP-Regular.otf}
  }
```

3.4 Support for Korean script

```
\ccDeclareBabelFont{korean}{[%
  BoldFont = NotoSerifKR-Bold.otf,%
  ItalicFont = NotoSerifKR-Regular.otf,%
  BoldItalicFont = NotoSerifKR-Medium.otf,%
  Path=./fonts/Noto/Korean/,%
  Script=CJK%
  ]{NotoSerifKR-Regular.otf}}
{[%
  BoldFont = NotoSansKR-Bold.otf,%
  ItalicFont = NotoSansKR-Regular.otf,%
  BoldItalicFont = NotoSansKR-Medium.otf,%
  Path=./fonts/Noto/Korean/,%
  Script=CJK%
  ]{NotoSansKR-Regular.otf}%
}
```

3.5 Support for Hebrew script

```
\ccDeclareBabelFont{hebrew}{[%
    Renderer=Harfbuzz,%
    Scale=MatchUppercase,%
    Path=./fonts/Noto/Hebrew/,%
    Ligatures=TeX,%
    Script=Hebrew,%
    BoldFont = NotoSerifHebrew-Bold.ttf]{NotoSerifHebrew-Regular.ttf}%
}{[%
    Renderer=Harfbuzz,%
    Scale=MatchUppercase,%
    Path=./fonts/Noto/Hebrew/,%
    Ligatures=TeX,%
    Script=Hebrew,%
    BoldFont = NotoSansHebrew-Bold.ttf]{NotoSansHebrew-Regular.ttf}%
}
```

3.6 Support for Arabic script

```
\ccDeclareBabelFont{arabic}{[%
    BoldFont = NotoNaskhArabic-Bold.ttf,%
    Path = ./fonts/Noto/Arabic/,%
    Script=Arabic%
    ]{NotoNaskhArabic-Regular.ttf}}
  {[%
    BoldFont = NotoSansArabic-Bold.ttf,%
    Path = ./fonts/Noto/Arabic/,%
    Script=Arabic%
    ]{NotoSansArabic-Regular.ttf}%
  }
```

3.7 Support for Greek script

```
\ccDeclareBabelFont{greek}{[%
    BoldFont = NotoSerif-Bold.ttf,%
    ItalicFont = NotoSerif-Italic.ttf,%
    BoldItalicFont = NotoSerif-BoldItalic.ttf,%
    Path = ./fonts/Noto/Serif/,%
    Script=Greek,%
    WordSpace = 1.25
    ]{NotoSerif-Regular.ttf}}
  {[BoldFont = NotoSans-Bold.ttf,%
    ItalicFont = NotoSans-Italic.ttf,%
    BoldItalicFont = NotoSans-BoldItalic.ttf,%
    Path = ./fonts/Noto/Sans/,%
    Script=Greek,%
    WordSpace = 1.25%
    ]{NotoSans-Regular.ttf}%
  }
```

3.8 Support for Ethiopian/Amharic script

```
\ccDeclareBabelFont{ethiop}{[%
    BoldFont = NotoSerifEthiopic-Bold.ttf,%
    ItalicFont = NotoSerifEthiopic-Regular.ttf,%
    BoldItalicFont = NotoSerifEthiopic-Bold.ttf,%
    Path = ./fonts/Noto/Ethiop/,%
    WordSpace = 1.25
    ]{NotoSerifEthiopic-Regular.ttf}}
  {[BoldFont = NotoSansEthiopic-Bold.ttf,%
    ItalicFont = NotoSansEthiopic-Regular.ttf,%
    BoldItalicFont = NotoSansEthiopic-Bold.ttf,%
    Path = ./fonts/Noto/Ethiop/,%
    WordSpace = 1.25%
    ]{NotoSansEthiopic-Regular.ttf}%
  }
\ccDeclareBabelFont{amharic}{[%
    BoldFont = NotoSerifEthiopic-Bold.ttf,%
    ItalicFont = NotoSerifEthiopic-Regular.ttf,%
    BoldItalicFont = NotoSerifEthiopic-Bold.ttf,%
    Path = ./fonts/Noto/Ethiop/,%
    WordSpace = 1.25
    ]{NotoSerifEthiopic-Regular.ttf}}
  {[BoldFont = NotoSansEthiopic-Bold.ttf,%
    ItalicFont = NotoSansEthiopic-Regular.ttf,%
    BoldItalicFont = NotoSansEthiopic-Bold.ttf,%
    Path = ./fonts/Noto/Ethiop/,%
    WordSpace = 1.25%
    ]{NotoSansEthiopic-Regular.ttf}%
  }
```

3.9 Support for Georgian script

```
\ccDeclareBabelFont{georgian}{%
  [%
    Path = ./fonts/Noto/Georgian/,%
    BoldFont = NotoSerifGeorgian-Bold.ttf,%
    ItalicFont = NotoSerifGeorgian-Regular.ttf,%
    BoldItalicFont = NotoSerifGeorgian-Bold.ttf,%
    Scale=0.85%
  ]{NotoSerifGeorgian-Regular.ttf}%
}{%
  [%
    Path = ./fonts/Noto/Georgian/,%
    BoldFont = NotoSerifGeorgian-Bold.ttf,%
    ItalicFont = NotoSerifGeorgian-Regular.ttf,%
    BoldItalicFont = NotoSerifGeorgian-Bold.ttf,%
    Scale=0.85%
  ]{NotoSerifGeorgian-Regular.ttf}%
}
```

3.10 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.

```
\expandafter\ifx\csname use@script@syriac\endcsname\@empty%
\RequirePackage{filecontents}
\begin{filecontents*}{babel-syriac.ini}
[identification]
charset = utf8
version = 0.1
date = 2019-08-25
name.local = ??????????
name.english = Classical Syriac
name.babel = classicalsyriac
tag.bcp47 = syc
tag.opentype = SYR
script.name = Syriac
script.tag.bcp47 = Syrc
script.tag.opentype = syrc
level = 1
encodings =
derivate = no
[captions]
[date.gregorian]
[date.islamic]
[time.gregorian]
[typography]
[characters]
[numbers]
[counters]
\end{filecontents*}
\fi
```

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

```
\ccDeclareBabelFont[import=syriac]{syriac}{[%
    BoldFont = NotoSansSyriac-Black.ttf,%
    ItalicFont = NotoSansSyriac-Regular.ttf,%
    BoldItalicFont = NotoSansSyriac-Black.ttf,%
    Path = ./fonts/Noto/Syriac/,%
    WordSpace = 1.25
    ]{NotoSansSyriac-Regular.ttf}}
  {[BoldFont = NotoSansSyriac-Black.ttf,%
    ItalicFont = NotoSansSyriac-Regular.ttf,%
    BoldItalicFont = NotoSansSyriac-Black.ttf,%
    Path = ./fonts/Noto/Syriac/,%
    WordSpace = 1.25%
    ]{NotoSansSyriac-Regular.ttf}%
  }
```

```
</script>
```

Module 9
coco-title.dtx

```
<*title>
```

This file provides macros and facilities for title pages.

```
%%
%% module for CoCoTeX for maketitle.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-title}
   [2024/12/13 v0.5.0 CoCoTeX title module]
\RequirePackage{coco-meta}
```

1 Top-Level Interface

`titlepage` is the main Container for the whole document's meta data.

```
\ccDeclareContainer{titlepage}{%
  \ccInherit {Components,Properties}{CommonMeta}%
  \ifarticle\ccInherit{Components}{article-meta}\fi
  \ccDeclareType{Components}{%
    \cct@simple@comps
    \cct@fundings@comp
    \cct@role@handlers{author}{Author}%
    \cct@declare@role{editor}{Editor}%
    \cct@declare@role{series-editor}{SeriesEditor}%
  }%
  \ccDeclareType{Properties}{}%
  \ccDeclareEnv[Meta]{\cctmeta}{\endcctmeta}%
}
```

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

```
\def\cct@declare@role#1#2{%
  \ccDeclareRole[#1]{#2}%
  \cct@role@handlers{#1}{#2}%
}
```

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

```
\def\cct@role@handlers#1#2{%
```

```
  \ccAddToRole{#2}{%
    \ccDeclareCountedComponent{Bio}%
    \ccDeclareCountedComponent{Biography}}%
  \ccDeclareGroupHandler{#2}{%
    \ccIfComp{Biography}{}{\ccIfComp{Bio}{\ccComponent{Biography}{\ccUseProperty{#1-biography-
        format}}}{}}%
  }%
  \ccDeclareRoleBlock[apply]{#2}{BioBlock}{#1-bio-block-format}%
}
```

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

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

`\cctmeta` is the code executed at the beginning of the `\ccPrefix Meta` Container

```
\def\cctmeta{\cc@opt@empty\cct@meta}
\def\cct@meta[#1]{%
  \UseHook{env/meta/begin}%
  \ccEvalAttributes[titlepage]{#1}%
  \ccEvalType{Components}%
  \let\and\relax
}
```

`\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.

```
\def\ccAddTitleRole#1#2{%
  \ccAddToType{Components}{titlepage}{\cct@declare@role{#1}{#2}}%
  \ccAddTitleEval{\cct@eds@eval{#2}}%
}
```

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

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

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

```
\def\cct@add@eval{}
```

`\endcctmeta` is the code executed at the end of the `Meta` Container

```
\def\endcctmeta{%
  \ccSetContainer{titlepage}%
  \ccEvalType{Properties}%
  \cct@maketitle
  \ccm@role@eval{Author}%
  \ccApplyCollection{Affil}{affil-block-item-format}{AffilBlock}%
  \cct@eds@eval{Editor}%
  \cct@eds@eval{SeriesEditor}%
  \ccm@generic@eval
  \cct@fundings@eval
  \cct@add@eval
  \cc@if@preamble\cct@set@pdfmeta\relax
```

Now, we expand the `cct/document/meta`.

```
  \UseHook{cct/document/meta}%
  \let\cc@cur@cont\@empty
}
```

2 Procesing 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.

```
\def\cct@set@pdfmeta{%
```

`\cct@write@pdf@meta@string` handles meta data that are stored as plain *strings* in the XMP file.

`{#1}` is the hyperref name for the meta datum
`{#2}` is the ltpdfa name for the meta datum
`{#3}` is the fallback value.

The fallback value is only chosen if either no XMP file exists, or if the XMP file does not contain the required data field.

In any case, the string is sanitized through `\ccSanitizeStr`.

```
  \def\cct@write@pdf@meta@string##1##2##3{%
    \let\cct@cur@data\@empty
    \ccIfAlly
      {\ccSanitizeStr\cct@cur@data{\directlua{tex.print(cocotex.ally.meta.##2)}}%
       \ifx\cct@cur@data\@empty
         \ccSanitizeStr\cct@cur@data{##3}%
       \fi
       \ifx\cct@cur@data\@empty
         \ccDebugMsg[pdfmeta]{##2 is empty (3: ##3); doing nothing}%
       \else
         \ccDebugMsg[pdfmeta]{Writing \string\setDocinfo[utf-8]{##2}{\expandafter\strip@prefix\
             meaning\cct@cur@data}}%
         \edef\x{\noexpand\ccaSetDocinfo[utf-8]{##2}}\expandafter\x\expandafter{\cct@cur@data}%
       \fi}
     {\cct@write@hr@infodict{##1}{##3}}}%
```

`\cct@write@pdf@meta@list` handles meta data that are represented as *lists* in the XMP file.

`{#1}` is the hyperref name for the meta datum
`{#2}` is the ltpdfa name for the meta datum
`{#3}` is the fallback value.
`{#4}` is the handler for the list. This is a macro that takes one argument. Each item of the list will be passed to that macro as argument.
`{#5}` is the separator between the items

```
  \def\cct@write@pdf@meta@list##1##2##3##4##5{%
    \let\cct@cur@data\@empty
    \ccIfAlly
      {\protected@edef\cct@cur@data{\directlua{tex.print(cocotex.ally.meta.##2)}}%
       \ifx\cct@cur@data\@empty
         \protected@edef\cct@cur@data{##3}%
         \ifx\cct@cur@data\@empty\else
           \edef\x{\noexpand\cct@split@pdf@meta{\noexpand##4}{##5}}\x%
      \fi
    \else
      \cct@split@pdf@meta{##4}{\and}%
    \fi}
   {\cct@write@hr@infodict{##1}{##3}}}%
```

`\cct@split@pdf@meta` is a helper function to split the list-form meta datum stored `\cct@cur@data` in into its items.

`{#1}` is a CS token that takes one argument. Each item of the list is recursively passed to that CS token
`{#2}` is the separator between the items

```
  \def\cct@split@pdf@meta##1##2{%
    \def\@cct@split@pdf@meta####1##2####2\@nil####3{%
      \if\relax\detokenize{####1}\relax\else
        ####3{####1}%
        \if!####2!\else
          \edef\@argi{##2}%
          \edef\@argii{####2}%
          \ifx\@argi\@argii\else
            \@cct@split@pdf@meta####2\@nil{####3}%
          \fi
        \fi
      \fi
    }%
    \expandafter\@cct@split@pdf@meta\cct@cur@data##2\@nil{##1}%
  }%
```

`\cct@write@hr@infodict` writes the PDF info dictionary entry via `hyperref` with the key `{#1}` and the value `{#2}`.

```
  \def\cct@write@hr@infodict##1##2{%
    \protected@edef\x{\noexpand\hypersetup{##1={\expandonce{##2}}}}\x
  }%
```

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

```
  \cct@title@insert@xmp
  \cct@title@process@bkc
  \cct@title@process@bkt
  \cct@title@process@bkk
  \cct@title@process@bka
  \cct@title@process@bkl
}
```

2.1 Processing of the Document's Main Language

`\cct@title@process@bkl` writes the document's main language into the Info dictionary if no accessibility features are active. If it *is* active, the document language is handled by ltpdfa. The language tag is extracted via babel's ini configuration file or the main document language.

```
\def\cct@title@process@bkl{%
  \ccUnlessAlly{%
    \edef\cct@lang@id{\localeinfo*{name.english}}%tag.bcp47
    \cct@write@hr@infodict{pdflang}{\cct@lang@id}%}%
  }%
}
```

2.2 Processing of the Document’s Title

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

```
\def\cct@title@process@bkt{%
  \cslet{\ccPrefix Break}\space
  \protected@xdef\@title{\ccUseProperty{doc-book-title}}%
  \cct@write@pdf@meta@string{pdftitle}{Title}{\@title}%
  \ccpgdefFromProperty{RunBookTitle}{run-book-title}%
}
```

2.3 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.

```
\def\cct@title@process@bka{%
  \@tempswatrue
  \begingroup
    \ccGobble
    \renewcommand\foreignlanguage[2]{{##2}}%
    \ccIfComp{AuthorPDFInfo}
      {\ccpgdefFromComp{RunBookName}{AuthorPDFInfo}}
      {\ccIfComp{EditorPDFInfo}
         {\ccpgdefFromComp{RunBookName}{EditorPDFInfo}}
         {\ccIfComp{AuthorNameList}
           {\ccpgdefFromComp{RunBookName}{AuthorNameList}}
           {\ccIfComp{EditorNameList}
             {\ccpgdefFromComp{RunBookName}{EditorNameList}}
             {\ifnum\ccAuthorCnt>\z@
                \@setpar{\@@par}%
                \ccpxdefFromCountedComp{RunBookName}{Author}{author-list-pdfinfo-format}%
              \else
                \ifnum\ccEditorCnt>\z@
                  \ccpxdefFromCountedComp{RunBookName}{Editor}{editor-list-pdfinfo-format}%
                \else
                  \ccPackageWarning{transcript-title}{Meta Data}{No author or editor given!}%
                  \@tempswafalse
                \fi
              \fi
            }}}}%
    \if@tempswa
      \ccSanitizeStr\@author{\csname\ccPrefix RunBookName\endcsname}%
      \cct@write@pdf@meta@list{pdfauthor}{Author}{\csname\ccPrefix RunBookName\endcsname}{\
          ccaAddAuthor}{\and}%
    \fi
  \endgroup
}
```

2.4 Processing of the PDF’s Creator, Producer, and Keywords Meta Data

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

```
\def\cct@title@process@bkc{%
  \cct@write@pdf@meta@string{pdfcreator}{Creator}{\ccWhenComp{PDFCreator}{\ccUseComp{PDFCreator
      }}}%
  \cct@write@pdf@meta@string{pdfproducer}{Producer}{\ccWhenComp{PDFProducer}{\ccUseComp{
      PDFProducer}}}%
}
```

`\cct@title@process@bkk` processed the metadata for the keywords.

```
\def\cct@title@process@bkk{%
  \cct@write@pdf@meta@list{pdfkeywords}{Keywords}{\ccWhenComp{Keywords}{\ccUseComp{Keywords}}}{\
      ccaAddKeyword}{\ccUseProperty{keywords-sep}}%
}
```

2.5 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.

```
\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.

```
\def\cct@title@insert@xmp@direct{%
  \edef\include@xmp{\noexpand\@include@xmp{\ccUseComp{XmpFile}.xmp}}%
  \def\@include@xmp##1{\IfFileExists{##1}{\@@include@xmp{##1}}{}}%
  \def\@@include@xmp##1{%
    \begingroup
      \immediate\pdfobj stream attr {/Type /Metadata /Subtype /XML}
      file{##1}
      \pdfcatalog{/Metadata \the\pdflastobj\space 0 R}
    \endgroup}%
  \include@xmp
}
```

`\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.

```
\def\cct@title@insert@xmp@ltpdfa{%
  \edef\cca@xmp@file@name{\ccUseComponentFrom{titlepage}{XmpFile}.xmp}%
  \IfFileExists{\cca@xmp@file@name}
    {\ccaAddToConfig{metadata}{xmpfile=\cca@xmp@file@name}%
     \directlua{ally.meta:extract()}}
    {\ccPackageWarning{A11y}{File}{%
\cca@xmp@file@name\space not found.^^J
Note that the ltpdfa package will create one^^J
from the Components given in the Meta Container.}}}
```

3 Intermediate Level Interfaces

`cct/maketitle/before` is expanded right before the titlepage is printed.

```
\NewHook{cct/maketitle/before}
```

`cct/maketitle/after` is expanded at the end of the titlepage.

```
\NewHook{cct/maketitle/after}
```

`cct/document/meta` is expanded at the very end of the `Meta` Container.

```
\NewHook{cct/document/meta}
```

`env/meta/begin` is used to add code to be executed at the very beginning of the Meta Container's main environment, before the Attribtues are evaluated.

```
\NewHook{env/meta/begin}
```

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

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

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

```
\def\cct@journal@titlepage{%
  \ccUseProperty{before-titlepage}%
  \ccWhenComp{Cover}{\ccUseProperty{coverpage}}%Cover ist kein Bild, wird von uns gebaut
  \ccUseProperty{before-titlepage-roman}%
  \ccUseProperty{titlepage-roman}%
  \ccUseProperty{after-titlepage}%
}
```

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

```
\def\cct@book@titlepage{%
  \ccUseProperty{before-titlepage}%
  \ccWhenComp{Cover}{\ccUseProperty{coverpage}}%
  \ccUseProperty{before-titlepage-roman}%
  \ccUseProperty{titlepage-roman}%
  \ccUseProperty{after-titlepage}%
}
```

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

```
\def\cct@maketitle{%
  \expandafter\gdef\csname\ccPrefix Maketitle\endcsname{%
    \let\cc@cnt@grp\@empty
```

Here, we expand the `cct/maketitle/before`.

```
    \UseHook{cct/maketitle/before}%
    \bgroup
      \ccSetContainer{titlepage}%
      \ccEvalType{Properties}%
      \ifarticle
        \cct@article@titlepage
      \else
        \ifjournal
          \cct@journal@titlepage
        \else
```

```
          \cct@book@titlepage
        \fi
      \fi
    \egroup
    \UseHook{cct/maketitle/after}%
  }%
}
```

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

```
\def\cct@fundings@comp{%
  \ccDeclareComponent{FundingBlock}{\expandafter\global}{}%
  \ccDeclareComponentGroup{Funding}{%
    \ccDeclareCountedComponent{FundName}%
    \ccDeclareCountedComponent{FundLogo}%
    \ccDeclareCountedComponent{FundID}%
  }{}%
}
```

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

```
\def\cct@fundings@eval{{%
    \def\cc@cur@cont{titlepage}%
    \ccComposeCollection{Funding}{fund-format}{FundingBlock}%
    \UseHook{cc/titlepage/funding}%
}}
\NewHook{cc/titlepage/funding}
```

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

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

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

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

3.2 Simple Component Declarations

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

```
\def\cct@simple@comps{%
```

General Information

`Cover` holds the path(!) to the cover image (without `\includegraphics`!)

```
\ccDeclareGlobalComponent{Cover}%
```

`Dedication` is a dedication.

```
\ccDeclareGlobalComponent{Dedication}%
```

`Acknowledgements` self explanatory.

```
\ccDeclareGlobalComponent{Acknowledgements}%
```

`Statement` additional publication statement

```
\ccDeclareGlobalComponent{Statement}%
```

`Editorial` generic statement by the editors of a periodical or collection.

```
\ccDeclareGlobalComponent{Editorial}%
```

Titles and Names

`Title` is the document's (printed) main title.

```
\ccDeclareGlobalComponent{Title}%
```

`ShortTitleOR` is a shortened version of the document title. If set, this is the title that is written into the PDF meta data (unless DocTitle is also set) and stored as running title (unless RunTitle is also set). If the Component is not used, the Title component is used, instead.

```
\ccDeclareGlobalComponent{ShortTitle}%
```

`DocTitleOR` is an override for the title that is written into the PDF meta data (unless an XMP file is used). This Component's value should only contain alphanumeric characters, ideally from the ASCII code block, but must not contain any LaTeX markup.

```
\ccDeclareGlobalComponent{DocTitle}%
```

`RunTitleOR` is an override for the title that is used as running title for page headers. It should contain only robust LaTeX markup.

```
\ccDeclareGlobalComponent{RunTitle}%
```

`AltTitle` is an alternative title for bastard title pages, etc. It is not used by CoCoTeX, but some publisher styles may need it.

```
\ccDeclareGlobalComponent{AltTitle}%
```

`Subtitle` is the document's subtitle.

```
\ccDeclareGlobalComponent{Subtitle}%
```

TitleNote additional printed information tied to the document's title.

```
\ccDeclareGlobalComponent{TitleNote}%
```

RunNamesOR is an override for the document author's names, which is intended to be used in running page headers. If not set, the (calculated) AuthorNameList or EditorNameList is used, instead.

```
\ccDeclareGlobalComponent{RunNames}%
```

AltNames is an alternative data field for additional names.

```
\ccDeclareGlobalComponent{AltNames}%
```

Series

Series is the series title.

```
\ccDeclareGlobalComponent{Series}%
```

SubSeries is the subtitle for the series.

```
\ccDeclareGlobalComponent{SubSeries}%
```

SeriesNote are additional notes concerning the series.

```
\ccDeclareGlobalComponent{SeriesNote}%
```

Volume is the volume of the document within a series.

```
\ccDeclareGlobalComponent{Volume}%
```

Number is the number of the document within the series.

```
\ccDeclareGlobalComponent{Number}%
```

EditorNameListCC is the Collection Component for the Editor's names.

```
\ccDeclareGlobalComponent{EditorNameList}%
```

SeriesEditorNameListCC is the Colection Component for the Series-Editor's names.

```
\ccDeclareGlobalComponent{SeriesEditorNameList}%
```

Publisher Information

Publisher is the publisher name

```
\ccDeclareGlobalComponent{Publisher}%
```

PubDivision is the publisher division.

```
\ccDeclareGlobalComponent{PubDivision}%
```

PubDivInfo holds additional information about the publisher division.

```
\ccDeclareGlobalComponent{PubDivInfo}%
```

PubPlace holds the place of publication or the publisher's address.

```
\ccDeclareGlobalComponent{PubPlace}%
```

PubLogo holds the publisher logo. Depending on the publisher style, this may be just a path, or a complete \includegraphics expression.

```
\ccDeclareGlobalComponent{PubLogo}%
```

PubNote additional generic notes about the publisher.

```
\ccDeclareGlobalComponent{PubNote}%
```

PubWeb holds the url or email contact address of a publisher.

```
\ccDeclareGlobalComponent{PubWeb}%
```

Publication Meta

XmpFile is the basename of the XMP meta data file without the .xmp file ending (which is added automatically). The default value is \jobname.

```
\ccDeclareGlobalComponent[\jobname]{XmpFile}%
```

PDFCreator is the tool with which the original document was created (for *xerif*, this is usually M$ Word).

```
\ccDeclareGlobalComponent{PDFCreator}%
```

PDFProducer is the tool with which the PDF was created. Defaults to ''le-tex xerif with CoCoTeX v(CoCoTeX version)''

```
\ccDeclareGlobalComponent[le-tex xerif with CoCoTeX v.v0.5.0]{PDFProducer}%
```

Year the year of the publication

```
\ccDeclareGlobalComponent{Year}%
```

Date the date of writing/finishing. Defaults to \today.

```
\ccDeclareGlobalComponent[\today]{Date}%
```

Edition the edition of the publication.

```
\ccDeclareGlobalComponent{Edition}%
```

EditionNote additional notes about the particular edition.

```
\ccDeclareGlobalComponent{EditionNote}%
```

`ISBNPreText` text added before the ISBN block

```
\ccDeclareGlobalComponent{ISBNPreText}%
```

`ISBN` the publication's international standard book number.

```
\ccDeclareGlobalComponent{ISBN}%
```

`ISSN` the publication's international standard serial number for periodicals.

```
\ccDeclareGlobalComponent{ISSN}%
```

`EISSN` additional ISSN for electronic publications

```
\ccDeclareGlobalComponent{EISSN}%
```

`EpubPreText` additional text between ISBN and eISBN

```
\ccDeclareGlobalComponent{EpubPreText}%
```

`EISBN` ISBN for electronic publications.

```
\ccDeclareGlobalComponent{EISBN}%
```

`EpubISBN` ISBN for EPUBs.

```
\ccDeclareGlobalComponent{EpubISBN}%
```

`ElibPDF` additional serial number for electronic libraries.

```
\ccDeclareGlobalComponent{ElibPDF}%
```

`BiblISSN` additional ISSN for special libraries.

```
\ccDeclareGlobalComponent{BiblISSN}%
```

`BibleISSN` additional electronic ISSN for special libraries.

```
\ccDeclareGlobalComponent{BibleISSN}%
```

Funding

`FundingPreText` additional text before the funding list.

```
\ccDeclareGlobalComponent{FundingPreText}%
```

`FundingPostText` additional text after the funding list.

```
\ccDeclareGlobalComponent{FundingPostText}%
```

Imprint Meta Data

`Biblio` bibliographic information block.

```
\ccDeclareGlobalComponent{Biblio}%
```

`BiblioTitle` the title of the bibliographic information block

```
\ccDeclareGlobalComponent{BiblioTitle}%
```

`Print` the name and address of the printing company.

```
\ccDeclareGlobalComponent{Print}%
```

`PrintNote` additional information about the printing process.

```
\ccDeclareGlobalComponent{PrintNote}%
```

`Lectorate` name and address of the lectorate

```
\ccDeclareGlobalComponent{Lectorate}%
```

`Translator` name of the document's translator

```
\ccDeclareGlobalComponent{Translator}%
```

`CoverConcept` concept creator of the front page cover

```
\ccDeclareGlobalComponent{CoverConcept}%
```

`CoverDesign` designer of the front page cover.

```
\ccDeclareGlobalComponent{CoverDesign}%
```

`Component` creator of the front page cover image

```
\ccDeclareGlobalComponent{CoverImage}%
```

`Typesetter` name and address of the typesetter

```
\ccDeclareGlobalComponent{Typesetter}%
```

`QA` name of the person(s) responible for quality assurance.

```
\ccDeclareGlobalComponent{QA}%
```

`UsedFont` information about the fonts used throughout the document

```
\ccDeclareGlobalComponent{UsedFont}%
```

`Conversion` name of the person(s) responsible for data conversion

```
\ccDeclareGlobalComponent{Conversion}%
```

`EnvDisclaimer` environmantal disclaimer, used paper, etc.

```
\ccDeclareGlobalComponent{EnvDisclaimer}%
```

`Advertise` advertisements.

```
\ccDeclareGlobalComponent{Advertise}%
```

Licencing

`LicenceText` License Description

```
\ccDeclareGlobalComponent{LicenceText}%
```

`LicenseLogo` the path(!) to the licence logo. `\includegraphics` is added automatically.

```
\ccDeclareGlobalComponent{LicenceLogo}%
```

`LicenceLink` URL to the license.

```
\ccDeclareGlobalComponent{LicenceLink}%
```

`LicenceName` the plain name of the license.

```
\ccDeclareGlobalComponent{LicenceName}%
```

`CopyrightDisclaimer` self explanatory...

```
\ccDeclareGlobalComponent{CopyrightDisclaimer}%
```

Journal-specific Meta Data

`JournalName` Full name of the journal.

```
\ccDeclareGlobalComponent{JournalName}%
```

`JournalAbbrev` short name of the journal.

```
\ccDeclareGlobalComponent{JournalAbbrev}%
```

`Issue` of the journal.

```
\ccDeclareGlobalComponent{Issue}%
```

`PubCycle` Publication cycle

```
\ccDeclareGlobalComponent{PubCycle}%
```

`Prices` of the journal issues or subscription models

```
\ccDeclareGlobalComponent{Prices}%
```

`MemberList` in case of publishing organizations, this Component may hold a list of members.

```
\ccDeclareGlobalComponent{MemberList}%
```

Startpage is the start page of the Journal

```
  \ccDeclareGlobalComponent{Startpage}%
```

Generic additional information

AddNoteI additional information for the first title page.

```
  \ccDeclareGlobalComponent{AddNoteI}%
```

AddNoteII additional information for the second title page.

```
  \ccDeclareGlobalComponent{AddNoteII}%
```

AddNoteIII additional information for the third title page.

```
  \ccDeclareGlobalComponent{AddNoteIII}%
```

AddNoteIV additional information for the fourth title page.

```
  \ccDeclareGlobalComponent{AddNoteIV}%
```

```
}
```

4 Default Settings

```
\ccAddToProperties{titlepage}{%
```

article-title <any> is the title of a single article. It defaults to what standard LaTeX does with \maketitle in the article class without the titlepage class option and uses the Title, AuthorNameList, and Date Components.

```
  \ccSetProperty{article-title}{%
    \newpage
    \null
    \vskip 2em
    \begin{center}%
      \ccaStructStart{Titlepage}%
      \let \footnote \thanks
      {\LARGE\ccaStructStart{Title}\ccaReplaceStruct{\cca@id@document@dummy}\ccUseComp{Title}\
          ccaStructEnd{Title}\par}%
      \vskip 1.5em%
      {\large
        \lineskip .5em%
        \ccaStructStart{Authors}\ccUseComp{AuthorNameList}\ccaStructEnd{Authors}\par
      }%
      \vskip 1em%
      {\large \ccaStructStart{P}\ccUseComp{Date}\ccaStructEnd{P}}% % Set date in \large size.
      \end{center}%
      \ccaStructEnd{Titlepage}%
      \par
      \vskip 1.5em
  }%
```

```
% Title page hooks
% Before \ccPrefix Maketitle and outside the group
\ccSetProperty{before-titlepage}{%
  \pagestyle{empty}%
  \parindent\z@
  \parskip\z@
}%
\ccSetProperty{after-titlepage}{\pagestyle{headings}}%
% Pages of title
%% Cover page
\ccSetProperty{coverpage}{%
  \bgroup
    \def\thepage{\@alph\c@page}%
    \smash{\rlap{%
        \raise\dimexpr\headheight+\headsep+\topmargin+\topskip-\paperheight\relax
        \vtop{%
          \hskip-\oddsidemargin
          \includegraphics[width=\paperwidth,height=\paperheight]{\ccUseComp{Cover}}%
        }}}%
    \ccUseProperty{after-coverpage}%
  \egroup
}%
\ccSetProperty{after-coverpage}{\cleardoublepage}%
\ccSetProperty{titlepage-roman}{%
  \ccUsePropertyEnv{titlepage-i}%
  \clearpage
  \ccUsePropertyEnv{titlepage-ii}%
  \clearpage
  \ccUsePropertyEnv{titlepage-iii}%
  \clearpage
  \ccUsePropertyEnv{titlepage-iv}%
  \clearpage
}%
%% Generic meta blocks
\ccSetProperty{generic-meta-heading-face}{\large}% format of the heading of a generic meta block
\ccSetProperty{generic-meta-format}{% Format of a single generic meta-block
  \ccIfComp{Heading}{{\ccUseProperty{generic-meta-heading-face}\ccUseComp{Heading}\par}\vskip\
      baselineskip}{}%
  \ccUseComp{Content}%
  \par%
}%
%% Funding
\ccSetProperty{funding-columns}{2}
\ccSetProperty{funding-format}{}%
```

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

```
\ccSetProperty{fund-width}{.5\textwidth}
\ccSetProperty{fund-vertical-sep}{\baselineskip}%
\ccSetProperty{fund-sep}{%
  \expandafter\@tempcnta\CalcModulo{\ccCurCount}{\ccUseProperty{funding-columns}}%
  \ifnum\@tempcnta=\z@
    \par
    \ifnum\ccCurCount<\ccTotalCount\relax
      \vskip\ccUseProperty{fund-vertical-sep}%
    \fi
  \else
    \hfill
  \fi}
\ccSetProperty{fund-format}{% Format of a single fund/grant/sponsor
```

```
    \strut\vtop{%
      \hsize\ccUseProperty{fund-width}%
      \ccWhenComp{FundName}{\ccaStructStart{P}\ccUseComp{FundName}\ccaStructEnd{P}\\[1ex]}%
      \ccaStructStart{Figure}%
      \ccaAddPlacement{Block}%
      \ccdefFromComp\cca@Gin@alt{FundName}%
      \ccSanitizeStr\@cca@Gin@alt{\cca@Gin@alt}%
      \ccaAddAltText{\@cca@Gin@alt}%
      \includegraphics[width=\ccUseProperty{fund-width}]{\ccUseComp{FundLogo}}%
      \ccaStructEnd{Figure}%
    }%
    \ccUseProperty{fund-sep}%
  }%
  \ccSetProperty{funding-sep}{4mm}%
  \ccSetProperty{funding-block}{%
    \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.

```
      \ccSetPropertyX{fund-width}{\dimexpr(\textwidth/\ccUseProperty{funding-columns})-(\
          ccUseProperty{funding-sep}/\ccUseProperty{funding-columns})\relax}
      \ccUseProperty{funding-format}%
      \ccGetComp{FundingPreText}%
      \ccGetComp*{FundingBlock}%
      \ccGetComp{FundingPostText}%
      \par
    \egroup
  }
  %% before the roman part of the title pages but after cover page
  \ccSetProperty{before-titlepage-roman}{%
    \setcounter{page}{1}%
    \def\thepage{\roman{page}}%
  }%
  \ccSetProperty{titlepage-i}{%
    \ccaStructStart{Titlepage}%
    \ifmonograph
      \ccaStructStart{Authors}\ccUseComp{AuthorNameList}\ccaStructEnd{Authors}%
    \else
      \ccaStructStart{Editors}\ccUseProperty{EditorNameList}\ccaStructEnd{Editors}%
    \fi%
    \vskip\baselineskip
    \bgroup
      \ccUseProperty{title-face}\ccaStructStart{Title}\ccaReplaceStruct{\cca@id@document@dummy}\
          ccUseComp{Title}\ccaStructEnd{Title}%
    \egroup
    \ccaStructEnd{Titlepage}%
  }%
  \ccSetProperty{titlepage-ii}{%
    \ccaStructStart{Titlepage}%
    \ccGetComp{Editorial}%
    \ccGetComp{SeriesNote}%
    \ccGetComp{GenericMetaBlock}%
    \vfill
    \ccUseProperty{bio-output}%
    \ccaStructEnd{Titlepage}%
  }%
  \ccSetProperty{titlepage-iii}{%
    \ccaStructStart{Titlepage}%
    \ifmonograph
      \ccaStructStart{Authors}\ccUseComp{AuthorNameList}\ccaStructEnd{Authors}%
    \else
      \ccaStructStart{Editors}\ccUseProperty{EditorNameList}\ccaStructEnd{Editors}%
    \fi%
    \par
    \ccUseProperty{title-format}
    \ccGetComp{Edition}%
    \ccGetComp{EditionNote}%
    \ccaStructEnd{Titlepage}%
    \vfill
    \clearpage
  }%
  \ccSetProperty{titlepage-iv}{%
    \ccaStructStart{Titlepage}%
    \ccGetComp{Dedication}% Dedication
    \ccGetComp{Acknowledgements}% Dedication
    \ccUseProperty{funding-block}%
    \vfill
    \bgroup
      \ccUseProperty{imprint-face}%
      \ccaStructStart{MetaDatumBlock}%
      \ccWhenComp{Biblio}{%
        \ccaStructStart{MetaDatum}%
          {\bfseries
           \ccaStructStart{MetaDatumLabel}%
           \ccGetComp*{BiblioTitle}%
           \ccaStructEnd{MetaDatumLabel}%
          }%
        \ccaStructStart{MetaDatumValue}%
        \ccGetComp*{Biblio}%
        \ccaStructEnd{MetaDatumValue}%
        \ccaStructEnd{MetaDatum}%
      }%
      \ccUseProperty{imprint-sep}%
      \ccUseProperty{imprint}%
      \ccaStructEnd{MetaDatumBlock}%
    \egroup
    \ccaStructEnd{Titlepage}
    \clearpage
  }%
  %% predefined face and format Properties
  \ccSetProperty{title-face}{\Huge\sffamily\bfseries}%
```

The document's main title is tagged with the `<Title/>` tag, which in PDF/UA-1 should be mapped to `<H1/>`.

```
  \ccSetProperty{title-format}{%
    \bgroup
      \ccaVstructStart{Title}% PDF 2.0
      \ccUseProperty{title-face}%
      \ccUseComp{Title}\par
      \ccaVstructEnd{Title}% PDF 2.0
    \egroup
    \ccWhenComp{Subtitle}{\ccUseProperty{subtitle-format}}%
    \ccWhenComp{TitleNote}{\ccUseProperty{title-note-format}}%
    \ccGetComp{Statement}%
    \vskip\baselineskip
  }%
  \ccSetProperty{title-note-face}{\large\sffamily}%
  \ccSetProperty{title-note-format}{%
    \bgroup
      \ccUseProperty{title-note-face}%
      \ccUseComp{TitleNote}%
    \egroup
    \par
  }%
  \ccSetProperty{subtitle-face}{\Large\sffamily\bfseries}%
  \ccSetProperty{subtitle-format}{%
    \bgroup
      \ccUseProperty{subtitle-face}%
      \ccUseComp{Subtitle}%
    \egroup
    \par
  }%
  %% Imprint
  \ccSetProperty{imprint-face}{\footnotesize}%
  \ccSetProperty{imprint-sep}{\ifhmode\par\fi\addvspace{\baselineskip}}%
  \ccSetProperty{imprint}{%
    \ccUseProperty{publisher}%
    \ccGetComp{Qualification}%%
    \ccGetComp{Conversion}%%
    \ccGetComp{CoverDesign}%%
    \ccGetComp{CoverImage}%%
    \ccGetComp{Lectorate}%%
    \ccGetComp{QA}%%
    \ccGetComp{Translator}%%
    \ccGetComp{Appraiser}%%
    \ccGetComp{Discussion}%%
    \ccGetComp{Typesetter}%%
    \ccGetComp{Print}%%
    \ccGetComp{UsedFont}%%
    \ccGetComp{DOI}%%
    \ccGetComp{Keywords}%%
    \ccUseProperty{imprint-sep}%
    \ccGetComp{ISBNPreText}%
    \ccGetComp{ISBN}%
    \ccGetComp{EpubPreText}%
    \ccGetComp{EISBN}%
    \ccGetComp{EpubISBN}%
    \ccUseProperty{imprint-sep}%
    \ccGetComp{EnvDisclaimer}%
  }%
  \ccSetProperty{journal-meta}{%
    \ccUseLabeledComp{Submitted}%
    \ccUseLabeledComp{Received}%
    \ccUseLabeledComp{Revised}%
    \ccUseLabeledComp{Accepted}%
    \ccUseLabeledComp{Published}%
    \ccUseLabeledComp{Copyright}%
    \ccUseLabeledComp{COIStatement}%
    \ccUseLabeledComp{Keywords}%
  }%
  \ccSetProperty{licence}{%
    \ccIfComp{LicenceLogo}{\includegraphics{\ccUseComp{LicenceLogo}}\par}{}%
    \ccGetComp{LicenceText}%
  }%
  \ccSetProperty{copyright}{%
    \ccaStructStart{MetaDatum}%
    \ccaStructStart{P}%
    \ccIfComp{Copyright}
      {\ccUseComp{Copyright}\par}
      {\textcopyright\space\ccUseComp{Year}\space\ccUseComp{Publisher},\space\ccUseComp{PubPlace
          }\par}%
    \ccaStructEnd{P}%
  \ccaStructEnd{MetaDatum}%
}%
\ccSetProperty{publisher}{%
  \ccGetComp{PubDivInfo}%
  \ccUseProperty{copyright}%
  \ccGetComp{PubNote}%
  \ccGetComp{PubWeb}%
}%
% Name Formats
\ccSetProperty{counted-meta-sep}{\ifnum\ccCurCount<\ccTotalCount\relax\vskip\baselineskip\fi}%
      separator between multiple instances of the same meta datum
\ccSetProperty{counted-name-sep}{% Separator between multiple names; titlepage-specific override of
     the same Property in coco-meta!
  \ifnum\ccTotalCount>1\relax
    \ifnum\ccCurCount<\ccTotalCount\relax
      \ifnum\ccCurCount<\numexpr\ccTotalCount-1\relax
        \ccUseProperty{name-sep}%
      \else
        \ccUseProperty{name-and}%
      \fi
    \fi
  \fi
}%
% Aliasses for different Roles, see coco-meta.sty for the actual Property values:
%% editors:
\ccPropertyLet{editor-cite-name-format} {role-cite-name-format}%
\ccPropertyLet{editor-short-cite-name-format} {role-short-cite-name-format}%
\ccPropertyLet{editor-full-name-format} {role-full-name-format}%
\ccPropertyLet{editor-pdfinfo-name-format} {role-pdfinfo-name-format}%
\ccPropertyLet{editor-correspondence-as-format} {role-correspondence-string-format}%
%
\ccPropertyLet{editor-list-print-format} {role-block-print-format}%
\ccPropertyLet{editor-list-cite-format} {role-block-cite-format}%
\ccPropertyLet{editor-list-short-cite-format} {role-block-short-cite-format}%
\ccPropertyLet{editor-list-pdfinfo-format} {role-block-pdfinfo-format}%
\ccPropertyLet{editor-list-correspondence-format} {role-block-correspondence-format}%
%% series-editors:
\ccPropertyLet{series-editor-cite-name-format} {role-cite-name-format}%
\ccPropertyLet{series-editor-short-cite-name-format} {role-short-cite-name-format}%
\ccPropertyLet{series-editor-full-name-format} {role-full-name-format}%
\ccPropertyLet{series-editor-pdfinfo-name-format} {role-pdfinfo-name-format}%
\ccPropertyLet{series-editor-correspondence-as-format} {role-correspondence-as-format}%
%
\ccPropertyLet{series-editor-list-print-format} {role-block-print-format}%
\ccPropertyLet{series-editor-list-cite-format} {role-block-cite-format}%
\ccPropertyLet{series-editor-list-short-cite-format} {role-block-short-cite-format}%
\ccPropertyLet{series-editor-list-pdfinfo-format} {role-block-pdfinfo-format}%
\ccPropertyLet{series-editor-list-correspondence-format} {role-block-correspondence-format}%
%% name Separators
\ccSetProperty{editor-suffix-sgl}{(Ed.)}%
\ccSetProperty{editor-suffix-pl}{(Eds.)}%
\ccSetProperty{editor-suffix}{%
  \space
  \ifnum\ccTotalCount=\@ne\relax
    \ccUseProperty{editor-suffix-sgl}%
  \else
    \ccUseProperty{editor-suffix-pl}%
  \fi
}%
% Biography
```

```
  % those Properties control how (Role specific) Biography Blocks are formatted, i.e. the list of all
      Biographies of a specific Role:
  \ccSetProperty{role-bio-block-face}{}% face for the entire, role-specific, Biography Block
  \ccSetProperty{role-bio-block-format}{{\ccUseProperty{role-bio-block-face}\ccUseComp{Biography
      }}\par}% Format of the whole, Role specific, Biography Block
  \ccPropertyLet{author-bio-block-format} {role-bio-block-format}% Override for single author meta
      info
  \ccPropertyLet{editor-bio-block-format} {role-bio-block-format}% Override for single editor meta
      info
  \ccPropertyLet{series-editor-bio-block-format} {role-bio-block-format}% Override for single
      series editor meta info
  % those Properties control how a (Role specific) Biography is formatted:
  \ccSetProperty{role-biography-format}{{\bfseries\ccUseComp{FullName}:}\space\ccUseComp{Bio}\
      par}% Format of a single entry in the Role specific Biography
  \ccPropertyLet{author-biography-format} {role-biography-format}% Override for single author meta
      info
  \ccPropertyLet{editor-biography-format} {role-biography-format}% Override for single editor meta
      info
  \ccPropertyLet{series-editor-biography-format} {role-biography-format}% Override for single
      series editor meta info
  \ccSetProperty{bio-output-format}{%
    \ccGetComp{AuthorBioBlock}%
    \ccGetComp{EditorBioBlock}%
    \ccGetComp{SeriesEditorBioBlock}%
  }%
  % Running headers
  \ccSetProperty{run-book-title}{%
    \ccIfComp{RunTitle}
      {\ccUseComp{RunTitle}}
      {\ccIfComp{ShortTitle}
        {\ccUseComp{ShortTitle}}
        {\ccIfComp{Title}{\ccUseComp{Title}}{No title given!}}}%
  }%
  \ccSetProperty{run-book-name}{%
    \ccIfComp{RunNames}
      {\ccUseComp{RunNames}}
      {\ifmonograph
         \ccIfComp{AuthorNameList}
           {\ccUseComp{AuthorNameList}}
           {no author defined!}%
       \else
         \ccIfComp{EditorNameList}
           {\ccUseComp{EditorNameList}}
           {no editor defined!}%
       \fi}%
  }%
  \ccSetProperty{doc-book-title}{%
    \ccIfComp{DocTitle}
      {\ccUseComp{DocTitle}}
      {\ccIfComp{ShortTitle}
        {\ccUseComp{ShortTitle}}
        {\ccUseComp{Title}}}%
  }%
}
```

5 Accessibility Features

5.1 Output Intent and ICC Profiles

```
\ifx\cc@color@enc\relax\else
\ccWhenAlly{%
```

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

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

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

```
    \ccDeclareGlobalComponent{IccProfileFile}
```

IccComponents holds the number of components in the color profile

```
    \ccDeclareGlobalComponent{IccComponents}
```

IccIdentifier holds the identifier of the color profile

```
    \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):

```
  \ifx\cc@color@enc\cc@str@cmyk
    \def\cca@default@icc@comp{4}%
    \def\cca@default@icc@iden{Coated FOGRA39}%
  \else\ifx\cc@color@enc\cc@str@rgb
      \def\cca@default@icc@comp{3}%
      \def\cca@default@icc@iden{IEC 61966-2.1 Default RGB colour space - sRGB}%
    \else
      \def\cca@default@icc@comp{1}%
      \def\cca@default@icc@iden{ISO Coated v2 - GREY 1c - (basICColor)}%
    \fi
  \fi
  \ccAddToType{Properties}{titlepage}{%
```

output-intent `<see below>` sends the output intent information to the ltpdfa package. It must contain of three data fields:

profile with the name of the to-be-embedded .icc file,
componetns with an integer telling the pdfwriter how many values are coded by each color (e.g., 4 for cmyk, 3 for rgb)
identifier with the identifying name of the profile (e.g., Coated FOGRA39 for the included cmyk profile, etc.)

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

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

```
  \AddToHook{cct/document/meta}{\edef\x{\noexpand\ccaAddToConfig{intent}{\ccUseProperty{output-
      intent}}}\x}
```

5.2 Encoding of the PDF-A Conformance

We assume that both PDF/UA standards also require conformity to PDF/A2-a, so we let ltpdfa write that into the PDF file.

```
 \AddToHook{cct/document/meta}{%
   \edef\x{\noexpand\ccaSetDocinfo{conformance}{%
       pdfaid=2;%
       level=A%
       ;pdfuaid=\if@cc@pdf@two2\else1\fi}}%
   \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.

```
 \if@cc@pdf@two\else\ccaAddRolemap{Title}{P}\fi
 \ccaAddRolemap{Titlepage}{Div}
```

```
}%ccWhenAlly
\fi
```

```
</title>
```

Module 10
coco-floats.dtx

Output driver for `coco-floats.sty`.

```
<*floats>
```

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

Note that we take the term ‘‘Float’’ quite liberally: ‘‘Floats’’ basicly mean *‘‘things that may have a caption and which are somewhat outside the main text body’’*, whether they actually float (i. e., moved into the `\@toplist` or `\@botlist` by LaTeX), or not.

```
%%
%% module for CoCoTeX that extends floating objects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-floats}
    [2024/12/13 v0.5.0 CoCoTeX floats module]
\ExplSyntaxOn
\keys_define:nn { cocotex/floats }
{
  nofigs .code:n = {\global\let\ccf@no@figs\relax}
}
\ProcessKeyOptions[cocotex/floats]
\ExplSyntaxOff
```

1 Package Setup

1.1 Hard requirements

For the list-of mechanism, we need the CoCoTeX common module, which also loads the CoCoTeX kernel module.

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

For landscape images, we load the `rotating` package.

```
\RequirePackage{rotating}
```

SInce file names form word often contain spaces and periods, we also include the `grffile` package.

```
\RequirePackage{grffile}
```

In order to save footnotes in captions, we require the `footnote` package.

```
\RequirePackage{footnote}
```

The adjustbox package is needed to restrict the maximum dimensions of image files.

```
\RequirePackage[Export]{adjustbox}
```

Finally, we need the stfloats package to allow bottom placed images on pages that start LaTeX's twocolumn mode.

```
\usepackage{stfloats}
\setcounter{dblbotnumber}{5}
```

1.2 Adjustments at the Beginning of the Document

```
\AtBeginDocument{%
```

The first adjustment implements the nofigs option by deactivating the \includegraphics macro.

```
  \ifx\ccf@no@figs\relax
    \renewcommand\includegraphics[2][]{}%
  \fi
```

\ccf@ltx@includegraphics stores the final definition of the \includegraphics macro for later use.

```
  \global\let\ccf@ltx@includegraphics\includegraphics
```

Adjustments to the htmltabs package, if it is used:

```
  \@ifpackageloaded{htmltabs}
    {\global\let\cc@uses@htmltabs\relax
     \def\ht@adjust@linewidth{%
       \advance\ht@h@offset\leftskip
       \advance\ht@h@offset\@totalleftmargin
       \advance\linewidth-\rightskip
     }%
    }{}%
```

In order to catch the actual dimensions of the float box, we need to hook into LaTeX'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.

```
  \gappto\@endfloatbox{%
    \global\ccf@total@height=\ht\@currbox\relax%
    \global\ccf@total@depth=\dp\@currbox\relax%
  }%
}%
```

1.3 Document Class-Option Overrides

Since CoCoTeX is mainly developed for automatic typesetting and float positioning, we set rather high tolerances for macros from LaTeX's standard .clo files:

```
\def\topfraction{0.9}
\def\textfraction{0.1}
\def\bottomfraction{0.8}
\def\totalnumber{8}
\def\topnumber{8}
```

```
\def\bottomnumber{8}
\def\floatpagefraction{0.8}
\@fptop\z@
\@fpbot\@flushglue
```

1.4 Internal Registers

`\ccf@floatbox` is for measuring the dimensions of the whole float

```
\newbox \ccf@floatbox
```

`\ccf@sub@box` is for measuring a single sub-float.

```
\newbox \ccf@sub@box
```

`\ccf@int@cnt` is an internal global counter that numbers all top-level floats sequentially.

```
\newcount\ccf@int@cnt \ccf@int@cnt\z@
```

`\ccSubFloatCnt` counts the sub-floats within a parent float Container instance.

```
\newcount\ccSubFloatCnt \ccSubFloatCnt=\z@\relax
```

`\ccf@int@sub@flt@cnt` is a temporary counter that holds the total number of subfloats inside a parent float Container instance.

```
\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

```
\newdimen\ccf@sub@maxheight \ccf@sub@maxheight=\z@\relax
```

`\ccf@sub@sep` is the space between sub-floats

```
\newdimen\ccf@sub@sep \ccf@sub@sep=\fboxsep\relax
```

`\ccf@total@width` stores the cumulated overall width of the entire float

```
\newdimen\ccf@total@width \ccf@total@width=\textwidth\relax
```

`\ccf@total@height` is the overall height of a float

```
\newdimen\ccf@total@height \ccf@total@height=\textwidth\relax
```

`\ccf@total@depth` is the overall depth of a float

```
\newdimen\ccf@total@depth \ccf@total@depth=\textwidth\relax
```

`\ccf@calc@width` is an internal dimension used to calculate the ratio between mutiple sub-floats that should be scaled to the same height

```
\newdimen\ccf@calc@width \ccf@calc@width=\ccf@total@width\relax
```

`\ccf@sep@top` holds the actual vertical skip inserted at the top of a float. If the float is floating, this equals to `intext-skip`, or `float-skip`, otherwise.

```
\newskip\ccf@sep@top \ccf@sep@top=\z@\relax
```

`\ccf@sep@bottom` is the same for the bottom vertical skip.

```
\newskip\ccf@sep@bottom \ccf@sep@bottom=\z@\relax
```

Internal dimensions for the horizontal margins:

`\ccf@margin@r` holds the right side margin

```
\newdimen\ccf@margin@r \ccf@margin@r=\z@\relax
```

`\ccf@margin@l` holds the left side margin

```
\newdimen\ccf@margin@l \ccf@margin@l=\z@\relax
```

`\ccf@margin@i` holds the inner margin

```
\newdimen\ccf@margin@i \ccf@margin@i=\z@\relax
```

`\ccf@margin@o` holds the outer margin

```
\newdimen\ccf@margin@o \ccf@margin@o=\z@\relax
```

`\if@ccf@break@capt` is a locally adjustable switch that indicates whether captions are allowed to break across pages (`true`) or not (`false`).

```
\newif\if@ccf@break@capt \@ccf@break@captfalse
```

`\if@ccf@sameheight` determins if subfloats should be scaled such that they are all the same height.

```
\newif\if@ccf@sameheight \@ccf@sameheighttrue
```

2 Internal macros

2.1 Generic resetter

Some macros are re-evaluated for each new top-level float.

`\ccf@reset@defaults` resets the those macros. It is called at the very beginning of each new float.

```
\def\ccf@reset@defaults{%
  \global\ccSubFloatCnt=\z@
  \global\ccf@total@width=\z@
  \global\let\ccf@has@capt@top\@undefined
  \global\let\ccf@has@capt@bottom\@undefined
  \global\let\ccf@has@subcapt@top\@undefined
  \global\let\ccf@has@subcapt@bottom\@undefined
  \global\let\ccf@sub@contentsline@store\@empty
  \global\ccf@sub@maxheight=\z@\relax
  \@tempcnta=\z@\relax
  \cc@reset@components{\cc@cur@cont}%
  \let\ccf@prefix\@empty
  \let\ht@cur@element\ccfCapType
  \global\let\ccf@current@class\relax
  \global\let\ccf@landscape\relax
}
```

2.2 Wrapper for LaTeX's Native float Environments

`\ccf@set@env` determines the low-level LaTeX float environment depending on orientation and document options. If no `float-pos` is given (implicitely or determined), the object is not treated as a float at all.

```
\def\ccf@set@env{%
  \ifx\ccf@floatpos\@empty
    \let\ccf@begin@env\bgroup
    \let\ccf@end@env\egroup
  \else
    \ifx\ccf@landscape\@empty
      \edef\ccf@env@name{sideways\ccfCapType}%
       \edef\ccf@begin@env{\noexpand\begin{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}%
       \edef\ccf@end@env{\noexpand\end{\ccf@env@name\ifx\ccf@do@dbl\relax*\fi}}
   \else
      \edef\ccf@env@name{\ifx\ccf@do@dbl\relax dbl\fi float}%
      \edef\ccf@begin@env{\expandafter\noexpand\csname @x\ccf@env@name\endcsname {\ccfCapType}[\
          ccf@floatpos]}%
      \edef\ccf@end@env{\expandafter\noexpand\csname end@\ccf@env@name\endcsname}%
    \fi
  \fi}
```

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

```
\def\ccf@get@seps{%
  \ifx\ccf@floatpos\@empty
    \expandafter\ccf@sep@top\dimexpr\ccUseProperty{intext-skip-top}\relax%
  \else
    \expandafter\ccf@sep@top\dimexpr\ccUseProperty{float-skip-top}\relax%
  \fi
  \ifx\ccf@landscape\relax
    \ifx\ccf@floatpos\@empty
      \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{intext-skip-bottom}\relax%
    \else
      \expandafter\ccf@sep@bottom\dimexpr\ccUseProperty{float-skip-bottom}\relax%
    \fi
  \fi}
```

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

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

3 The Generic float Container

Components in Containers that are derived from the abstract `float` are in fact all Counted Components, where top-level instances use 0 as their internal counter and sub-floats are counted incrementally. Thus, we can *simplify* the **internal** names to `<Componentname>-<Counter>`, which is done via a custom wrapper for the `\cc@def@counted@comp` Component declarator.

`\ccfMakeComp` is a shortcut for float Component declarations.

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

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

`\ccfMakeCompL` is a shortcut to declare Float Components together with their *list-of* overrides.

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

```
\def\ccfMakeCompL#1{%
  \ccfMakeComp{#1}%
  \ccfMakeComp{Listof#1}}
```

`float` is the main parent Container for all floats.

```
\ccDeclareContainer{float}{%
```

3.1 Common Float Components

```
  \ccDeclareType{Components}{%
```

First, we set the naming scheme of the internal Component macros which is then valid for all Component declarations by locally re-defining `\cc@counted@comp@scheme`.

```
    \def\cc@counted@comp@scheme#1{#1-\the\ccSubFloatCnt}%
```

`Content` is the main content holder of a float.

```
    \ccfMakeComp{Content}%
```

`Caption` is the main caption of a float.

`ListofCaptionOR` is the corresponding list-of-entry

```
    \ccfMakeCompL{Caption}%
```

`Legend` is a legend to a float.

`ListofLegendOR` is the corresponding list-of-entry

```
    \ccfMakeCompL{Legend}%
```

`Source` is the source of a float.

`ListofSourceOR` is the corresponding list-of-entry

```
    \ccfMakeCompL{Source}%
```

`Number` is the counter of the float (including the label)

`ListofNumberOR` is the corresponding list-of-entry

```
    \ccfMakeCompL{Number}%
```

`RefLabel` is the float's ID used for cross-references (replaces LaTeX's `\label` command)

```
    \ccfMakeComp{RefLabel}%
```

`ListofEntryCL` is the Collection Component for the entire Listof entry.

```
    \ccfMakeComp{ListofEntry}%
  }%
```

3.2 Common Float Properties

```
  \ccDeclareType{Properties}{%
```

Placement and Spacing

`intext-skip-top` `<skip>` vertical space between the text body and following non-floating floats

```
    \ccSetProperty{intext-skip-top}{\intextsep}%
```

`intext-skip-bottom` `<skip>` vertical space between non-floating floats and the following text body

```
    \ccSetProperty{intext-skip-bottom}{\intextsep}%
```

`float-skip-top` `<skip>` vertical space between text body and following floating floats

```
    \ccSetProperty{float-skip-top}{\z@}%
```

`float-skip-bottom` `<skip>` vertical space between floating floats and following text body

```
    \ccSetProperty{float-skip-bottom}{\z@}%
```

`sub-float-sep` `<skip>` horizontal space between sub-floats

```
    \ccSetProperty{sub-float-sep}{\ccf@sub@sep}%
```

`margin-inner` `<skip>` inner margins of floats in twopage mode, i. e., left margin on odd pages and right margin on even pages, respectively.

```
    \ccSetProperty{margin-inner}{\z@}%
```

`margin-outer` `<skip>` outer margin of floats in twopage mode, i. e., right margin on odd pages and left margin on even pages, respectively.

```
    \ccSetProperty{margin-outer}{\z@}%
```

`margin-left` `<skip>` horizontal space between the left page area boundary and the float.

```
\ccSetProperty{margin-left}{\z@}%
```

`margin-right` `<skip>` horizontal space between the right page area boundary and the float.

```
\ccSetProperty{margin-right}{\z@}%
```

`before-float` `<any>` is the code that is executed before a float's content is evaluated.

```
\ccSetProperty{before-float}{\parindent\z@}%
```

`fix-dimen` `[true|false]` if `true`, the content is *always* scaled to `\hsize`, i.e., the `\textwidth` minus the left and right margins, even if it means to scale *up* an image file.

```
\ccSetProperty{fix-dimen}{false}%
```

Properties for Float-Type Handlers

`subfloat-content` `<any>` is the material that is put into the `\ccf@sub@box` for further processing.

```
\ccSetProperty{subfloat-content}{\ccUseComp{Content}}%
```

`float-render` `<any>` the output routine for top-level float type specific contents

```
\ccSetProperty{float-render}{\ccUseComp{Content}}%
```

`subfloat-render` `<any>` the output routine for second-level float type specific contents.

```
\ccSetProperty{subfloat-render}{\ccUseComp{Content}}%
```

Properties for Captions

`caption-face` `<any>` style applied to both top and bottom placed captions

```
\ccSetProperty{caption-face}{}%
```

`caption-face-top` `<any>` style applied to top placed captions only

```
\ccSetProperty{caption-face-top}{}%
```

`caption-face-bottom` `<any>` style applied to bottom placed captions only

```
\ccSetProperty{caption-face-bottom}{}%
```

`source-face` `<any>` style applied to the printed `Source` Component.

```
\ccSetProperty{source-face}{}%
```

`legend-face` `<any>` style applied to the printed `Legend` Component.

```
\ccSetProperty{legend-face}{}%
```

`caption-sep-top` `<skip>` vertical space between top caption and content, i. e., the skip *after* the top placed caption.

```
\ccSetProperty{caption-sep-top}{\z@}%
```

`caption-sep-top` `<skip>` vertical space between bottom caption and content, i.e., the skip *before* the bottom placed caption.

```
  \ccSetProperty{caption-sep-bottom}{\z@}%
```

`caption-top` `<any>` the content of the top placed caption

```
  \ccSetProperty{caption-top}{%
    \ccIfComp{Number}{{\ccUseProperty{number-face}\ccUseComp{Number}\ccUseProperty{number-sep
        }}}{}%
    \ccUseComp{Caption}%
  }%
```

`caption-bottom` `<any>` the content of the bottom placed caption

```
  \ccSetProperty{caption-bottom}{%
    \ccIfComp{Legend}{{\ccUseProperty{legend-face}\ccUseComp{Legend}}}{}%
    \ccIfComp{Source}{%
      \ccIfComp{Legend}{\par\nopagebreak}{}%
        {\ccUseProperty{source-face}%
         \ccUseComp{Source}}}{}}%
```

`subcaption-face` `<any>` the style of captions of second level floats

```
  \ccPropertyLet{subcaption-face}{caption-face}%
```

`subcaption-face-top` `<any>` the style of top placed captions of second level floats

```
  \ccSetProperty{subcaption-face-top}{\ccUseProperty{caption-face-top}}%
```

`subcaption-face-bottom` `<any>` the style of bottom placed captions of second level floats

```
  \ccSetProperty{subcaption-face-bottom}{\ccUseProperty{caption-face-bottom}}%
```

`subcaption-add-sep-top` `<skip>` additional vertical space between top caption and top sub-caption

```
  \ccSetProperty{subcaption-add-sep-top}{\z@}%
```

`subcaption-add-sep-bottom` `<skip>` additional vertical space between bottom sub-caption and bottom caption

```
  \ccSetProperty{subcaption-add-sep-bottom}{\z@}%
```

`subcaption-sep-top` `<skip>` vertical space between top placed sub-captions and content, i. e., the space *after* top placed sub-captions.

```
  \ccSetProperty{subcaption-sep-top}{\ccUseProperty{caption-sep-top}}%
```

`subcaption-sep-top` `<skip>` vertical space between content and top placed sub-captions, i. e., the space before bottom placed sub-captions.

```
  \ccSetProperty{subcaption-sep-bottom}{\ccUseProperty{caption-sep-bottom}}%
```

`subcaption-top` `<any>` the content of top placed sub-captions

```
  \ccSetProperty{subcaption-top}{\ccUseProperty{caption-top}}%
```

`subcaption-bottom` `<any>` the content of bottom placed sub-captions

```
  \ccSetProperty{subcaption-bottom}{\ccUseProperty{caption-bottom}}%
```

`subcaption-valign-top` `[top|bottom|middle]` vertical alignment of neighboring top-placed sub-captions

```
\ccSetProperty{subcaption-valign-top}{top}%
```

`subcaption-valign-bottom` `[top|bottom|middle]` vertical alignment of neighboring bottom-placed sub-captions

```
\ccSetProperty{subcaption-valign-bottom}{top}%
```

Properties for Counters

`auto-number-prefix` `<any>` Prefix for auto-generated Number components

```
\ccSetProperty{auto-number-prefix}{\csname\ccfCapType name\endcsname}%
```

`auto-number-prefix-sep` `<any>` Separator between the auto-generated number prefix and the auto-generated Number component.

```
\ccSetProperty{auto-number-prefix-sep}{~}%
```

`numbering` `[auto|<any>]` if `auto`, float counters in instances without the Number component are generated automatically. Any other value suppresses auto-numbering.

```
\ccSetProperty{numbering}{auto}%
```

`numbering` `[auto|<any>]` if `auto`, subfloat counters in instances without the Number component are generated automatically. Any other value suppresses auto-numbering.

Note: this Property has only effect when subfloats are second-level. In first-level sub-floats, the `numbering` Property is used.

```
\ccSetProperty{sub-numbering}{}%
```

`number-sep` `<any>` separator bewteen the printed float number and the caption

```
\ccSetProperty{number-sep}{\enskip}%
```

`number-face` `<any>` style of number, additional to caption-format

```
\ccSetProperty{number-face}{\bfseries}%
```

`sub-number-sep` `<any>` separator between number and caption in sub-floats

```
\ccSetProperty{sub-number-sep}{\,}%
```

`sub-number-style` `[arabic|Alph|alph|roman|Roman]` numbering style for automatically generated subfloat counters

```
\ccSetProperty{sub-number-style}{alph}%
```

`sub-number-face` `<any>` style of the number of a subfloat

```
\ccSetProperty{sub-number-face}{}%
```

`sub-number-before` `<any>` stuff that is put immediately before the automatically generated subfloat counter

```
\ccSetProperty{sub-number-before}{(}%
```

`sub-number-before` `<any>` stuff that is put immediately after the automatically generated subfloat counter

```
\ccSetProperty{sub-number-after}{)}%
```

`sub-number-format` `<any>` the format of the number

```
\ccSetProperty{sub-number-format}{%
  \ccUseProperty{float-number}%
  \ccUseProperty{sub-number-sep}%
  \ccUseProperty{sub-number}}%
```

`label-pos` `[top|bottom]` position of the cross reference anchor, refering to top or bottom placed captions.

```
\ccSetProperty{label-pos}{top}%
```

`sublabel-pos` `[top|bottom]` position of the cross reference anchor for sub-floats, refering to top or bottom placed sub-captions.

```
\ccSetProperty{sublabel-pos}{top}%
```

Properties for List-Of Entries

`list-of-page-sep` `<any>` separator between the listof-entry and the page

```
\ccSetProperty{list-of-page-sep}{\ccaVstructStart[Document]{leaders}\dotfill\ccaVstructEnd{
    leaders}}%
```

`list-of-number-face` `<any>` style of the listof-entry

```
\ccPropertyLet{list-of-number-face}{list-of-caption-face}%
```

`list-of-number-sep` `<any>` separator between the number and the listof entry.

```
\ccSetProperty{list-of-number-sep}{\enskip}%
```

`list-of-number-align` `[left|center|right]` horizontal alignment of the listof number within its local hbox.

```
\ccSetProperty{list-of-number-align}{left}%
```

`list-of-number-format` `<any>` format of the number in listof entries.

```
\ccSetProperty{list-of-number-format}{%
  \bgroup
    \ccUseProperty{list-of-number-face}%
    \ccUseComp{ListofNumber}%
    \ccUseProperty{list-of-number-sep}%
  \egroup}%
```

`list-of-parfillskip` `<skip>` parfillskip of an entry in the listof

```
\ccSetProperty{list-of-parfillskip}{-\rightskip}%
```

`list-of-margin-right` `<skip>` right margin of the listof entry

```
\ccSetProperty{list-of-margin-right}{\@pnumwidth \@plus 1fil}%
```

`list-of-margin-left` `[auto|<skip>]` right margin of the listof entry

```
\ccSetProperty{list-of-margin-left}{auto}%
```

`list-of-indent` `[auto|<dimen>]` horizontal offset of the first line of an listof-entry, relative to `margin-left`.

```
\ccSetProperty{list-of-indent}{auto}%
```

list-of-block <any> format of the entire listof entry.

```
  \ccSetProperty{list-of-block}{%
    \ccUseProperty{list-of-caption-face}%
    \ccaStructStart{TOCI}%
    \ccaStructStart{P}\ccaStructStart{Reference}%
    \ccIfComp{ListofNumber}
      {\ccaStructStart{Lbl}\ccUseComp{list-of-hang-number}\ccaStructEnd{Lbl}}
      {\leftskip0pt}%
    \ccaStructStart{Span}%
    \ccUseComp{ListofCaption}%
    \ccUseProperty{list-of-page-sep}\ccUseComp{ListofPage}%
    \ccaStructEnd{Span}%
    \ccaStructEnd{Reference}\ccaStructEnd{P}%
    \ccaStructEnd{TOCI}%
  }%
```

list-of-before-entry <any> material inserted at the beginning of each listof entry

```
  \ccSetProperty{list-of-before-entry}{%
    \ccGobble
    \leftskip\ccUseProperty{list-of-margin-left}\relax%
    \rightskip \ccUseProperty{list-of-margin-right}\relax%
    \parfillskip \ccUseProperty{list-of-parfillskip}\relax
    \parindent\z@
    \@afterindenttrue
    \interlinepenalty\@M
    \leavevmode
    \null\nobreak
  }% list-of-float appearance
```

list-of-after-entry <any> material inserted at the end of a listof entry.

```
  \ccSetProperty{list-of-after-entry}{\par}%
}% /Properties
\ccDeclareType{Attributes}{%
```

class <string> is the class of the Float.

```
  \ccDeclareAttributeHandler{class}{\xdef\ccf@current@class{\ccAttrVal}}%
```

break-caption <bool> whether or not the caption is allowed to break across pages

```
  \ccDeclareAttributeHandler*{break-caption}[\@ccf@break@captfalse]{\@ccf@break@capttrue}%
```

float-pos [(h|t|p|b)*|h!] the float position of the float. h! means that the float is not actually floating and is equivalent to omitting the Attribute.

```
  \ccDeclareAttributeHandler{float-pos}[\let\ccf@floatpos\@empty]{\ccf@attr@pos{\ccAttrVal}}%
```

orientation landscape is whether the float is rotated by 90° (landscape) or not (if omitted, default)

```
  \ccDeclareAttributeHandler{orientation}{\ccf@attr@orient{\ccAttrVal}}
```

debug <flag> if set, additional debugging is written into the shell and log file.

```
  \ccDeclareAttributeHandler*{debug}[\let\ccf@debug\relax]{\let\ccf@debug\ccf@attr@debug}
```

no-same-height <flag> if set, the same-height calculations are de-activated for this float.

```
    \ccDeclareAttributeHandler*{no-same-height}{\@ccf@sameheightfalse}%
```

```
  }
}% /Container
```

3.3 The Generic float Environment

This section defines the macros for the float's Container-specific LaTeX environment.

\ccf@float is a mid-level Macro that provides the common floating LaTeX environment. #1 is the float environment's kv-attribute list.

#1 float position (optional)

```
\def\ccf@float{\cc@opt@empty\@ccf@float}
\def\@ccf@float[#1]{%
  \par
  \begingroup
    \@cc@is@finalfalse
    \global\advance\ccf@int@cnt\@ne
    \ccEvalType{FloatEnvInfo}%
    \ccf@reset@defaults
    \ccToggleCountedConditionals
    \ccEvalType{Properties}%
    \ccIfPropVal{subfloat-same-height}{true}{\global\@ccf@sameheighttrue}{\global\
        @ccf@sameheightfalse}
    \ccEvalAttributes{#1}%
    \ccf@eval@class
    \ccf@set@hsize
    \ccf@get@seps
    \ccEvalType{Components}%
    \ccUseProperty{before-float}%
    \ccf@set@env
    \ifx\ccf@floatpos\@empty\else\savenotes\fi
    \@cc@is@finaltrue
    \ignorespaces}
```

\endccf@float is the end of the common float environment.

```
\def\endccf@float{%
    \ccf@begin@env
      \@cc@is@finalfalse
      \ccf@set@top@sep
      \ccf@int@sub@flt@cnt=\ccSubFloatCnt\relax
      \ccSubFloatCnt=\z@\relax
      \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
        {\ccf@create@counter
         \ccf@compose@listof}%
      \ccSubFloatCnt=\ccf@int@sub@flt@cnt\relax
      \ccf@test@caption{0}{}{top}%
      \ccf@test@caption{0}{}{bottom}%
      \bgroup
        \@cc@is@finaltrue
```

Floats as a whole are tagged as <Aside/> when PDF/UA standard is 2.0, or as <Div/>, otherwise.

```
      \ccaStructStart{\if@cc@pdf@two\relax Aside\else Div\fi}%
      \edef\ccf@parstruct@id{\ccaGetCurStruct{idx}}%
      \hsize\ccf@total@width
      \ccf@process
      \ccaStructEnd{\if@cc@pdf@two Aside\else Div\fi}%
      \par
    \egroup
    \ccSavePage
    \ccf@set@bot@sep
  \ccf@end@env
  \ccf@debug%
  \ifx\ccf@floatpos\@empty\else\spewnotes\fi
 \endgroup
 \ccf@store@dimens
 \global\let\ccf@current@class\relax
}
```

`\ccf@store@dimens` writes the float's final dimensions into the aux file.

```
\def\ccf@store@dimens{%
 \immediate\write\@auxout
  {\string\expandafter\string\gdef\string\csname\space cc-float-\the\ccf@int@cnt-dimens\string
      \endcsname{%
      {\the\ccf@total@width}%
      {\the\ccf@total@height}%
      {\the\ccf@total@depth}%
    }}%
 }
```

3.4 The SubFloat Environment

The SubFloat Sub-Container

Second-level floats (or SubFloats) are sub-containers of the float container.

`\ccSubFloat` is the user-level environment for sub-floats

```
\def\ccSubFloat{%
 \ifx\ccf@is@subfloat\relax
  \PackageError{coco-floats.sty}{Nested ccSubFloats detected!}{You cannot (yet) nest a `
      ccSubFloat' environment into another `ccSubFloat' environment!}%
 \else
  \global\let\ccf@is@subfloat\relax
  \global\advance\ccSubFloatCnt\@ne
 \fi
 \global\cslet{ccf@made@label@for@\the\ccSubFloatCnt}\relax
 \ignorespaces}
```

`\endccSubFloat` is the end of the sub-float environment

```
\def\endccSubFloat{%
 \ifhmode\unskip\fi
 \setbox\ccf@sub@box\hbox{\ccGobble
  \@cc@is@finalfalse
  \let\includegraphics\ccf@measuresubgraphics
  \ccUseProperty{subfloat-content}%
 }%
```

```
  \expandafter\xdef\csname ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcsname{\the\wd\
      ccf@sub@box}%
  \expandafter\xdef\csname ccf@\cc@cur@cont @height-\the\ccSubFloatCnt\endcsname{\the\ht\
      ccf@sub@box}%
  \expandafter\xdef\csname ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcsname{\the\dp\
      ccf@sub@box}%
  \@tempdima=\dimexpr\the\ht\ccf@sub@box+\the\dp\ccf@sub@box\relax
  \@tempdimb=\dimexpr\the\wd\ccf@sub@box\relax
  \ifdim\@tempdima>\ccf@sub@maxheight\relax
    \global\ccf@sub@maxheight=\@tempdima\relax
  \fi
  \global\setbox\ccf@sub@box\box\voidb@x
  \global\let\ccf@is@subfloat\@undefined
  \aftergroup\ignorespaces
}
```

Printing the Subfloats

`\ccfRenderSubFloats` iterates through the single sub-floats and renders them in a nice row. #1 is the subfloat counter, #2 is the Component name that contains the actual contents of the sub-float, for `\ccPrefix Figure` it is `Fig`, for `\ccPrefix Table` it is `Content`.

```
\long\def\ccfRenderSubFloats#1#2{%
  \leavevmode
  \savenotes
  \ifnum#1>\@ne\relax\hfill\fi
  \vtop\bgroup
    \expandafter\hsize\csname cc@\cc@cur@cont @res@width-#1\endcsname\relax
    \let\includegraphics\ccf@includesubgraphics
    \leavevmode
    \ccf@render@sub{#1}{#2}%
  \egroup
  \spewnotes
}
```

`\ccf@render@sub` renders a single sub-float. For the arguments, see `\ccfRenderSubFloats`, above.

```
\long\def\ccf@render@sub#1#2{%
  \ccSubFloatCnt=#1\relax
  \ccf@make@subcaption{top}%
  \bgroup\strut\ccUseComp{#2}\strut\par\egroup%
  \ccf@make@subcaption{bottom}}
```

3.5 Attribute Handlers

The following macros handle the Attributes of Float Container instances.

`\ccf@eval@class` expands the style class specific Properties.

```
\def\ccf@eval@class{%
  \ccUseStyleClass{default}{\ccfCapType}%
  \ifx\ccf@current@class\relax\else
    \ccUseStyleClass{\ccf@current@class}{\ccfCapType}%
  \fi}
```

`\ccf@attr@pos` is the handler for determining the float position. Some float Properties and Attributes restrict and override the explicit float positions, e.g., fully rotated floats must be positioned in `p` mode (i.e., as float page).

`{#1}` is the value of the float-pos Attribute. It may be any combination of `h`, `t`, `p`, `b`; or `h!`, which means that the float is non-floating (which is equivalent to an omitted `float-pos` Attribute)

```
\def\ccf@attr@pos#1{%
  \edef\ccf@floatpos{#1}%
  \def\@tempa{h!}\ifx\ccf@floatpos\@tempa\let\ccf@floatpos\@empty\fi
  \def\@tempa{h}\ifx\ccf@floatpos\@tempa\def\ccf@floatpos{htbp!}\fi
  \ifx\ccf@do@dbl\relax
    \ifx\ccf@floatpos\@empty\def\ccf@floatpos{htpb!}\fi% 11514
    \linewidth\dimexpr2\columnwidth+\columnsep\relax
    \hsize\linewidth\relax
  \fi
}
```

`\ccf@attr@orient` is the handler for the `orientation` Attribute.

`{#1}` is the value of the `orientation` Attribute. Currently, the only value that does things is `landscape`.

```
\def\ccf@attr@orient#1{%
  \ccIfStrEqual{#1}{landscape}
    {\linewidth\textheight
     \hsize\linewidth
     \global\let\ccf@landscape\@empty
     \def\ccf@floatpos{p}}{}}
```

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

```
\def\ccf@attr@debug{%
  \message{^^J[CoCo Float Debug]^^J
      Textheight:\space\the\textheight^^J
      Type:\space\space\space\space\space\space\space\cc@cur@cont^^J
\ifx\ccfCapType\cc@str@figure
      Path: \space\space\space\space\space\space\ccf@fig@path^^J
\fi
      Class:\space\space\space\space\space\space\ccf@current@class^^J
      Floatpos:\space\space\space\ccf@floatpos^^J
      Environ:\space\space\space\space\expandafter\noexpand\ccf@begin@env...\expandafter\noexpand
          \ccf@end@env^^J
      Subfloat:\space\space\space \the\ccSubFloatCnt^^J
\ifnum\ccSubFloatCnt=\z@
      Width:\space\space\space\space\space\space\the\ccf@total@width^^J
      Height:\space\space\space\space\space\the\ccf@total@height^^J
      Depth:\space\space\space\space\space\space\the\ccf@total@depth^^J
\else
      Width \the\ccSubFloatCnt:\space\space\space\space\space\space\expandafter\meaning\csname
          ccf@\cc@cur@cont @width-\the\ccSubFloatCnt\endcsname^^J
      Height \the\ccSubFloatCnt:\space\space\space\space\space \expandafter\meaning\csname ccf@\
          cc@cur@cont @height-\the\ccSubFloatCnt\endcsname^^J
      Depth \the\ccSubFloatCnt:\space\space\space\space\space\space\expandafter\meaning\csname
          ccf@\cc@cur@cont @depth-\the\ccSubFloatCnt\endcsname^^J
\fi}}
```

3.6 Handling of List-of Entries

`\ccf@generate@listof@handlers` generates handlers for listof-entries.

#1	is the file ending
#2	is the caption type

`#3` is the Container name

```
\def\ccf@generate@listof@handlers#1#2#3{%
```

`cc@listof@extract@data` The first macro that is dynamicly 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.

```
  \expandafter\gdef\csname cc@#2@extract@data\endcsname##1##2##3##4{%
    \ccSetContainer{#3}%
    \def\ccfCapType{#2}%
    \ccEvalType[#3]{Properties}%
    \ccDeclareComponent{ListofCaption}{}{}%
    \ccDeclareComponent{ListofLegend}{}{}%
    \ccDeclareComponent{ListofSource}{}{}%
    \ccDeclareComponent{ListofNumber}{}{}%
    \ccDeclareComponent{ListofPage}{}{}%
    \ccComponent{ListofPage}{\ccUseProperty{list-of-page-face}##4}%
    \cc@expand@l@contents{##3}{#3}{Listof}{Caption}%
    \cc@format@number{list-of-}{Listof}{##1}%
  }%
```

`\cc@listof@print@entry` The second dynamically defined macro is the entry renderer. It applies the Listof properties and selects the components to be printed. ##1 is the caption type of the float.

```
  \expandafter\gdef\csname cc@#2@print@entry\endcsname##1{%
    \bgroup
      \UseHook{cc/listof/##1/before}% was list-of-before-hook-##1
      \ccUseProperty{list-of-before-entry}%
      \@cc@is@finaltrue
      \ccUseProperty{list-of-block}%
      \UseHook{cc/listof/##1/after}% was list-of-after-hook-##1
      \ccUseProperty{list-of-after-entry}%
    \egroup}%
```

`cc/listof/[type]/before` is exanded before the List-of entry for a float of type `[type]` is printed

```
  \NewHook{cc/listof/#2/before}%
```

`cc/listof/[type]/after` is exanded after the Listof entry for a float of type `[type]` is printed, but before the Property `list-of-after-entry` is called.

```
  \NewHook{cc/listof/#2/after}%
}
```

`\ccf@addcontentsline` fork of LaTeX's `\addtocontents` macro.

```
\def\ccf@addcontentsline{%
  \ccWhenComp{ListofEntry}{%
    \protected@write\@auxout
      {\ccGobble\ccaProtect}%
      {\string\@writefile{\ccf@cap@list@type}
        {\protect\ccContentsline
          {\ifnum\ccSubFloatCnt>\z@\ccIfAttr{\ccfCapType}{subfloat}{sub}{}\fi\ccfCapType}
        {\ccUseComp{ListofEntry}}
        {\thepage}
        {\@currentHref}\protected@file@percent}}\relax%
   \protected@write\@auxout{\ccaEnable}{\relax}%
  }%
}
```

`\ccf@check@empty` is a wrapper for CoCoTeX kernel's `\cc@check@empty`

```
\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.

```
\def\ccf@compose@listof{%
  \ccf@check@empty{Number}%
  \ccf@check@empty{Caption}%
  \ccf@check@empty{Legend}%
  \ccf@check@empty{Source}%
  \let\ccf@listof@entry\relax
  \ccWhenComp{ListofCaption}{\csgappto{ccf@listof@entry}{\string\ccComponent{ListofCaption}{\
      ccUseComp{ListofCaption}}}}%
  \ccWhenComp{ListofNumber}{\csgappto{ccf@listof@entry}{\string\ccComponent{ListofNumber}{\
      ccUseComp{ListofNumber}}}}%
  \ccWhenComp{ListofLegend}{\csgappto{ccf@listof@entry}{\string\ccComponent{ListofLegend}{\
      ccUseComp{ListofLegend}}}}%
  \ccWhenComp{ListofSource}{\csgappto{ccf@listof@entry}{\string\ccComponent{ListofSource}{\
      ccUseComp{ListofSource}}}}%
  \ifx\ccf@listof@entry\relax\else
   \bgroup
     \ccGobble
     \ccaProtect
     \protected@edef\@ccf@listof@entry{\ccf@listof@entry}%
     \ccComponentEA{ListofEntry}{\@ccf@listof@entry}%
   \egroup
  \fi
}%
```

`\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.

```
\def\ccf@write@listof{%
  \ccUnlessAttr{\ccfCapType}{nolist}
   {\ifnum\ccSubFloatCnt=\z@\relax
      \ccIfAttr{\ccfCapType}{subfloat}
        {\ccSubFloatCnt=\z@\relax
         \cc@iterate{\ccSubFloatCnt}{\z@}{\the\ccf@int@sub@flt@cnt}
           {\ccf@addcontentsline}}%
        {\ccf@addcontentsline}%
    \else
      \ccIfAttr{\ccfCapType}{subfloat}{}{\ccf@addcontentsline}%
    \fi}%
}
```

3.7 Label and Referencing mechanisms

Generation of Number Components

`\ccf@create@counter` checks for the various parameters that control whether or not a Number component is auto-generated for each sub-float.

```
\def\ccf@create@counter{%
  \ccIfAttrIsSet{\ccfCapType}{nonumber}{}
    {\ccUnlessComp{Number}
      {\ccIfPropVal{numbering}{auto}
        {\ccIfAttr{\ccfCapType}{subfloat}
          {\ifnum\ccSubFloatCnt=\z@\relax
             \ccf@set@top@counter%
           \else
             \ccIfPropVal{sub-numbering}{auto}
               {\ccf@set@subcounter}{}%
           \fi}
          {\ccf@set@top@counter}}{}}}}
```

`\ccf@set@top@counter` generates first level float counter.

```
\def\ccf@set@top@counter{%
  \ccWhenComp{Caption}{%
    \global\expandafter\advance\csname c@\ccfCapType\endcsname\@ne\relax
    \ccdefFromProperty\ccf@name@prefix{auto-number-prefix}%
    \ccdefFromProperty\ccf@name@sep{auto-number-prefix-sep}%
    \protected@edef\@tempa{\ccf@name@prefix\ccf@name@sep\expandafter\the\csname c@\ccfCapType\
        endcsname}%
    \ccComponentEA{Number}{\@tempa}%
  }%
}
```

`\ccf@set@subcounter` generates second level counters for numbered sub-floats. #1 is the sub-float counter.

```
\def\ccf@set@subcounter{%
```

`float-number` `<any>` the counter of a first-level float

```
  \ccSetPropertyVal{float-number}{\csname cc@\cc@cur@cont @Number-0\endcsname}%
```

`sub-number` `<any>` the counter of a second-level float

```
  \ccSetPropertyVal{sub-number}{%
    \begingroup
      \expandonce{\ccUseProperty{sub-number-face}}%
      \relax\ccUseProperty{sub-number-before}%
      \csname @\ccUseProperty{sub-number-style}\endcsname{\the\ccSubFloatCnt}%
      \ccUseProperty{sub-number-after}%
    \endgroup}%
  \ccComponent{Number}{\ccUseProperty{sub-number-format}}%
}
```

Generation of LaTeX Labels

`\ccfCreateLabel` creates labels

```
\def\ccfCreateLabel{%
  \ccIfComp{Number}
    {\def\cc@fallback@anchor{%
       \ccGobble
```

```
      \ccdefFromComp\@currentlabel{Number}%
      \ccdefFromComp\@currentlabelname{ListofCaption}}%
    \def\cc@labelname@comp{Caption}}
   {\def\cc@fallback@anchor{\phantomsection}}%
  \expandafter\ccCreateLabel\expandafter{\ccfCapType}}
```

3.8 Processing the Float

Sizes, Spacing and Margins

`\ccf@set@hsize` calculates the available maximum width for the float contents and captions according to the values of the `margin-right` and the `margin-left` properties.

```
\def\ccf@set@hsize{%
  \expandafter\ccf@sub@sep\ccUseProperty{sub-float-sep}\relax%
  \global\ccf@total@width=\hsize\relax
  \expandafter\ccf@margin@l\ccUseProperty{margin-left}\relax
  \expandafter\ccf@margin@r\ccUseProperty{margin-right}\relax
  \expandafter\ccf@margin@i\ccUseProperty{margin-inner}\relax
  \expandafter\ccf@margin@o\ccUseProperty{margin-outer}\relax
  \ccf@set@margins
  \global\advance\ccf@total@width-\ccf@margin@r\relax
  }
```

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

```
\def\ccf@set@margins{%
  \ccTestPage
  \if@cc@odd
    \advance\ccf@margin@l\ccf@margin@i
    \advance\ccf@margin@r\ccf@margin@o
  \else
    \advance\ccf@margin@l\ccf@margin@o
    \advance\ccf@margin@r\ccf@margin@i
  \fi
}
```

Processing the Contents of the Float Environment

`\ccf@process` calculates the dimensions of the content of a float environment (including captions and spacing) and eventually prints the contents using the `float-render` and `subfloat-render` Properties.

```
\def\ccf@process{%
  \ifx\ccf@has@capt@top\@empty\leavevmode\fi
  \ccf@make@outer@caption{top}%
  \ifnum\the\ccSubFloatCnt=\z@\relax
    \bgroup\advance\hsize-\ccf@margin@l
      \@cc@is@finaltrue
      \ccUseProperty{float-render}%
    \egroup
  \else
    \ccf@test@subcapt
    \ccf@calc@sameheight
    \def\ccf@prefix{sub}%
    \ifx\ccf@has@subcapt@top\@empty\ccf@calc@row@ht{top}\fi%
    \ifx\ccf@has@subcapt@bottom\@empty\ccf@calc@row@ht{bottom}\fi%
    \bgroup
      \advance\hsize-\ccf@margin@l
      \@cc@is@finaltrue
      \ccUseProperty{subfloat-render}%
    \egroup
    \let\ccf@prefix\@empty
  \fi
  \ccf@make@outer@caption{bottom}%
}
```

\ccf@calc@row@ht calculates the heights of all captions in the same row.

{#1} determins if the top or bottom row is calculated.

```
\def\ccf@calc@row@ht#1{%
  \@tempcnta\z@
  \@tempdima\z@
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \setbox\z@\vbox{%
      \ccSubFloatCnt\@tempcnta\relax
      \expandafter\hsize\expandafter\dimexpr\csname cc@\cc@cur@cont @res@width-\the\@tempcnta\
          endcsname\relax
      \ccGobble
      \ccUseProperty{\ccf@prefix caption-face}%
      \ccUseProperty{\ccf@prefix caption-face-#1}%
      \leavevmode
      \strut\ccUseProperty{caption-#1}\strut%
      }%
    \expandafter\ifdim\dimexpr\ht\z@+\dp\z@\relax>\@tempdima \@tempdima\dimexpr\ht\z@+\dp\z@\
        relax\fi
  }%
  \expandafter\edef\csname ccf@capt@row@height@#1\endcsname{\the\@tempdima}%
}
```

\ccf@calc@sameheight calculates the target width of each sub-image in the Figure Container if each of the sub-images is required to match a uniform height.

```
\def\ccf@calc@sameheight{%
  \if@ccf@sameheight
```

If all sub figures should be scaled to the same height, we reserve two registers: dimension \@tempdima, which holds the cumulated widths of all adjusted sub images, and \@tempcnta, which serves as a temporary index for each processed sub float.

```
    \@tempdima=\z@\relax
    \@tempcnta=\z@\relax
```

The \ccf@calc@width dimension register holds the target width of the total sub float block. It is initialized to \ccf@total@width and reduced by the amount of the left margin.

```
    \ccf@calc@width=\ccf@total@width\relax
    \advance\ccf@calc@width-\ccf@margin@l\relax
```

now, we iterate through the sub floats, storing the current index in \@tempcnta.

```
    \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
```

First, we calculate the ratio between the height of the largest image in the Figure container, and the height of the current image, resulting in \@tempa being a rational number ≥ 1.

```
      \edef\@tempa{\CalcRatio{\ccf@sub@maxheight}{\csname ccf@\cc@cur@cont @height-\the\@tempcnta
          \endcsname}}%
```

Now, we subtract the mandatory space between to sub images (as indicated by the Property) to the total target width \ccf@calc@width.

```
    \ifnum\the\@tempcnta>\@ne\relax
      \advance\ccf@calc@width-\ccf@sub@sep\relax%
    \fi
```

Temporary length register \@tempdimc holds the natural width of the current image.

```
    \expandafter\@tempdimc\csname ccf@\cc@cur@cont @width-\the\@tempcnta\endcsname\relax
```

Temporary length register \@tempdimb holds the width of the down-scaled image. The scaling factor is \@tempa calculated earlier.

```
    \@tempdimb=\@tempa\@tempdimc\relax
```

This values is stored in cc@\cc@cur@cont @adj@width-\the\@tempcnta for each image.

```
    \csedef{cc@\cc@cur@cont @adj@width-\the\@tempcnta}{\the\@tempdimb}%
```

This width is added to the \@tempdima register.

```
    \advance\@tempdima\@tempdimb
  }%
```

Once we calculated the new width of each image, we reset the temporary sub-float counter \@tempcnta, and the temporary length registers \@tempdimb and \tempdimc.

```
  \@tempcnta=\z@\relax
  \@tempdimb=\z@\relax
  \@tempdimc=\z@\relax
```

Up to this point, we calculated the ratio of each image in relation to the laregst image and what width each image would have if we scaled it *up* to this maximum height. As a by-product, we also calculated the target width of all sub images by substracting the separator space inserted between each image.

Now, we need to calculate the acual scaling factor. For that, we loop through all subfloats once more, re-using the temproary counter \@tempcnta as index of the current image.

```
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
```

\@tempdima stores the sum of the widths of all images if they were upscaled to match the highest image. We now calculate the ratio between the width of the current, upscaled, image and and sum of the widths of all upscaled images. The result is stored in tempa as a rational number between 0 and 1.

```
    \edef\@tempa{\CalcRatio{\csname cc@\cc@cur@cont @adj@width-\the\@tempcnta\endcsname}{\
        @tempdima}}%
```

We now use this factor to calculate the target width of the sub float by multiplying this scaling factor with the actually available width of the Container stored in \@ccf@calc@width.

```
    \csedef{cc@\cc@cur@cont @res@width-\the\@tempcnta}{\dimexpr\@tempa\ccf@calc@width\relax}%
```

We now store the natural height of the current image in \@tempdimc…

```
    \@tempdimc\dimexpr\csname ccf@\cc@cur@cont @height-\the\@tempcnta\endcsname\relax
```

…and also multiply it by the scaling factor such that \@tempdimc now holds the actual height of the current image after down-scaling.

```
    \@tempdimc\dimexpr\@tempa\@tempdimc\relax
```

\@tempdimb stores the total height of the down-scaled images. If all calculations are correct, \@tempdimb should be equal for each iteration.

```
    \ifdim\@tempa\@tempdimb<\@tempdimc\relax
      \@tempdimb=\@tempdimc\relax
    \fi
  }%
\else
```

If images should not be scaled to the same height, we simply set the target width of each sub image to its natural value.

```
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \csletcs{cc@\cc@cur@cont @res@width-\the\@tempcnta}{ccf@\cc@cur@cont @width-\the\@tempcnta}
        %
  }%
\fi
```

In any case, the total height of the entire image Container is stored in cc@\cc@cur@cont @res@height.

```
\csedef{cc@\cc@cur@cont @res@height}{\the\@tempdimb}%
}
```

3.9 Caption mechanism

\ccf@test@caption tests if the current sub-float has any top or bottom caption that needs to be printed.

#1	is the value of the sub-float counter
#2	indicates if the caption belongs to the whole float (empty) or a sub-float (sub)
#3	top or bottom

We compare the caption of the current \SubCounter level with a caption of a non-existing, negative, float level in case there is non-expandable material hard-coded into the caption-#3 Property. If we were to compare the width of the \hbox with \z@, this scenario would give us false positives.

Warning: Long captions can cause the hbox's width to exceed \maxdimen. To avoid 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!

```
\def\ccf@test@caption#1#2#3{%
  \ccaDisable\@cc@is@finalfalse
  \setbox\cc@tempboxa\hbox{\ccGobble\ccSubFloatCnt=0#1\relax\ccUseProperty{#2caption-#3}\relax}%
  \setbox\cc@tempboxb\hbox{\ccGobble\ccSubFloatCnt\m@ne\relax\ccUseProperty{#2caption-#3}\relax}
      %
  \edef\my@wda{\expandafter\strip@pt\wd\cc@tempboxa sp}%
  \edef\my@wdb{\expandafter\strip@pt\wd\cc@tempboxb sp}%
  \ifdim\my@wda>\my@wdb\relax
    \expandafter\global\expandafter\let\csname ccf@has@#2capt@#3\endcsname\@empty
  \fi
  \ccaEnable\@cc@is@finaltrue
}
```

\ccf@test@subcapt tests if the current float has any top or bottom captions that need to be printed

```
\def\ccf@test@subcapt{%
```

```
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \ccf@test@caption{\the\@tempcnta}{sub}{top}%
    \ccf@test@caption{\the\@tempcnta}{sub}{bottom}%
  }%
}
```

`\ccf@capt@top@offset` determines the spacing inserted **above both captions**.

```
\def\ccf@capt@top@offset#1{%
  \ccIfStrEqual{#1}{top}{}{%
    \par\if@ccf@break@capt\else\nopagebreak\fi%
    \expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-bottom}\relax%
    \advance\@tempskipa\dimexpr-\topskip+\dp\strutbox\relax
    \if@ccf@break@capt\advance\@tempskipa\dimexpr-\baselineskip-\ht\strutbox+\topskip\relax\fi
    \ifx\ccf@has@subcapt@bottom\@empty
      \ifnum\the\ccSubFloatCnt=\z@
        %% subcapt-bot exists and capt-bot is rendered
        \advance\@tempskipa\dimexpr\dp\strutbox\relax
        \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-bottom}\
            relax%
      \fi
    \fi
    \vskip\@tempskipa
    \leavevmode
  }}
```

`\ccf@capt@bottom@offset` determines the spacing inserted **below the captions**.

```
\def\ccf@capt@bottom@offset#1{%
  \ccIfStrEqual{#1}{top}
    {\@tempskipa=\z@\relax
     \expandafter\advance\expandafter\@tempskipa\ccUseProperty{\ccf@prefix caption-sep-top}%
     \ifnum\the\ccSubFloatCnt=\z@\relax
       \ifx\ccf@has@subcapt@top\@empty
         %% subcapt-top exists and capt-top is rendered
         \advance\@tempskipa\dimexpr\ht\strutbox-\topskip-\p@\relax
         \expandafter\advance\expandafter\@tempskipa\ccUseProperty{subcaption-add-sep-top}\relax%
       \else
         \advance\@tempskipa\dimexpr-\dp\strutbox\relax
       \fi
     \fi
     \vskip\@tempskipa
     \par\if@ccf@break@capt\else\nopagebreak\fi}
   {\ifnum\the\ccSubFloatCnt>\z@\relax
      \vskip\dp\strutbox
    \fi}}
```

`\ccf@make@caption` prints the caption.

`#1`	is the placement (`top`, `bottom`)
`#2`	is the vertical alignment (`top`, `middle`, `bottom`)

```
\long\def\ccf@make@caption#1#2{%
  \ccf@capt@top@offset{#1}%
  \ifnum\the\ccSubFloatCnt=\z@\relax
    \def\ccf@caption@box{%
      \ifx\ccf@landscape\@empty
        \setbox\@tempboxa\vbox\bgroup\hsize\textheight
      \else
```

```
        \hskip\ccf@margin@l%
         \setbox\@tempboxa\vbox\bgroup\advance\hsize-\ccf@margin@l%
      \fi}%
  \else
    \expandafter\cc@tempskipa\csname ccf@capt@row@height@#1\endcsname\relax
    \expandafter\advance\expandafter\cc@tempskipa\dimexpr-\baselineskip+\topskip\relax
    \def\ccf@caption@box{\setbox\@tempboxa\vbox to \cc@tempskipa\bgroup}%
  \fi
  \ccf@caption@box%
    \ccIfStrEqual{#2}{top}{}{\if@ccf@break@capt\else\vss\fi}%
    \ccUseProperty{\ccf@prefix caption-face}%
    \ccUseProperty{\ccf@prefix caption-face-#1}%
```

The caption as a whole is tagged with <Caption/>.

```
    \ccaStructStart{Caption}%
    \cc@topstrut\ccUseProperty{\ccf@prefix caption-#1}\strut%
    \ccaStructEnd{Caption}%
    \ifx\ccf@measure\relax\else
      \ccIfPropVal{label-pos}{#1}{%
        \ccfCreateLabel%
        \ccf@write@listof%
      }{}%
    \fi
    \ccIfStrEqual{#2}{bottom}{}{\if@ccf@break@capt\else\vss\fi}%
  \egroup%
  \if@ccf@break@capt\unvbox\@tempboxa\else\box\@tempboxa\fi%
  \ccf@capt@bottom@offset{#1}%
}
```

`\ccf@make@outer@caption` is a shell for the outer captions. #1 is the placement (`top` or `bottom`)

```
\def\ccf@make@outer@caption#1{%
```

now, we print the actual captions, if they contain contents.

```
  \expandafter\ifx\csname ccf@has@capt@#1\endcsname\@empty
    \setbox\z@\vbox{%
      \@cc@is@finalfalse
      \let\ccf@measure\relax
      \ccGobble
      \ccSubFloatCnt\z@
      \ccf@make@caption{#1}{top}%
    }%
    \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}}%
    \bgroup
      \savenotes
      \if@ccf@break@capt\else\nopagebreak\fi
      \ccSubFloatCnt\z@
      \ccf@make@caption{#1}{top}%
      \spewnotes
    \egroup
    \ccIfStrEqual{#1}{top}{\if@ccf@break@capt\else\nopagebreak\fi}{}%
  \else
```

If no caption at `{#1}` is given, we need to issue a `\par`, which otherwise would come from `\ccf@capt@top@offset`:

```
    \par
  \fi}
```

`\ccf@make@subcaption` creates the caption for subfloats. #1 is the position (`top` or `bottom`).

```
\def\ccf@make@subcaption#1{%
  \expandafter\ifx\csname ccf@has@\ccf@prefix capt@#1\endcsname\@empty
    \ccf@make@caption{#1}{\ccUseProperty{\ccf@prefix caption-valign-#1}}%
  \fi}
```

4 Generic User-Level Float Containers

`\ccDeclareFloat` is a user-level macro used to declare a new `ccFloat` environment.

`[#1]`	Name of the float Container from which the declared Container should inherit Properties (*optional*)
`{#2}`	top-level name of the float environment (e.g., `\ccPrefix Table`, `\ccPrefix Figure`)
`{#3}`	caption type (e.g., `table`, `figure`)
`{#4}`	list (e.g., `lot`, `lof`)
`{#5}`	additional Component body, use this to add to Types or introduce custom Handlers to the Float Container.

```
\def\ccDeclareFloat{\cc@opt@empty\ccf@declare@float}
\long\def\ccf@declare@float[#1]#2#3#4#5{%
```

First, we check if the Container already exists. If so, we issue an error message. May we force the style programmers learn to make use of CoCoTEX's extensive toolbox.

```
  \ifcsdef{cc@container@#2}{%
    \ccPackageError{Float}{}
        {Attempt to re-define pre-existing float Container `#2'}
        {You cannot re-define an existing float Container. Use
\string\ccAddToType{<Type>}{#2}{<code>} to alter the #2 container!}}{}%
```

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

```
  \def\ccf@parent{#1}%
  \ccDeclareContainer{#2}{%
    \ccPackageInfo{Floats}{}{Declaring float `#2'}%
    \ifx\ccf@parent\@empty
      \ccInherit{Properties,Components,Attributes}{float}
    \else
      \ccInherit{Properties,Components,Attributes}{\ccf@parent}
    \fi
    \ccDeclareType{FloatEnvInfo}{%
      \ccSetContainer{#2}%
      \def\ccfCapType{#3}%
      \def\ccf@cap@list@type{#4}%
    }% /FloatEnvInfo
```

The macro actually defines two LaTeX environments; a normal one for one-column floats, and a starred one for page-wide floats in two-column mode.

```
    \ccDeclareEnv[#2]{\ccf@float}{\endccf@float}%
    \ccDeclareEnv[#2*]{\if@twocolumn\let\ccf@do@dbl\relax\else\fi\ccf@float}{\if@twocolumn\let\
        ccf@do@dbl\relax\fi\endccf@float}%
    \ccDeclareType{Components}{}%
    \ccDeclareType{Properties}{}%
```

Generating the Handlers for the list-of entries and define the corresponding `l@` macros

```
    \ccf@generate@listof@handlers{#4}{#3}{#2}%
    \bgroup
      \def\cc@cur@cont{#2}%
      \cc@init@l@[list-of]{#3}{0}{#3}% Generate listof-Entries for first level floats
      \cc@init@l@[list-of]{#4}{1}{sub#3}% Generate listof-Entries for sub-floats
    \egroup
    #5
  }% /container
}
```

5 Image Containers

5.1 Abstract Graphics Container

Graphic is an abstract Container that represents an image file.

```
\ccDeclareContainer{Graphic}{%
  \ccDeclareType{Components}{%
    \def\cc@counted@comp@scheme#1{#1-\the\ccSubFloatCnt}%
```

Fig holds the includegraphics with the path to and the options for the actual image file.

```
    \ccfMakeComp{Fig}%
```

AltText is the alternative text for accessibility.

```
    \ccfMakeComp{AltText}%
```

```
  }%
  \ccDeclareType{Properties}{}%
}
```

5.2 Floating Figure Container

Figure is the user-level Container for display-style images or image clusters including their respective captions. Figures may either be placed as free-standing in-situ blocks or as floats.

```
\ccDeclareFloat{Figure}{figure}{lof}{%
  \ccInherit{Properties,Components}{Graphic}%
  \ccDeclareType{Properties}{%
```

subfloat-same-height [true|false] Whether all images in subfloats sould be scaled to the same height (true) or not (false).

```
    \ccSetProperty{subfloat-same-height}{true}%
```

subfloat-content <any>

```
    \ccSetProperty{subfloat-content}{%
      \ifx\ccf@no@figs\relax
        \rule{0pt}{1pt}\rule{1pt}{0pt}%
      \else
        \ccUseComp{Fig}%
      \fi}%
```

float-render <any> figure specific output routine.

```
    \ccSetProperty{float-render}{\ccfFigureRender}%
```

subfloat-render <any> figure specific output routine for sub-floats.

```
    \ccSetProperty{subfloat-render}{\ccfSubFigureRender}%
  }%
}
```

5.3 Figure Output Routines

\ccfFigureRender tells the float Container how the main content Component if Figure-type Floats is to be rendered. It is called via the Property.

```
\def\ccfFigureRender{%
  \bgroup
    \ifx\ccf@landscape\@empty
      \hsize\dimexpr\textwidth-\ccf@margin@r-\ccf@margin@l\relax%
    \fi
    \let\includegraphics\ccf@includesubgraphics
    \hskip\ccf@margin@l
    \strut\ccUseComp{Fig}\strut
  \egroup}
```

\ccfSubFigureRender tells the abstract float Container how the main content Component of Figure-type sub-floats are to be rendered. It is called via the Property.

```
\def\ccfSubFigureRender{%
  \hskip\ccf@margin@l
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \ccfRenderSubFloats{\the\@tempcnta}{Fig}%
  }}
```

\ccf@includesubgraphics is an override of LaTeX's \includegraphics patched to adjust for maximum width and height, and to capture the alt key in order to pass it down to ltpdfa.

In order to capture multiple images in the same Figure Container (i.e. real or fake Sub-Floats), tagging as <Figure/> of images takes place here, as does assignment of alternative text(s). Those can be submitted by the AltText Component or by the alt key in the optional argument of \includegraphics. If both are given, the alt key takes precedence. If neither is given, a -- is inserted.

```
\def\ccf@includesubgraphics{\cc@opt@empty\@ccf@includesubgraphics}%
\def\@ccf@includesubgraphics[#1]#2{%
  \def\@igopts{max width=\hsize,max height=\vsize}%
  \ccIfPropVal{fix-dimen}{true}{\apptocmd\@igopts{,width=\hsize}{}{}}{}%
  \if!#1!\else
    \apptocmd\@igopts{,#1}{}{}%
  \fi
  \gdef\ccf@fig@path{#2}%
  \if@cc@is@final
    \ccaStructStart{Figure}%
    \ccaAddPlacement{Block}%
  \fi%
  \expandafter\ccf@ltx@includegraphics\expandafter[\@igopts]{#2}%
  \if@cc@is@final
    \ifx\cca@Gin@alt\@undefined\let\cca@Gin@alt\relax\fi
    \ifx\relax\cca@Gin@alt\relax
    \ccIfComp{AltText}
      {\ccSanitizeStr\@cca@Gin@alt{\ccUseComp{AltText}}}
      {\def\@cca@Gin@alt{--}}%
  \else
    \ccSanitizeStr\@cca@Gin@alt{\cca@Gin@alt}%
  \fi
  \ccaAddAltText{\@cca@Gin@alt}%
  \ccaStructEnd{Figure}%
 \fi
}
```

`\ccf@measuresubgraphics` is an override of LaTeX's `\includegraphics` that is used to measure the natural dimensions of the included image. It also checks if the `\includegraphics` has either an height or width explicitly given. if so, we de-activate the same-height calculations for the entire float.

```
\def\ccf@measuresubgraphics{\cc@opt@empty\@ccf@measuresubgraphics}
\def\@ccf@measuresubgraphics[#1]#2{%
 \begingroup
  \setkeys{Gin}{#1}%
  \ifx\Gin@ewidth\Gin@exclamation
   \ifx\Gin@eheight\Gin@exclamation\else
    \global\@ccf@sameheightfalse
   \fi
  \else
   \global\@ccf@sameheightfalse
  \fi
 \endgroup
 \ccf@ltx@includegraphics[#1]{#2}%
}
```

5.4 Inline Figures

Inline Figure Container

`InlineFigure` is the user-level Container for inline graphics (e. g., images in tables or symbols inside the main text body). Note that this Container is *not* derived from the abstract `float` Container. Also, there is no LaTeX environment for that Container but a simple macro.

```
\ccDeclareContainer{InlineFigure}{%
 \ccInherit{Properties,Components}{Graphic}%
 \ccDeclareType{Attributes}{}%
 \ccDeclareType{Properties}{%
```

`smash` `[true|false]` whether the image is allowed to stretch the line it is in (false) or not (true) if the height exceeds `\baselineskip`.

```
  \ccSetProperty{smash}{false}
```

`vertical-align` `[top|middle|bottom]` the vertical alignment of the inline image relative to the baseline of the surrounding text. If the value is `bottom`, the bottom border of the image is aligned with the baseline, `top` aligns the top border of the image at baseline + `\ht\strutbox`, `middle` centers the image at baseline + $0.5 \times$ `\ht\strutbox`.

```
  \ccSetProperty{vertical-align}{bottom}
```

`float-render` `<any>` specific output routine for inline figures

```
  \ccSetProperty{float-render}{\ccUseComp{Fig}}
 }%
```

```
}
```

Inline Figure User Macro

`\ccInlineFigure` is the Handler for an inline figure's main content Component.

`[#1]`	is the attribute list for the figure
`{#2}`	is the Container Body

```
\def\ccInlineFigure{\cc@opt@empty\cc@inline@figure}
\def\cc@inline@figure[#1]#2{%
  \begingroup
    \ccSetContainer{InlineFigure}%
    \def\ccfCapType{figure}%
    \ccToggleCountedConditionals
    \ccEvalType{Properties}%
    \ccEvalAttributes{#1}%
    \ccf@eval@class
    \ccEvalType{Components}%
    \ignorespaces
    #2%
    \ccSubFloatCnt=\z@\relax
    \bgroup
      \ccUseProperty{float-render}%
    \egroup
    \ccf@debug%
    \ccf@store@dimens
  \endgroup
}
\csdef{\ccPrefix InlineFigure}{\ccInlineFigure}%
```

6 Table Containers

6.1 The Abstract Tabular Container

CoCoTEX's float module supports the three basic Standard LaTeX tabular environments (*tabular*, `tabularx` and `tabulary`) as well as `htmltab` from the `htmltabs` package. For the measuring to work correctly, we need to render the tables as a whole and store the result inside `\ccf@floatbox` for measuring and further processing.

`Tabular` is an abstract Container that represents raw table data. Its main purpose is to provide a unified interface to patch some of LaTeX's standard *tabular* environments, as well as the `htmltab` environment, it the `htmltabs` package is loaded.

```
\ccDeclareContainer{Tabular}{%
  \ccDeclareType{Properties}{}%
  \ccDeclareType{Components}{%
    \ccf@reserve@tabular
  }%
}
```

`\ccf@reserve@tabular` is a shell macro that temporarily stores the default macro definitions for various tabular environments and patches them such that the contents are stored inside the `\ccf@floatbox`. The macro is called at the very beginning of the Table Container's environemnt and the patches only hold inside that environment. Thus, all tabular environments can be used in their vanilla state outside CoCoTEX's Table environments.

```
\def\ccf@reserve@tabular{%
  \ccf@reserve@tab{}%
  \ccf@reserve@tab{x}%
  \ccf@reserve@tab{y}%
  \ccf@reserve@htmltab%
}
```

`\ccf@reserve@tab` stores the default definitions for a specific vanilla-LaTeX 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.

```
\def\ccf@reserve@tab#1{%
  \csletcs{orig@tabular#1}{tabular#1}%
  \csletcs{orig@endtabular#1}{endtabular#1}%
  \csdef{tabular#1}{%
    \global\setbox\ccf@floatbox
    \vbox\bgroup
      \if!#1!\else
        \let\tabular\orig@tabular
        \let\endtabular\orig@endtabular
      \fi
      \csname orig@tabular#1\endcsname}%
  \csdef{endtabular#1}{\csname orig@endtabular#1\endcsname\egroup}%
}
```

6.2 The User-Level Table Container

`Table` is a user-level Container for display-style tables including their captions. They may wither be places as free-standing in-situ blocks or as floats.

```
\ccDeclareFloat{Table}{table}{lot}{%
  \ccInherit{Properties,Components}{Tabular}%
  \ccDeclareType{Properties}{%
    \ccSetProperty{subcaption-valign-top}{bottom}%
    \ccSetProperty{subfloat-content}{%
      \PackageError{coco-floats.sty}
        {ccSubFloat does not support sub-tables (yet)!}
        {You cannot yet use a tables within the `ccSubFloat'!}%
      }%
    \ccSetProperty{float-render}{\ccfTableRender}%
    \ccSetProperty{subfloat-render}{\ccfSubTableRender}%
  }%
}
```

`\ccf@reserve@htmltab` special handler for tables using the `htmltabs` package:

```
\AtBeginDocument{%
  \@ifpackageloaded{htmltabs}{%
    \def\ccf@reserve@htmltab{%
      \let\ccf@add@style\@empty
      \ifx\ccf@floatpos\@empty
        \expandafter\ifx\csname ccFloat\the\ccf@int@cnt Captop\endcsname\relax\else
          \htInitSkip\csname ccFloat\the\ccf@int@cnt Captop\endcsname
          \advance\htInitSkip\ccf@sep@top%
        \fi
        \expandafter\ifx\csname ccFloat\the\ccf@int@cnt Capbottom\endcsname\relax\else
          \htAddToBottom\csname ccFloat\the\ccf@int@cnt Capbottom\endcsname
          \advance\htAddToBottom\ccf@sep@bottom%
      \fi
     \else
      \def\ccf@add@style{;break-table:false;}%
     \fi
     \edef\cc@tempa{margin-left:\ccf@margin@l\ccf@add@style}%
     \expandafter\htAddStyle\expandafter{\cc@tempa}%
     \global\setbox\htTableBox\box\voidb@x
     \let\htOutputTable\relax
   }}{\let\ccf@reserve@htmltab\relax}%
}
```

6.3 The Table Output Handler

\ccfGetTableContent returns the \ccf@floatbox if it is not un-itialized or void.

```
\def\ccfGetTableContent{%
  \ifx\htTableBox\@undefined\else
   \ifvoid\htTableBox\else
    \let\ccf@floatbox\htTableBox%
   \fi\fi}
```

\ccfTableRender is the content of the Property specific for tables.

```
\def\ccfTableRender{%
  \ccfGetTableContent
  \ccComponent{Content}{\unvbox\ccf@floatbox}%
  \ccUseComp{Content}%
  \ccaStructStart{Table}%
  \ifx\ht@structID@THead\@undefined
   \ccaMoveChildren{\ht@structID@TBody}%
  \else
   \ccaMoveStruct{\ht@structID@THead}%
   \ifx\ht@structID@TBody\@undefined\else\ccaMoveStruct{\ht@structID@TBody}\fi%
  \fi
  \ifx\ht@structID@TFoot\@undefined\else\ccaMoveStruct{\ht@structID@TFoot}\fi%
  \par\if@ccf@break@capt\else\nopagebreak\fi
  \vskip\dp\strutbox
  \ccaStructEnd{Table}%
}
```

\ccfSubTableRender Is the content of the table-specific Property

Note that table sub-floats aren't allowed yet, so this definition is un-used at the moment. TeX will crash with an error message before this Property is ever expanded.

```
\def\ccfSubTableRender{%
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
   \ccfGetTableContent
   \@cc@is@finalfalse
   \ccComponent{Content}{\unvbox\ccf@floatbox}%
   \@cc@is@finaltrue
   \ccfRenderSubFloats{\the\@tempcnta}{Content}%
  }}
```

7 Other Float-Related Macros

`\ccFloatBarrier` can be used to force all pending floats to be printed at the next shipout.

```
\def\ccFloatBarrier{\AtBeginShipoutNext{\clearpage}}
```

Output Driver for the `coco-floats.sty`.

```
</floats>
```

Module 11

coco-frame.dtx

This file provides facilities to visualise crop marks and the print area.

```
<*frame>
```

```
%%
%% module for CoCoTeX for crop marks and print area frames.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-frame}
   [2024/12/13 v0.5.0 coco-frame]\relax
```

1 Top-Level Interface

```
\let\cc@frame@mode n
\ExplSyntaxOn
\keys_define:nn { cocotex/frame }
{
  frame .choice:,
  frame / none .code:n = { \global\let\cc@frame@mode n }
  frame / crop .code:n = { \global\let\cc@frame@mode p }
  frame / frame .code:n = { \global\let\cc@frame@mode w }
}
\ProcessKeyOptions[cocotex/frame]
\ExplSyntaxOff
```

2 Cropmark printer

```
\ifx\cc@frame@mode p\relax
  \ifx\bleed\@undefined \newdimen\bleed \bleed4mm\relax\fi
  \ifx\cc@frame@@offset\@undefined \newdimen\cc@frame@@offset \cc@frame@@offset4em\relax\fi
  \voffset\dimexpr\cc@frame@@offset-1in\relax
  \hoffset\dimexpr\cc@frame@@offset-1in\relax
  \edef\l@offset{\strip@pt\dimexpr\cc@frame@@offset*7200/7227\relax}
  \edef\r@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperwidth)*7200/7227\relax}
  \edef\u@offset{\strip@pt\dimexpr(\cc@frame@@offset)*7200/7227\relax}
  \edef\o@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperheight)*7200/7227\relax}
  \edef\b@l@offset{\strip@pt\dimexpr(\cc@frame@@offset-\bleed)*7200/7227\relax}
  \edef\b@r@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperwidth+\bleed)*7200/7227\relax}
  \edef\b@u@offset{\strip@pt\dimexpr(\cc@frame@@offset-\bleed)*7200/7227\relax}
  \edef\b@o@offset{\strip@pt\dimexpr(\cc@frame@@offset+\paperheight+\bleed)*7200/7227\relax}
  \edef\@tempa{%
    /TrimBox [\l@offset\space\u@offset\space\r@offset\space\o@offset]
    /BleedBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
    %/CropBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
    %/MediaBox[\b@l@offset\space\b@u@offset\space\b@r@offset\space\b@o@offset]
  }
  \expandafter\pdfpageattr\expandafter{\@tempa}
\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:

```
\@ifundefined{pdfpagewidth}{%
  \RequirePackage{luatex85}
  \pdfpagewidth\paperwidth
  \pdfpageheight\paperheight
}{}
```

Setting PDF boundaries

```
\ifx\cc@frame@mode n\relax\else
  \ifx\cc@frame@mode p\relax
    \edef\stockwidth{\the\dimexpr\paperwidth+\cc@frame@@offset+\cc@frame@@offset\relax}
    \edef\stockheight{\the\dimexpr\paperheight+\cc@frame@@offset+\cc@frame@@offset\relax}
  \fi
```

Cropmarks and page area frames both are painted via the `crop` package.

```
  \RequirePackage{crop}
  \renewcommand*\CROP@marks{%
    \CROP@setmarkcolor
    \CROP@user@b
    \vskip1in\hskip1in\relax
    \CROP@ulc\null\hfill\CROP@@@info\CROP@upedge\hfill\null\CROP@urc\hskip-1in\null
    \vfill
    \CROP@ledge\hfill\CROP@redge
    \vfill
    \hskip1in\relax
    \CROP@llc\null\hfill\CROP@loedge\hfill\null\CROP@lrc\hskip-1in\null
    \vskip-1in}%
  \ifx\cc@frame@mode p\relax
    \def\camcross{%
      \smash{\rlap{%
          \kern-0.15\p@
          \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
          \kern-0.15\p@
          \kern-1.7mm\relax
          \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax
          \kern-0.3\p@
          \raise1.7mm\rlap{\vrule\@width3.4mm\@height\z@\@depth0.3\p@}%
          \lower1.7mm\rlap{\vrule\@width3.4mm\@height0.3\p@\@depth\z@}%
          \hbox{\vrule\@width3.4mm\@height0.15\p@\@depth0.15\p@}%
          \kern-0.3\p@
          \vrule\@width0.3\p@\@height1.7mm\@depth1.7mm\relax}}}
    \def\cammcrossleft{%
      \llap{\camcross\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\kern\
          bleed}}
    \def\cammcrossright{%
      \rlap{\kern\bleed\vrule\@width\dimexpr\bleed+2mm\relax\@height0.15\p@\@depth0.15\p@\
          camcross}}
  \def\cammcrossup{%
    \rlap{\smash{\raise\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
        \kern-0.15\p@\vrule\@width0.3\p@\@height\dimexpr\cc@frame@@offset-2mm\relax\@depth-\bleed}}}
  \def\cammcrossdown{%
    \rlap{\smash{\lower\dimexpr\cc@frame@@offset-2mm\relax\hbox{\camcross}%
        \kern-0.15\p@\vrule\@width0.3\p@\@height-\bleed\@depth\dimexpr\cc@frame@@offset-2mm\relax}}}
  \def\CROP@@ulc{\cammcrossup\cammcrossleft}
  \def\CROP@@urc{\cammcrossup\cammcrossright}
  \def\CROP@@llc{\cammcrossdown\cammcrossleft}
  \def\CROP@@lrc{\cammcrossdown\cammcrossright}
  \renewcommand*\CROP@@info{{%
      \global\advance\CROP@index\@ne
      \def\x{\discretionary{}{}{\hbox{\kern.5em---\kern.5em}}}%
      \ifx\CROP@pagecolor\@empty
      \else
        \advance\dimen@\CROP@overlap
      \fi
      \hb@xt@\z@{%
        \hss
        \lower1em\vbox to\z@{\vss
          \centering
          \hsize\dimexpr\paperwidth-20\p@\relax
          \normalfont
          \large
          \vskip5mm\relax
          \addvspace{\bleed}}%
        \hss}}%
  }%
  \crop[cam]
```

the code for the page area frame

```
\else% w
  \@tempdima\dimexpr\textheight\relax
  \divide\@tempdima by\baselineskip
  \multiply\@tempdima by65536\relax
  \edef\cnt@baselines{\strip@pt\@tempdima}%
  \def\cc@frame@lines{%
    \@tempcnta\z@
    \loop\advance\@tempcnta\@ne
      \hsize1em\relax
      \ifodd\count\z@
        \vrule\@width1em\@height0.2\p@\@depth0.02\p@
        \llap{\smash{\the\@tempcnta\,}}%
      \fi%
      \rlap{%
        \ifodd\count\z@\else\fi
        \vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
        \if@twocolumn
          \kern\columnsep\vrule\@width\columnwidth\@height0.00005\p@\@depth0\p@
        \fi
        \ifodd\count\z@\else
          \vrule\@width1em\@height0.00005\p@\@depth0\p@%
          \llap{\smash{\the\@tempcnta\,}}%
        \fi
      }%
      \break
    \ifnum\@tempcnta<\cnt@baselines
    \repeat}
    \def\cc@frame@margin{%
      \vrule height\textheight%
      \hskip-\marginparwidth\relax
      \vbox to\textheight{\hsize\marginparwidth\relax
        \rlap{\vbox to\z@{\hrule width\marginparwidth}}%
        \null\vss
        \rlap{\vbox to\z@{\hrule width\marginparwidth}}%
      }%
      \vrule height\textheight%
    }
    \renewcommand*\CROP@@frame{%
      \vskip0in%
      \color[cmyk]{0.4,0,0,0}%
      \ifodd\count\z@\let\@themargin\oddsidemargin\else\let\@themargin\evensidemargin\fi
      \advance\@themargin1in
      \moveright\@themargin
      \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@
        \vskip\topmargin\vbox to\z@{\vss\hrule width\textwidth}%
        \vskip\headheight\vbox to\z@{\vss\hrule width\textwidth}%
        \vskip\headsep\vbox to\z@{\vss\hrule width\textwidth}%
        \hbox to\textwidth{%
          \ifodd\count\z@
            \rlap{\hskip\dimexpr\textwidth+\marginparsep+\marginparwidth\relax\cc@frame@margin}%
          \else
            \rlap{\hskip-\marginparsep\relax\cc@frame@margin}%
          \fi
          \llap{\vbox to\textheight{\tiny\let\@tempa\f@size\normalsize\let\f@size\@tempa\
              selectfont
            \vskip\topskip\cc@frame@lines\null\vss}}%
          \llap{\vrule height\textheight}%
          \if@twocolumn
            \hskip\columnwidth\rlap{\vrule height\textheight}%
            \hskip\columnsep\rlap{\vrule height\textheight}%
          \fi
          \hfil\vrule height\textheight
        }%
        \vbox to\z@{\vss\hrule width\textwidth}%
        \vskip\footskip\vbox to\z@{\vss\hrule width\textwidth}%
        \vss}%
      \vbox to\z@{\baselineskip\z@skip\lineskip\z@skip\lineskiplimit\z@%
        \vskip-0in\rlap{\hskip1in%
          \vbox to\z@{\vbox to\z@{\vss\hrule width\paperwidth}%
            \hbox to \paperwidth{\llap{\vrule height\paperheight}\hfil%
              \vrule height\paperheight}%
            \vbox to\z@{\vss\hrule width\paperwidth}%
            \vss}}\vss}}
    \crop[frame,noinfo]%
  \fi
\fi
```

```
</frame>
```

Module 12
coco-lists.dtx

```
<*lists>
```

This module provides handlers for list-like environments like item lists, enumerations, glossaries and descriptions.

Note: The coco-lists module diverges somewhat from the other CoCoTEX modules insofar as that its main Container does not follow the CoCoTEX's usual ''collect all–process later'' approach, but all Properties are processed at the beginning of each Container's instances and the contents are processed as they are parsed by the \LaTeX interpreter, just like ''reguar'' LATEX lists. Configuration of lists, however, follows the CoCoTEX playbook.

1 Preamble

```
%%
%% Common document class for \textit{xerif} projects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
\NeedsTeXFormat{LaTeX2e}[2023/11/01]
\ProvidesPackage{coco-lists}
   [2024/12/13 v0.5.0 CoCoTeX lists module]
\RequirePackage{coco-common}
```

1.1 Package Options

If the replace option is set, LaTeX's default lists are replaced by coco-lists module. This effects LATEX's enumerate, itemize, and description environments.

```
\newif\if@ccl@replace \@ccl@replacefalse
\ExplSyntaxOn
\keys_define:nn { cocotex/lists }
{
  replace .code:n = {\global\@ccl@replacetrue}
}
```

The option inherit defines how nested lists inherit their properties. Currently, there are two ways: common: All nested lists of the same type inherit only from the same, generic type definition; conseq: nested lists of the same type inherit from the next-higher level list of the same type, and from the generic type definition.

For example, if inherit=common, 3rd level itemize and 2nd level itemize both inherit only the property values of the same generic itemize list type. If inherit=conseq, 3rd level inherits the property lists from 2nd level itemize.

Since inheritance is a transitive relation, 3rd level itemize will ultimately also inherit the Properties from generic itemize, but in contrast to common, conseq allows 2nd level itemize to override some Properties of generic itemize, which will be propagate down to 3rd level itemize, while with inherit=common, the override on 2nd level itemize would have no effect on 3rd level itemize.

`\ccl@ih@common` is used for comparisons. It represents the `inherit=common` package option.

```
\def\ccl@ih@common{common}
```

`\ccl@ih@conseq` is used for comparisons. It represents the `inherit=conseq` package option.

```
\def\ccl@ih@conseq{conseq}%
```

`\ccl@inherit` stores the value of the `inherit` package option.

```
\let\ccl@inherit\ccl@ih@common
\keys_define:nn { cocotex/lists }
{
  inherit .choice:,
  inherit / conseq .code:n = { \global\let\ccl@inherit\ccl@ih@conseq },
  inherit / common .code:n = { \global\let\ccl@inherit\ccl@ih@common },
  inherit .initial:n = common
}
```

`\ccl@str@local` is a string for comparison. It represents the `nesting=local` option.

```
\def\ccl@str@local{local}%
```

`\ccl@str@global` is a string for comparison. It represents the `nesting=global` option.

```
\def\ccl@str@global{global}%
```

`\ccl@nesting` The nesting option sets whether the nesting level of a list should be counted list-specific (value `local`), or globally (value `global`, default).

```
\let\ccl@nesting\ccl@str@global
\keys_define:nn { cocotex/lists }
{
 nesting .choice:,
 nesting / local .code:n = {\global\let\ccl@nesting\ccl@str@local },
 nesting / global .code:n = {\global\let\ccl@nesting\ccl@str@global },
 nesting .initial:n = global
}
```

```
\ProcessKeyOptions[cocotex/lists]
\ExplSyntaxOff
```

2 The List Container

`List` is the most abstract Container for lists.

```
\ccDeclareContainer{List}{%
```

2.1 List Properties

```
  \ccDeclareType{Properties}{%
```

List Boundaries

before-list <any> is expanded at the very beginning of a (nested) list.

```
\ccSetProperty{before-list}{% at the very beginning of each (nested) list
  \if@noskipsec \leavevmode \fi
  \ifvmode\else
    \unskip \par
  \fi
```

<L> is the opening List tag

```
  \ccaStructStart{L}%
}%
```

after-list <any> is expanded at the very end of a (nested) list. By default, it calls the after-item Property.
</L> is the closing List tag

```
\ccSetProperty{after-list}{%
  \ccUseProperty{after-item}%
  \ccaStructEnd{L}% end tag for the (nested) list
}%
```

List Margins

margin-top <skip> is the vertical skip at the beginning of each List instance.

```
%% list margins
\ccSetProperty{margin-top}{\z@}%
```

margin-bottom <skip> is the vertical skip at the end of each List instance.

```
\ccSetProperty{margin-bottom}{\z@}% vertical space before the list.
```

margin-left [auto|<skip>] is the horizontal space to the left of each list instance, from left boundary of the page area. auto means that the left margin is set to the width of widest label + prev-margin-left. The value is passed through \dimexpr, so basic arithmatic is allowed.

```
\ccSetProperty{margin-left}{\csname leftmargin\@roman\cclCurDepth\endcsname-\ccUseProperty{
    label-sep}+\ccUseProperty{prev-margin-left}}%
```

max-label-width <dimen> is the maximum space reserved for a list item's label.

```
\ccSetProperty{max-label-width}{.33\textwidth}%
```

margin-right <skip> is the right margin of the list instance.

```
\ccSetProperty{margin-right}{\z@}% horizontal space to the right of each list item
```

Between List Items

item-sep <skip> is the *vertical* space between two adjacent list items. Note that the real value value is advanced by the value of the par-skip Property.

```
\ccSetProperty{item-sep}{\z@}%
```

after-indent [true|false] determins whether the text paragraph after the (top-level) list is indented (true) or not (false).

```
\ccSetProperty{after-indent}{false}%
```

`at-begin-item-body` `<any>` is expanded right at the beginning of a new item body and sets the `<LBody>` tag.

```
\ccSetProperty{at-begin-item-body}{\ccaVstructStart{LBody}}%
```

`at-end-item-body` `<any>` is expanded at the very end of an item body, but before the final `\par`. By default, it only sets the closing `</LBody>` tag.

```
\ccSetProperty{at-end-item-body}{\ccaVstructEnd{LBody}}%
```

`after-item` `<any>` is expanded after each list item. It calls the `at-end-item-body` Property and closes the item's final paragraph as well as the `</LI>` tag.

```
\ccSetProperty{after-item}{%
  \ccUseProperty{at-end-item-body}%
  \ccaVstructEnd{LI}% Close list item tags
  \par}%
```

`before-item` `<any>` is called at the very beginning of each list item. If the current item is the first item, the `\ifcclFirst` conditional is set to false. All non-first items of the same List instance call the `after-item` Property and add a vertical skip of `item-sep` amount.

After that, the paragraph formatting parameters for the list-item `par-indent`, `par-skip`, and `par-fill-skip`, as well as the starting `<LI>` tag are set.

```
\ccSetProperty{before-item}{%
  \ifcclFirst
    \global\cclFirstfalse
  \else
    \ccUseProperty{after-item}%
    \vskip\ccUseProperty{item-sep}%
  \fi
  \parindent\ccUseProperty{par-indent}\relax%
  \parskip\ccUseProperty{par-skip}\relax%
  \parfillskip\ccUseProperty{par-fill-skip}\relax%
  \noindent
  \leavevmode
  \ccaVstructStart{LI}% Start tag for a list item
}%
```

`item-offset` `<any>` calculates `\cclItemIndent` from the `indent` and `label-sep` Properties and sets the horizontal offset of the first line of the list item. After that, the value of the macro is unsigned.

```
\ccSetProperty{item-offset}{%
  \cclItemIndent\ccUseProperty{indent}%
  \advance\cclItemIndent\dimexpr-\ccUseProperty{label-sep}\relax
  \hskip\cclItemIndent\relax%
  \ifdim\ccUseProperty{indent}>\z@
    \cclItemIndent\ccUseProperty{indent}%
  \else
    \cclItemIndent-\ccUseProperty{indent}%
  \fi
}%
```

`par-indent` `<skip>` is the indent of the first line of a *new* paragraph inside a list item

```
\ccSetProperty{par-indent}{\parindent}%
```

`par-fill-skip` `<skip>` is the skip at the end of the last line of each paragraph inside a list item

```
\ccSetProperty{par-fill-skip}{\@flushglue}%
```

par-skip <dimen> vertical space between two adjacent paragraphs inside the same List item

```
\ccSetProperty{par-skip}{\z@}%
```

Label Formatting

label <any> prints the Label component.

```
\ccSetProperty{label}{\ccUseComp{Label}}%
```

indent [auto|auto-global|<dimen>] is the indent of each List item's first line (relative to margin-left).

If the value is auto, the real indent and left margin of a item's first line is calculated using coco-common's indentation mechanism (see Sect. 3.3 in Module Module 3). The first-line indent will thereby be calculated from the widest width of all labels of the same list type and nesting level.

Note: the value auto-global is allowed, but it causes *all* lists – regarless of the nesting level – to have the same left margin and indent!

```
\ccSetProperty{indent}{-\dimexpr\csname leftmargin\@roman\cclCurDepth\endcsname-\
    ccUseProperty{label-sep}\relax}%
```

label-sep <dimen> is the horizontal space between the label and the item body.

```
\ccSetProperty{label-sep}{.5em}%
```

label-face <any> is the style of the label.

```
\ccSetProperty{label-face}{}%
```

label-align [left|center|right] is the alignment of the label within its local \hbox.

```
\ccSetProperty{label-align}{left}%
```

label-format <any> is the format of the label. It should call the label-face and label properties and enclose the latter with <Lbl> and </Lbl>.

```
\ccSetProperty{label-format}{%
  \ccUseProperty{label-face}%
  \ccaVstructStart{Lbl}%
  \ccUseProperty{label}%
  \ccaVstructEnd{Lbl}%
}%
```

label-box <any> is the property that builds a local \hbox into which the Label Component is printed. It should respect the label-align Property and call label-format.

```
\ccSetProperty{label-box}{%
  \hbox to \cclItemIndent{%
    \ccIfPropVal{label-align}{left}{}{\hss}%
    \ccUseProperty{label-format}%
    \ccIfPropVal{label-align}{right}{}{\hss}}%
}%
```

item-format <any> contains material printed at the beginning of a new item. It should call the before-item, item-offset, label-box and label-sep Properties.

```
\ccSetProperty{item-format}{%
  \ccUseProperty{before-item}%
  \ccUseProperty{item-offset}%
```

```
      \ccUseProperty{label-box}%
      \hskip\ccUseProperty{label-sep}%
    }%
  }%
```

2.2 List Components

```
  \ccDeclareType{Components}{%
```

`Label` represents a List item's local label.

```
    \ccDeclareComponent{Label}%
```

```
  }%
  \ccDeclareEnv{cc@list}{endcc@list}%
}
```

3 Declaring List Types

List Types are the next layer of abstraction for lists. This layer distinguishes numbered from unnnumbered and description lists.

`\DeclareListType` declares a new list type. #1 is the name of the list type, #2 is the declaration body. Each new list type should declare at least an Attribute handler and a Label handler. #3 is a list of type specific properties that are appended to the generic list's property list.

```
\long\def\ccDeclareListType#1#2#3{%
```

`\DeclareAttributeHandler` declares a new handler for a list's attributes. ##1 is the definition body.

```
  \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.

```
  \def\DeclareLabelHandler##1{\csdef{ccl@make@label@#1}{##1}}%
```

```
  \ccDeclareContainer{#1List}{%
    \ccInherit{Components,Properties}{List}%
    \ccDeclareType{Properties}{%
```

`list-type` `<any>` holds the name of the list type.

```
      \ccSetProperty{list-type}{#1}%
```

```
      #3%
    }%
    \ccDeclareEnv[#1-list]{\cc@list}{\endcc@list}%
  }%
  #2%
}
```

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.

```
\def\ccDeclareList#1#2#3{%
  \csxdef{cc@cur@depth@#1}{\z@}%
  \ccDeclareContainer{#1}{%
    \ccInherit{Properties,Components}{#2List}%
    \ccDeclareType{Properties}{#3}%
    \ccDeclareEnv[#1]{\cc@list}{\endcc@list}%
  }%
  \ccDeclareNested{#1}{\z@}{}%
}
```

`\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.

```
\def\ccDeclareNested#1#2#3{%
  \@tempcnta=#2\relax
  \ifx\@tempcnta<\z@\relax
    \ccPackageError{lists}{Nesting}{Invalid nesting level!}{You cannot declare nesting levels
        less than 0!}%
  \fi
  \advance\@tempcnta\@ne\relax
  \ccDeclareContainer{#1-\the\@tempcnta}{%
    \ifcsdef{cc@container@#1}
      {\ccInherit{Properties,Components}{#1}}
      {\ccPackageError{lists}{Inheritance}
        {List `#1' undefined!}
        {You need to define the list `#1' before you can declare nested list overrides!}}%
      \ccDeclareType{Properties}{#3}%
    }%
}
```

We want to count each list type seperately 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.

```
\newcount\ccl@depth
```

`\ccl@item@cnt` is the internal counter for the items within a (nested) list level.

```
\newcount\ccl@item@cnt
```

`\ifcclFirst` is true as long as the first item of a list is processed.

```
\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.

```
\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.

```
\def\ccl@advance@depth@global#1{%
  \edef\cclPrevDepth{\the\ccl@depth}%
  \global\advance\ccl@depth#1\relax
  \edef\cclCurDepth{\the\ccl@depth}%
}
```

`\ccl@advance@depth@local` is called when the nesting level should be counted for each list type individually.

```
\def\ccl@advance@depth@local#1{%
  \letcs\cclPrevDepth{cc@cur@depth@\cc@cur@cont}%
  \expandafter\@tempcnta\csname cc@cur@depth@\cc@cur@cont\endcsname\relax
  \advance\@tempcnta#1\relax
  \csxdef{cc@cur@depth@\cc@cur@cont}{\the\@tempcnta}%
  \edef\cclCurDepth{\csname cc@cur@depth@\cc@cur@cont\endcsname}%
  \global\advance\ccl@depth#1\relax
}
```

`\cclItemIndent` stores the actual calculated indent of an List item's first line.

```
\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.

```
\newcount\cclTopID \cclTopID\z@\relax
```

`\cclID` stores a unique ‘‘identifier’’ number for each list, irrespective their nesting levels.

```
\newcount\cclID \cclID\z@\relax
```

An internal global counter register `\ccl@total@list@cnt` is used to count the overall number of opening lists. Currently, the global ID of each list is unused.

```
\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.

```
\def\ccl@incr@count{%
  \csxdef{ccl@prev@cnt@\the\ccl@depth}{\the\cclID}%
  \global\advance\ccl@total@list@cnt\@ne\relax
  \global\cclID\ccl@total@list@cnt\relax
  \ifnum\cclCurDepth=\@ne\relax
    \global\advance\cclTopID\@ne\relax
  \fi
}
```

\ccl@decr@count resets the list counter for the next lower nesting level, whenever a nested list is closed.

```
\def\ccl@decr@count{%
  \global\cclID\csname ccl@prev@cnt@\the\ccl@depth\endcsname\relax
}
```

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.

```
\def\cc@list{\cc@opt@empty\@cc@list}
\def\@cc@list[#1]{%
  \ccl@advance@depth\@ne%
  \ccl@incr@count%
  \edef\ccl@cur@cont{\cc@cur@cont-\cclCurDepth}%
  \global\cclFirsttrue
```

If the nesting goes deeper than the style programmer anticipated:

```
  \ifcsdef{cc@container@\ccl@cur@cont}{}
   {\ifx\ccl@inherit\ccl@ih@common
      \let\ccl@cur@cont\cc@cur@cont%
    \else
      \global\csletcs
        {cc@type@Properties@\cc@cur@cont-\cclCurDepth}
        {cc@type@Properties@\cc@cur@cont-\cclPrevDepth}%
    \fi}%
```

Horizontal margin Properties from the previous nesting level are stored so that the nested lists can use them.

```
  \edef\ccl@leftskip{\the\dimexpr\leftskip\relax}%
  \edef\ccl@rightskip{\the\dimexpr\leftskip\relax}%
```

prev-margin-left <skip> stores the left margin of the next higher list level (i. e., the left margin of the list item that the current list is nested into)

```
  \ccSetPropertyX{prev-margin-left}{\ccl@leftskip}%
```

prev-margin-right <skip> stores the superior list item's right margin.

```
  \ccSetPropertyX{prev-margin-right}{\ccl@rightskip}%
  \ccEvalType[\ccl@cur@cont]{Properties}%
```

\ccl@list@type locally stores the current value of the list-type Property.

```
  \edef\ccl@list@type{\ccUseProperty{list-type}}%
```

Processing of the optional argument.

```
  \cclUseAttributeHandler{#1}%
```

The exact values of the margins are calculated.

```
  \cclCalculateMarginLeft%
  \cclCalculateVMargin{top}%
  \cclCalculateVMargin{bottom}%
```

`\Item` is a used to separate the single items of a list.

```
  \csdef{\ccPrefix Item}{\cc@opt@empty\ccl@item}%
  \def\ccl@item[##1]{%
    \protected@edef\ccl@item@label{##1}%
    \ifx\ccl@item@label\@empty
      \cclUseLabelHandler%
    \else
      \ccComponent{Label}{##1}%
    \fi
    \sbox\z@{\@cc@is@finalfalse\ccUseProperty{label-format}}%
    \@tempdima=\dimexpr\ccUseProperty{max-label-width}\relax
    \ifdim\wd\z@<\@tempdima\relax
      \@tempdima=\the\wd\z@\relax%
    \fi
    \bgroup
      \def\cc@cur@cont{list}%
      \cc@store@latest{\the\cclTopID-number-\cclCurDepth-maxwd}{\the\@tempdima}%
      \cc@store@latest{\the\cclTopID-number-maxwd}{\the\@tempdima}%
    \egroup
    \ccSetPropertyX{label-width}{\the\@tempdima}%
    \ccUseProperty{item-format}%
    \ccUseProperty{at-begin-item-body}\ignorespaces%
  }%
```

`\item` If default LaTeX macros are replaced per package option, `\item` is made into a copy of the local definition of `\ccPrefix Item`.

Warning: this might be dangerous when the User tries to embed something inside a CoCoTeX list that uses LaTeX's standard `\list` or `\trivlist` environments!

```
  \if@ccl@replace\letcs\item{\ccPrefix Item}\fi%
```

Up to this point, we only managed Properties. From this point forward, we actually print the list. We start by using the `before-list` Property.

```
  \ccUseProperty{before-list}%
```

then, we add the top vertical skip by `int-margin-top` amount.

```
  \ccUseProperty{int-margin-top}%
```

and set the left and right margins using the `margin-left`, `label-sep` and `margin-right` Properties.

```
  \leftskip\dimexpr\ccUseProperty{margin-left}+\ccUseProperty{label-sep}\relax%
  \rightskip\dimexpr\ccUseProperty{margin-right}\relax%
}
```

`\endcc@list` is called at the end of each List Container's respective environment. It basicly calls the `after-list` Property one last time, decrements the depth counter(s) and adds the `int-margin-bottom` vertical skip.

```
\def\endcc@list{%
  \ccUseProperty{after-list}%
  \ccl@decr@count%
  \ccl@advance@depth\m@ne%
  \ccUseProperty{int-margin-bottom}%
```

If the List is not nested, we eventually evaluate the `after-indent` Property.

```
  \ifnum\cclCurDepth=\z@\relax
    \ccIfPropVal{after-indent}{false}{%
      \global\@afterindentfalse
      \aftergroup\cc@afterbox}{}%
  \fi
}
```

`\cclCalculateVMargin` generates a macro that sets the internal margin Properties of the (nested) list. #1 is the orientation (`top` or `bottom`).

```
\def\cclCalculateVMargin#1{%
  \ifdim\ccUseProperty{margin-#1}=\z@\relax
    \ccSetProperty{int-margin-#1}{\relax}%
  \else
    \ccSetProperty{int-margin-#1}{\addvspace{\ccUseProperty{margin-#1}}}%
  \fi
}
```

`\cclCalculateLeftMargin` generates the value that `\leftskip` is set to.

```
\def\cclCalculateMarginLeft{%
  \ifcsdef{cc-list-\the\cclTopID-number-maxwd}
    {\ccSetPropertyVal{number-width-max}{\csname cc-list-\the\cclTopID-number-maxwd\endcsname}}
    {\ccSetPropertyVal{number-width-max}{1sp}}%
  \ifcsdef{cc-list-\the\cclTopID-number-\cclCurDepth-maxwd}
    {\ccSetPropertyVal{number-width-level-max}{\csname cc-list-\the\cclTopID-number-\cclCurDepth-maxwd\endcsname}}
    {\ccSetPropertyVal{number-width-level-max}{1sp}}%
  \cc@get@indent[\ccl@calc@margin@left]{}{\the\cclTopID}%
}
```

`\ccl@calc@margin@left` is an override for coco-common's `\cc@calc@margin@left` specific for lists. Accordings 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.

```
\def\ccl@calc@margin@left#1#2{%
  \@tempdima=\ccUseProperty{prev-margin-left}\relax%
  \ccSetPropertyX{margin-left}{\the\dimexpr\@tempdima-\ccUseProperty{indent}\relax}%
}
```

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.

```
\def\cclUseLabelHandler{%
  \expandafter\ifx\csname ccl@make@label@\ccl@list@type\endcsname\relax
    \ccPackageError{lists}{type}
      {List type `\ccl@list@type' does not provide a Label Handler.}
      {Make sure that the body of \ccl@list@type's declaration contains a \string\DeclareLabelHandler.}
  \else
```

```
    \csname ccl@make@label@\ccl@list@type\endcsname
  \fi
}
```

`\cclUseAttributeHandler` checks if the list type specific attribute handler exists and applies it to the attribute list #1.

```
\def\cclUseAttributeHandler#1{%
  \ccParseAttributes{\cc@cur@cont-\cclCurDepth}{#1}%
  \expandafter\ifx\csname ccl@eval@attrs@\ccl@list@type\endcsname\relax
    \ccPackageError{Lists}{Type}
      {List type `\ccl@list@type' does not provide an Attribute Handler.}
      {Make sure that the body of \ccl@list@type's declaration contains a \string\
          DeclareAttributeHandler.}
  \else
    \csname ccl@eval@attrs@\ccUseProperty{list-type}\endcsname
  \fi
}
```

5 Default List Types

Vanilla CoCoTEX supports three list types: numbered lists (corresponds to LATEX's `enumerate` environment), unnumbered lists (`itemize`), and description lists (`descripton`).

5.1 Unnumbered Lists

`unnumbered` is technically an abstract child Container of the `List` parent.

```
\ccDeclareListType{unnumbered}{%
```

`\ccl@make@label@unnumbered` generates the `Label` Component of an unnumbered list type.

```
  \DeclareLabelHandler{%
    \ccComponent{Label}{\ccUseProperty{default-label}}}
```

`\ccl@eval@attrs@itemize` is the handler for attributes of itemize-like list types. Currently, it does nothing.

```
  \DeclareAttributeHandler{}}
```

Itemize-Type List Specific Properties

`default-label` `<any>` is a property that holds a fallback label which is used when the optional argument of `\Item` is omitted.

```
  {\ccSetProperty{default-label}{-}}
```

Itemize-Style Default Lists

`Itemize` is the user-level unnumbered `List` Container.

```
\ccDeclareList{Itemize}{unnumbered}{\ccSetProperty{default-label}{\textbullet}}
\ccDeclareNested{Itemize}{1}{%
```

```
  \ccSetProperty{label-face}{\normalfont\bfseries}%
  \ccSetProperty{default-label}{ \textendash}}
\ccDeclareNested{Itemize}{2}{\ccSetProperty{default-label}{\textasteriskcentered}}
\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.

```
\newcount\ccl@item@adv
```

`numbered` is an abstract child Container of the List parent that represents numbered lists.

```
\ccDeclareListType{numbered}{%
```

`\ccl@eval@attrs@numbered` is the handler for attributes specific to the enumerate-like list types.

```
  \DeclareAttributeHandler{%
```

The attribute `step` indicates by what amount the interal counter should be advanced for each item. Defaults to +1 if none is given.

```
    \ccIfAttr{\cc@cur@cont-\cclCurDepth}{step}
      {\ccl@item@adv=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@step\
          endcsname\relax}%
      {\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`.

```
    \ccIfAttr{\cc@cur@cont-\cclCurDepth}{start}
      {\ccl@item@cnt=\expandafter\numexpr\csname cc@\cc@cur@cont-\cclCurDepth @attr@start\
          endcsname\relax
       \advance\ccl@item@cnt-\ccl@item@adv}%
      {\ccl@item@cnt=\z@\relax}%
    }
```

`\ccl@make@label@numbered` is the Label handler of a numbered list type.

```
  \DeclareLabelHandler{%
    \advance\ccl@item@cnt \ccl@item@adv\relax
    \expandafter\ifx\csname ccl@label@type@\ccUseProperty{enum-type}\endcsname\relax
      \ccPackageWarning{lists}{type}{Enum type \ccUseProperty{enum-type} is unknown, revert to
          numeric counters!}
      \let\ccl@label\ccl@label@type@arabic%
    \else
      \letcs\ccl@label{ccl@label@type@\ccUseProperty{enum-type}}%
    \fi
    \ccComponent{Label}{\ccl@label{\ccl@item@cnt}}
  }%
```

```
}{%
```

Numbered List-Specific Properties

New Properties

enum-type [arabic|roman|Roman|Alph|alph] controls how the item counter is rendered when it is not given explicitly with the optional argument of \item. The default values are borrowed from LaTeX's default enumerate types and defined below.

```
  \ccSetProperty{enum-type}{arabic}%
```

Properties with Deviating Default Values

By default, numeric Label are followed by a period to accommodate LaTeX customs.

```
  \ccSetProperty{label}{\ccUseComp{Label}.}}
```

Available Counting Styles

\ccl@label@type@arabic transforms the value of the following (implicit) counter to arabic numerals.

```
\def\ccl@label@type@arabic{\@arabic}
```

\ccl@label@type@roman transforms the value of the following (implicit) counrer to lower case roman numerals.

```
\def\ccl@label@type@roman{\@roman}
```

\ccl@label@type@Roman transforms the value of the following (implicit) counrer to upper case roman numerals.

```
\def\ccl@label@type@Roman{\@Roman}
```

\ccl@label@type@alph transforms the value of the following (implicit) counrer to lower case alphabetic letters.

```
\def\ccl@label@type@alph{\@alph}
```

\ccl@label@type@Alph transforms the value of the following (implicit) counrer to upper case alphabetic letters.

```
\def\ccl@label@type@Alph{\@Alph}
```

Enumerate-Style Default Lists

Enumerate is the user-level Container for numbered List Containers.

```
\ccDeclareList{Enumerate}{numbered}{}
\ccDeclareNested{Enumerate}{1}{% (
  \ccSetProperty{label}{\ccUseComp{Label})}%
  \ccSetProperty{enum-type}{alph}%
}
\ccDeclareNested{Enumerate}{2}{\ccSetProperty{enum-type}{roman}}
\ccDeclareNested{Enumerate}{3}{\ccSetProperty{enum-type}{Alph}}
```

5.3 Description Lists

text is an abstract child Container of the List parent used for description-like list types.

```
\ccDeclareListType{text}{%
```

`\ccl@eval@attrs@text` is the handler for the attributes of description-like list types.

```
\DeclareAttributeHandler{%
  \ccIfAttr{\cc@cur@cont-\cclCurDepth}{width}
    {\ccSetPropertyVal{min-margin-left}{\expandafter\dimexpr\csname cc@\cc@cur@cont-\
         cclCurDepth @attr@width\endcsname\relax}}%
    {\ccSetProperty{min-margin-left}{2em}}%
  \ccIfPropVal{label-growth}{down}
    {\long\def\ccl@vbox##1{\smash{\vtop{##1}}}}
    {\long\def\ccl@vbox##1{\vbox{##1}}}%
}
```

`\ccl@make@label@text` creates the label of a description-like list type.

```
\DeclareLabelHandler{%
  \ccComponent{Label}{}%
}}
```

Description-Type Specific Properties

New Properties

`label-growth` `[up|down]` controls the direction labels ‘‘grow’’ into when they need more space than `max-label-width`. On TEX-primitive level, it controlls whether the label is put into a `\vbox` or `\vtop` with `\hsize=\cclItemIndent`.

Improtant 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.

```
{\ccSetProperty{label-growth}{up}%
```

Properties with Deviating Default Values

The Properties `margin-left` and `indent` of text-type lists are by default set to `auto`.

```
\ccSetProperty{indent}{auto}%
\ccSetProperty{margin-left}{auto}%
```

To accommodate for the new `label-grow` option, the `label-box` has a conditional that switches between regular `\hbox` labels and the two `\vbox` variants described above.

```
\ccSetProperty{label-box}{%
  \ifdim\ccUseProperty{label-width}<\ccUseProperty{max-label-width}\relax
    \hbox to \cclItemIndent{%
      \ccIfPropVal{label-align}{left}{}{\hss}%
      \ccUseProperty{label-format}%
      \ccIfPropVal{label-align}{right}{}{\hss}}%
  \else
    \ccl@vbox{\relax%
      \hsize\dimexpr\cclItemIndent%
      \leftskip\z@
      \rightskip\z@
      \parindent\z@
      \leavevmode
      \ccUseProperty{label-format}%
      \@@par
    }%
  \fi
}}
```

Description-Type Default Lists

`Description` is the user-level Container for text type `List` Containers.

As with the standard LaTeX`description` environment, there are no default definitions for nested Description-type lists.

```
\ccDeclareList{Description}{text}{%
  \ccSetProperty{label-face}{\bfseries}
}
```

5.4 Replacing LaTeX's Default Lists

At the User's descretion (using the `replace` package option, see Sect. 1.1, above), LaTeX's default list environments *`itemize`*, *`enumerate`*, and `description` are re-defined to use CoCoTeX's list mechanism, instead.

```
\if@ccl@replace
  \letcs\itemize{\ccPrefix Itemize}
  \letcs\enditemize{end\ccPrefix Itemize}
  \letcs\enumerate{\ccPrefix Enumerate}
  \letcs\endenumerate{end\ccPrefix Enumerate}
  \letcs\description{\ccPrefix Description}
  \letcs\enddescription{end\ccPrefix Description}
\fi
```

```
</lists>
```

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.

Macro and Environment Index

In this index, the `cc(@)`- and module specific `ccX(@)`-Prefixes were omitted when sorting the entries.

Container Index

Component Index

In this index, the name in parentheses denote the (abstract) Container within which the Component entry is defined.

Property Index

In this index, the name in parentheses denote the (abstract) Container within which the Property entry is defined.

Hook Index

Tag Index

Attribute Index