The cocotex.dtx Package

A modular package suite for automatic, flexible typesetting

Version 0.4.1

(2024/03/23)

Lupino

lupino@le-tex.de

Table of contents

Introduction

1 Basic concepts

The core concept of the CoCoTEX Framework is the strict separation between document specific information bearing units and publisher specific layout and rendering instructions to a degree that is far more versatile and delicate than LATEX's usual distinction between form and content.

The basic data type in the Framework is the **Container**. On the end-user level, this is virtually always a LATEX environment that contain a specific set of macros used to store the atomic units of information. Those macros and their contents are called **Components**.

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

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

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

Modules, that utilize the Property and Component mechanisms, define a *Declare macro*. This Declare macro is basicly a constructor for a new LATEX environment which should share some common *Properties* and *Components* with other environments that are defined with the same Declare macro. Modules, therefore, constitute what in other programming languages may be referred to as *Namespaces*.

The purpose of the Declare macro is

1. to define a LATEX environment to be used in tex documents,
2. to define the Component macros available and allowed within that environment
3. to define the available Properties used to determine the appearance of the environment's content in the final render.
4. to define the processing of the information specific to each instance of the environment.

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

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

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

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

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

One more driver function

```
%<*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>
```

Modul 1

cocotex.dtx

This is the main class file for the CoCoTEX LATEX package.

```
%<*class>
```

File Preamble

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

1 Hard-coded requirements

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

2 Class Options

Passing options down to the LATEX standard packages

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

The option pubtype (short for “publication type”) has possible four values: mono, collection, journal, and article . mono (also the default when no pubtype is given) and collection are used to switch between single and multiple contributor documents; collection and journal to switch between one-time text collections and periodicals, respectively. All three types implicitly load the 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
\define@choicekey{cocotex.cls}{pubtype}[\cc@pub@type\nr]{collection,article,journal,mono}{%
  \ifcase\nr\relax% collection
    \global\collectiontrue
  \or% article
    \global\articletrue
  \or% journal
    \global\journaltrue
  \else% monograph
    \global\monographtrue
  \fi
}
\DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{article}}
\DeclareOptionX*{\PassOptionsToClass{\CurrentOption}{book}}
```

Passing options down to various CoCoTEX modules:

```
\DeclareOptionX{debug}{\PassOptionsToPackage{\CurrentOption}{coco-kernel}}
\DeclareOptionX{a11y}{\PassOptionsToPackage{init}{coco-accessibility}}
\DeclareOptionX{lang-id}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
\DeclareOptionX{nodetree}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
\DeclareOptionX{showspaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
\DeclareOptionX{no-spaces}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
\DeclareOptionX{no-paras}{\PassOptionsToPackage{\CurrentOption}{coco-accessibility}}
\DeclareOptionX{no-compress}{\let\cc@no@pdf@compression\relax}
\DeclareOptionX{color-enc}{\PassOptionsToPackage{\CurrentOption}{coco-common}}
\DeclareOptionX{usescript}{\PassOptionsToPackage{\CurrentOption}{coco-script}}
\DeclareOptionX{nofigs}{\PassOptionsToPackage{\CurrentOption}{coco-floats}}
\DeclareOptionX{ennotoc}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
\DeclareOptionX{endnotes}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
\DeclareOptionX{resetnotesperchapter}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
\DeclareOptionX{endnotesperchapter}{\PassOptionsToPackage{\CurrentOption}{coco-notes}}
\ProcessOptionsX
```

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). The hook is expanded at the very end of the `cocotex.cls` file. The `coco-xerif` module itself is loaded early in `coco-common.sty`.

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

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

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

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

```
\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 `book` or `article`:

```
\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
```

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

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

```
\ifarticle\else
  \newif\if@frontmatter \@frontmatterfalse
  \renewcommand\frontmatter{%
    \cleardoublepage
    \@mainmatterfalse
    \@frontmattertrue
    \pagenumbering{arabic}}
  \renewcommand\mainmatter{%
    \cleardoublepage
    \@frontmatterfalse
    \@mainmattertrue}
  \renewcommand\backmatter{%
    \cleardoublepage
    \@mainmatterfalse
    \@frontmatterfalse}
\fi
```

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

```
\usepackage{soul}
```

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

```
\RequirePackage{hyperref}
```

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

```
\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 necessary

```
  ,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=auto%
```

Bookmarks are numbered by default.

```
  ,bookmarksnumbered=true%
}
```

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

```
\ifx\cc@no@pdf@compression\relax
  \ifx\pdfobjcompresslevel\@undefined
    \edef\pdfobjcompresslevel{\pdfvariable objcompresslevel}%
  \fi
  \pdfcompresslevel=0
  \pdfobjcompresslevel=0
\fi
```

8 End of Dcument Class Hook

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

```
\ccAfterClassHook
```

```
%</class>
```

Part I

Core Functions

Modul 2

coco-kernel.dtx

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

```
%<*kernel>
```

1 Preamble

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

1.1 Hard dependencies

```
\RequirePackage{kvoptions-patch}
\RequirePackage{xkeyval}
\RequirePackage{etoolbox}
```

1.2 Package Options

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

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

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

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

2 Exception handlers

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

`\ccPackageError` creates an error message specific to the Framework. #1 is the module, #2 is the type of error, #3 is the immediate error message, #4 is the help string.

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

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

`\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.

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

```
\def\ccKernelDebugMsg#1{\if@cc@debug\message{[CoCo Kernel Debug]\space\space#1^^J}\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
```

`\ccIfAlly` the same, 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
```

`\if@cc@modern` is a switch to distinguish older versions of the LaTeX Kernel from newer versions. The pivot date is defined to be 2020/06/01.

```
\ifx\IfFormatAtLeastTF\@undefined
  \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}%
\fi
\newif\if@cc@modern
\IfFormatAtLeastTF{2020/06/01}{\@cc@moderntrue}{\@cc@modernfalse}
```

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 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}%
\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.

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

`\ccDeclareType` Each Container is defined by the data types it provides. These data types are declared with this macro. The first argument ##1 is the name of the data type. The second argument ##2 is a list of 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}}%
```

\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*. With the optional argument ##1 you can override the environment's name. However, keep in mind that the Container's name is not changed by re-naming the corresponding environment. ##2 is used for the stuff done at the beginning of the environment, ##3 for the stuff done at the end.

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

```
    \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}%
```

```
    \def\x{%
      #2%
    }%
  \expandafter\x\endgroup
}
 %\@onlypreamble\ccDeclareContainer
```

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

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

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

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

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

```
\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
    \ccKernelDebugMsg{Evaluating cc@type@#2@#1:^^J \csmeaning{cc@type@#2@#1}}%
    \csname cc@type@#2@#1\endcsname
  \fi
}
```

\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
}
```

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

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

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

```
\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}
```

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

```
\def\cc@do@inherit#1#2#3{%
  \ccKernelDebugMsg{#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}%
    \ccKernelDebugMsg{value cc@type@#1@#3:^^J \expandafter\meaning\csname cc@type@#1@#3\
        endcsname}%
  \fi
}
```

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 simple component macros.

#1 is the Component's identifier. The internal macro that is used to store the Component's value is `\csname cc@<current Container name>@<#1>\endcsname`. If omitted, #1 is the same as #2.

#2 is the Component's name.

#3 is code that is executed *before* assignment of the user's value
#4 is code that is executed *after* assignment of the user's value

```
\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}\ignorespaces
        #4}%
}
```

\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%
}
```

Once declared, a component can be set in two ways: The first way is to use \ccPrefix<name> with one argument for its value. The second, preferred, way is to use the \ccComponent macro which takes two arguments: #1 is the name of the Component, #2 is the value. This macro checks whether an Component of name #1 has actually been declared and does so, if not.

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

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

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

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

\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}%
}
```

\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 high 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}}
```

\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.

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

\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, \ccGetComp* is a high level command to return the contents stored in a Component of name #1 as a paragraph iff the Component is neither empty nor \relax. If Accessibility features are activated, the returned content of the Component is autmatically tagged with a Para tag. The starred version of \ccGetComp supresses auto-tagging for that Component.

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

\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\cc@long@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 (=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.

Optional #1 is the prefix of the fallback component, #2 is the Container name, #3 is the name of the Component, #4 is the Override's prefix.

```
\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 #1 and the body #2.

```
\def\ccDeclareComponentGroup#1#2{%
  \csnumgdef{cc#1Cnt}{\z@}%
  \csdef{\ccPrefix#1}{\cc@opt@empty{\csname cc@group@#1\endcsname}}%
  \csdef{cc@group@#1}[##1]{%
    \def\cc@cnt@grp{cc#1}%
    \csxdef{cc#1Cnt}{\expandafter\the\expandafter\numexpr\csname cc#1Cnt\endcsname+\@ne\relax}%
    \if!##1!\else\csgdef{cc@\cc@cur@cont @#1-\csname cc#1Cnt\endcsname @attrs}{##1}\fi
    #2%
    \csname @#1@hook\endcsname
  }%
  \csdef{end\ccPrefix#1}{{\ccToggleCountedConditionals\csuse{cc@compose@group@#1}}}%
}
```

\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.}}%
}
```

\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 specific Property of a group.

```
\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\relax
    \cslet{#1}\relax
```

```
  \else
    \expandafter\csname #1\expandafter\endcsname\expandafter{\cc@iterate@res}%
  \fi
  \global\let\cc@iterate@res\relax
}
```

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

```
\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}{%
    \ifcsdef{cc#1Cnt}{%
      \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@cnt@grp{cc#1}}%
         \expandafter\@tempcntb=\csname cc#1Cnt\endcsname\relax
         \cc@iterate{\@tempcnta}{\@ne}{\@tempcntb}{%
           \edef\@tempb{%
             %% top-level counter for user interaction
             \noexpand\ccCurCount=\the\@tempcnta
             %% evaluating group attributes
             \ifcsdef{cc@\cc@cur@cont @#1-\the\@tempcnta @attrs}{\noexpand\ccParseAttributes{#1-\
                 the\@tempcnta}{\csname cc@\cc@cur@cont @#1-\the\@tempcnta @attrs\endcsname}}{}
             %% internal counter for macro grabbing
             \noexpand\csnumdef{cc#1Cnt}{\ccCurCount}%
             \noexpand\ccUseProperty{#2}}%
           \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}%
         \cc@assign@res{\ccPrefix#3}%
      \fi
    }{}}%
}
```

\ccApplyCollection is an alternative version of \ccComposeCollection and fully expands the 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}%
   }%
}
```

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

```
\def\cc@apply@collection#1#2{%
  \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}{\ccParseAttributes{#1-\the\@tempcnta
            }{\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
}
```

`\cc@comp@def` is used to pass a Counted Component into a TeX macro. #1 is a prefix to the def command, e.g., `\global` or `\protected`; #2 is a CS token, #3 is the Name of the Counted Component, and #4 is the Property that should be applied to all Members of the Counted Component.

```
\def\cc@comp@def{\cc@opt@empty\@cc@comp@def}
\def\@cc@comp@def[#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
}
```

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

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

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

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

Declaring Counted Component

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

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

\cc@def@counted@comp registers counter dependent Components. #1 is the internal name of the Component which is composed out of the group name, the value of the group counter and the user-level macro name #2; #3 is some custom code passed to the second argument of \ccDeclareComponent; and #4 is a modifier to the internal macro definition.

```
\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}%
   #4\expandafter\long\expandafter\def\csname cc@\cc@cur@cont @#2\endcsname{\csname cc@\
       cc@cur@cont @#1\endcsname}%
}
```

Resetting Counted Component

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

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 type of the Component set.

```
\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 Hooks

TODO: Use latex3's hook facility instead.

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

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

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

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

```
\def\ccAddToHook{\cc@opt@curcont\cc@add@to@hook}
\def\cc@add@to@hook[#1]#2#3{%
  \expandafter\ifx\csname cc@hook@#1@#2\endcsname\relax
    \ccDeclareHook[#1]{#2}%
  \fi
  \csgappto{cc@hook@#1@#2}{#3}%
}
```

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

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

7 Properties

7.1 Setting Properties

`\ccSetProperty` is a user-level macro that provides the Property–Value interface for Containers. #1 is the name of the Property, #2 is the Value assigned to that Property.

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

\ccAppToProp can be used add material to the *end* of an existing Property vaue. #1 is the name of the Property, #2 is the material to be added to previous value of that Property.

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

\ccPreToProp can be used add material to the *beginning* of an existing Property. #1 is the name of the Property, #2 is the material to be inserted before to previous value of that Property.

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

\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}}}
```

\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}}
```

\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}}
```

\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}}
```

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

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

7.2 Using Properties

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

```
\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}%
}
```

`\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]}
\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 variant of etoolbox's `\ifstrequal` that first fully expands both comparatives before evaluating 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, and #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.

`\ccPrefix SetPropLocal, \ccPrefix AppPropLocal, \ccPrefix PrePropLocal` They all take two arguments: #1 is the name of the Property, #2 is the value to be set, appended, or prepended to that Property, respectively.

```
\def\ccSetPropLocal{\cc@set@property@locally\ccSetProperty}
\cslet{\ccPrefix SetPropLocal}\ccSetPropLocal%
\def\ccAppPropLocal{\cc@set@property@locally\ccAppToProp}
\cslet{\ccPrefix AppPropLocal}\ccAppPropLocal%
\def\ccPrePropLocal{\cc@set@property@locally\ccPreToProp}
\cslet{\ccPrefix PrePropLocal}\ccPrePropLocal%
```

7.3 Processing Instructions

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

7.4 Property Conditionals

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

```
\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}
```

\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}
```

8 Helper macros

8.1 Handling of Optional Arguments

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

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

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

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

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

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

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

8.2 Iterators

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

```
\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}%
```

8.3 Attributes

Many macros and environments deal with optional arguments that are used to alter the behaviour of that macro or environment. The combination of a parameter and its set of possible values are 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 parent node of the attribute, #2 is the attribute chain

```
\gdef\ccParseAttributes#1#2{%
  \if!#1!\else
    \if!#2!\else
      \def\cc@cur@node{#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 parameter 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@node @attr@#1\endcsname\@empty%
  \else
    \ifx #2 =\else
      \expandafter\def\csname cc@\cc@cur@node @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
}
\endgroup
```

\ccGetAttribute returns the value of an attribute.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8.4 Style Classes

Style Classes are locally usable sub-Containers.

\ccDeclareClass The top-level macro \ccDeclareClass[#1]{#2}[#3]{#4} has four arguments, two of which are optional. #2 is the name of the class. If this argument is empty, the special class name default is used. #4 is the declaration block of the class. This argument usually containsa set of property assignments using the \ccSetProperty{<prop>}{<val>} macro, see Sect. 7. The first optional argument #1 is the Style Class' parent Container. Using parent Containers, you can have Style Classes of the same name for different (sub-)Containers, e.g., a default class for each float and heading Container. The second optional argument #3 is the parent Style Class. Properties from that Style Class are loaded automatically prior to the loading of the current Style Class's Properties. This applies recursively allowing for a cascading of property values, as in CSS.

```
\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}%
}
```

\ccUseStyleClass is a user-level macro to expand and âĂIJactivateâĂİ 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.

```
\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'!}%
    \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}
```

\CoCoTeX the CoCoTEX Logo.

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

```
%</kernel>
```

Modul 3

coco-common.dtx

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

```
%<*common>
```

```
%%
%% module for CoCoTeX that provides some commonly used base macros.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-common}
    [2024/03/23 0.4.1 CoCoTeX common module]
```

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

```
\RequirePackage{kvoptions-patch}
\RequirePackage{xkeyval}
\RequirePackage{iftex}
```

1 Package options

1.1 Accessibility Features

Default color encoding passed as option to the xcolor package.

```
\def\cc@color@enc{cmyk}
\define@choicekey{coco-common.sty}{color-enc}[\@cc@color@enc\nr]{srgb,rgb,gray,cmy,cmyk,natural
    }[cmyk]{%
  \let\cc@color@enc\@cc@color@enc
  \ifcase\nr\relax% srgb
    \def\cc@color@enc{rgb}%
  \or% rgb
  \or% gray
  \or% cmy
    \def\cc@color@enc{cmyk}%
  \or% cmyk
  \else% natural, i.e. no conversion of color spaces takes place
  \fi
}
\ProcessOptionsX
\PassOptionsToPackage{\cc@color@enc}{xcolor}%
```

\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}
\DeclareGraphicsRule{.EPS}{eps}{.EPS}{}
```

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}
```

Box Registers

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

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

`\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

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

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

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

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

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

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

2.4 Masks

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

`\hack` intended to mask line breaking macros.

```
\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
}%
```

2.5 Arithmetics

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

```
\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`

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

`\cc@abspage`

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

`\thecc@abspage`

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

`\if@cc@odd`

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

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

We split this into two parts:

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

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

```
\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

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

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

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

```
\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.

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

\cc@extract@generic

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

\cc@print@generic

```
\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 namespace, #3 is the Component prefix and #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

`\ccDeclareExtraToc` provides an interface for additional content lists. #1 is the name of the custom content, #2 is a comma separated list of container names, the instances of which should be listed in the custom contents list.

```
\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}%
      {\protect\ccContentsline{#2}{#3}{#4}{#5}\protected@file@percent}\relax
  }%
  \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
      \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}{}%
  \expandafter\label\expandafter{#1}%
}
\def\cc@fallback@anchor{\phantomsection}%
```

3.5 Linguistic Name generation and selection

\ccSetBabelLabel defined a language-dependent string macro for German and English varieties. #1 is the language, #2 is the internal reference name, and #3 is the language specific label.

```
\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 **??**y using to create a hyperref anchor for label #1.

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

```
%</common>
```

Modul 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
%%
%% lualatex - texlive > 2018
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-accessibility}
    [2024/03/23 0.4.1 CoCoTeX accessibility module]
\RequirePackage{kvoptions-patch}
\RequirePackage{xkeyval}
\RequirePackage{atbegshi}
\RequirePackage{xparse}
```

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

We start with adopting ltpdfa's package options.

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

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

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

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

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

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

```
\DeclareOptionX{show-spaces}{\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
\DeclareOptionX{no-spaces}{\let\cca@do@dospaces\@undefined}
```

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

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

Processing the options.

```
\ProcessOptionsX
```

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

1.2 Activating and Deactivating Accessibility Features

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

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

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

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

1.3 Accessibility-specific additions

Loading Further Dependencies

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

```
\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}
  \directlua{ltpdfa.config.debug = \if@cc@debug true\else false\fi}
  \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}
```

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

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

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

```
  \directlua{ltpdfa.config.driver = "\luaescapestring{pdftex}"}
  \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 disable accessibility macros during runtime, we need each tagging markup macro to exist in two states, one where they trigger tagging into the pdf, and one where they do nothing.

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

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

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

and

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

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

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

`\ccaDisable` disables all ltpdfa commands

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

and

`\ccaEnable` enables all ltpdfa commands.

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

`\CsToStr` is a xparse 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}%
  \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}%
```

Finally, we reset the temporary argument signature macro.

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

Some macros from `ltpdfa.sty`:

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

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

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

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

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

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

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

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

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

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

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

\ccaAddID adds the ID #1 to the current node. If #1 is "auto" the ID 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}')}}
```

\ccaAddScope

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

1.5 Lua injection

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

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

1.6 Hyperlink handling

To tag hyperlinks, we define some ltpdfa interface macros.

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

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

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

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

\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}
     {\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}
     {}{\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}
    {\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%
    }{}{\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
```

1.7 Tagging Page Styles as Artifacts

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

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

```
\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.

```
\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}}
```

1.8 generic artifacts

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

1.9 Tagging for Floats

Taggin for Figures

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

```
\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{%
  \if@cc@modern
    \let\ltx@Ginclulde@graphics\Ginclude@graphics
    \def\Ginclude@graphics#1{\if@cc@is@final\ccaFigureStart{}\fi\ltx@Ginclulde@graphics{#1}\
        if@cc@is@final\ccaFigureEnd{}\fi}%
  \else
    \@ifpackageloaded{grffile}
      {\pretocmd\grffile@Ginclude@graphics{\if@cc@is@final\ccaFigureStart{}\fi}{}{}%
       \apptocmd\grffile@Ginclude@graphics{\if@cc@is@final\ccaFigureEnd{}\fi}{}{}}
      {\pretocmd\Ginclude@graphics{\if@cc@is@final\ccaFigureStart{}\fi}{}{}%
       \apptocmd\Ginclude@graphics{\if@cc@is@final\ccaFigureEnd{}\fi}{}{}}%
  \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}}{}}
```

1.10 Transformation of Typographic Unicode characters

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

```
\ifx\pdfextension\@undefined\else
```

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

1.11 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{%
  \ccDeclareHook[document]{cca/at/begin/document}
  \AtBeginDocument{%
    \directlua{ltpdfa.beginDocument('\luaescapestring{\ltpdfa@last@page}')}
    \ccUseHook[document]{cca/at/begin/document}%
    \directlua{ltpdfa.configAutoclose()}
    \ccaVstructStart{document}%
  }
  \AtEndDocument{%
    \ccaVstructEnd{document}
    \directlua{ltpdfa.endDocument()}%
  }
}
```

2 Default Role Mapping

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

```
\ccaAddRolemap{document}{Document}
\ccaAddRolemap{Para}{P}
```

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

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

End of TEX source code.

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

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

3 Lua code

3.1 Local Variables and Tables

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

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

3.2 Meta Data Extraction

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

```
277 local meta = {
278   Author = '',
279   Title = '',
280   Creator = '',
281   Producer = '',
282   Keywords = '',
```

The method `meta.extract()` reads the meta data from the `\jobname.xmp` and stores certain values to be accessed by LaTeX. This is used to fill the DocumentInfo when a xmp file is available during the expansion of `\cct@write@pdf@meta` from the coco-title module (see Sect. 2).

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

3.3 Public Methods

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

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

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

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

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

```
309 return cocotex.ally
```

no more lua code.

```
310 %</a11y-lua>
```

Modul 5

coco-meta.dtx

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

```
%<*meta>
```

File preamble

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

`Container 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 and one for its Content.

```
\def\ccm@generic@comp{%
  \ccDeclareComponent{GenericMetaBlock}{\expandafter\global}{}%
  \ccDeclareComponentGroup{GenericMeta}{%
    \ccDeclareCountedComponent{Heading}%
    \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 sub-containers that represent the meta-data of people that share a role in contributing content to a document. Examples for such roles are an article/chapter/book's authors, or a collection/series' editors.

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

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

```
\def\ccDeclareRole{\cc@opt@second\cc@declare@role}%
\def\cc@declare@role[#1]#2{%
  \ccDeclareComponentGroup{#2}{%
    \ccDeclareCountedComponent{FullName}%
    \ccDeclareCountedComponent{CiteName}%
    \ccDeclareCountedComponent{ShortCiteName}%
    \ccDeclareCountedComponent{PDFInfoName}%
    \ccDeclareCountedComponent{Initial}%
    \ccDeclareCountedComponent{LastName}%
    \ccDeclareCountedComponent{FirstName}%
    \ccDeclareCountedComponent{MidName}%
    \ccDeclareCountedComponent{Honorific}%
    \ccDeclareCountedComponent{Lineage}%
    \ccDeclareCountedComponent{ORCID}%
    \ccDeclareCountedComponent{AffilRef}% for references to the Affil Group
    \ccDeclareCountedComponent{Affiliation}% for affiliations as direct Author meta data
    \ccDeclareCountedComponent{Email}%
    \ccDeclareCountedComponent{CorrespondenceAs}%
  }%
  \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.}}}%
  }%
  \ccDeclareRoleBlock{#2}{NameList}{#1-list-print-format}%
  \ccDeclareRoleBlock{#2}{CitationList}{#1-list-cite-format}%
```

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

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

```
\def\ccAddToRole#1#2{%
  \csgappto{@#1@hook}{#2}%
}
```

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

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

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

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

```
  \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@role@eval\ccApplyCollection{#1}{#2}}
```

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

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

2 Labeled Components

\ccDeclareLabeledComp declares two Components: one named \ccPrefix #2 for the value and another one named \ccPrefix #2Label for its corresponding label. #3 is used for property overrides. The optional Argument #1 allows to set a default value for the Label.

```
\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
}
```

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

```
\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

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

\ccCurComp stores the currently active Component name

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

```
    \ifx\ccm@no@tag\relax\else\ccaStructStart{MetaDatum}\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}{Para}
\ccaAddRolemap{Affiliations}{Para}
\ccaAddRolemap{MetaDatum}{Div}
\ccaAddRolemap{MetaDatumLabel}{Para}
\ccaAddRolemap{MetaDatumValue}{Para}
\ccaAddRolemap{Abstract}{Div}
\ccaAddRolemap{AbstractLabel}{P}
\ccaAddRolemap{AbstractText}{Div}
\ccaAddRolemap{Keywords}{Div}
\ccaAddRolemap{KeywordsLabel}{P}
\ccaAddRolemap{KeywordsText}{Div}
```

4 Common Meta Data

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

```
\def\ccm@declare@comp{%
    \ccDeclareComponent{Copyright}{\expandafter\global}{}% Copyright text
    \ccDeclareComponent{DOI}{\expandafter\global}{}% DOI
}%
```

Container article-meta

```
  %% for single articles
\ccDeclareContainer{article-meta}{%
  \ccDeclareType{Components}{%
    \ccDeclareGlobalComponent{StartPage} % Start page of a single article
    \ccDeclareGlobalComponent{EndPage} % End page of a single article
    \ccDeclareLabeledComp[Cite as]{CiteAs}{cite-as} % As what the article should be cited
    \ccDeclareLabeledComp[Submitted]{Submitted}{sumbitted} % Date the article was submitted
    \ccDeclareLabeledComp[Received]{Received}{received} % Date the article was recieved
    \ccDeclareLabeledComp[Revised]{Revised}{revised} % Date the article was revised
    \ccDeclareLabeledComp[Reviewed]{Reviewed}{reviewed} % Date the article was reviewed
    \ccDeclareLabeledComp[Accepted]{Accepted}{accepted} % Date the article was accepted
    \ccDeclareLabeledComp[Published]{Published}{published} % Date the article was published
    \ccDeclareLabeledComp[Conflict of Interest]{COIStatement}{coi-statement}% Conflict of Interest
        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{%
  \ccDeclareLabeledComp[Abstract]{Abstract}{abstract}%
  \ccDeclareLabeledComp[Keywords]{Keywords}{keyword}%
  \ccDeclareLabeledComp{DOI}{doi}%
  \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{%
  \ccDeclareComponent{AffilBlock}{\expandafter\global}{}%
  \ccDeclareComponentGroup{Affil}{%
    \ccDeclareCountedComponent{Affiliation}%
    \ccDeclareCountedComponent{Address}%
    \ccDeclareCountedComponent{Institute}%
    \ccDeclareCountedComponent{Country}%
    \ccDeclareCountedComponent{Department}%
    \ccDeclareCountedComponent{AffilID}%
  }%
  \ccDeclareGroupHandler{Affil}{%
    \ccUnlessComp{AffilID}{\ccComponentEA{AffilID}{\ccAffilCnt}}%
    \ccUnlessComp{Affiliation}{\ccComponent{Affiliation}{\ccUseProperty{affiliation-format}}}%
  }%
}
```

Defaut Property settings for the Meta Container.

```
\ccAddToProperties{CommonMeta}{%
  \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
  }
  \ccSetProperty{initials-sep}{~}
  \ccSetProperty{initials-period}{.}
  %
  %% Properties that control how the composed compoents WITHIN each item in a Role are formatted:
  %
  \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%
  }% How FullName for each name is built
```

```
\ccSetProperty{role-cite-name-format}{\ccIfComp{LastName}{\ccUseComp{LastName},~\ccUseComp{Initial}}{\ccUseComp{FullName}}}% How CiteName for each name is built
\ccSetProperty{role-short-cite-name-format}{\ccUseComp{LastName}}% how ShortCiteName for each name is built
\ccPropertyLet{role-pdfinfo-name-format}{role-cite-name-format}% How PDFInfoName for each item is built
\ccSetProperty{role-correspondence-as-format}{\ccUseComp{Email}}% How PDFInfoName for each item is built
%% Properties that control how the single items in a compoent list are formatted:
\ccSetProperty{role-block-print-format}{\ccUseComp{FullName}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How <Role>NameList for each name is build
\ccSetProperty{role-block-cite-format}{\ccUseComp{CiteName}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How each item in Component <Role>CitationList is formatted
\ccSetProperty{role-block-short-cite-format}{\ccUseComp{ShortCiteName}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How each item in the Component <Role>ShortCitationList is formatted
\ccSetProperty{role-block-pdfinfo-format}{\ccUseComp{PDFInfoName}\ifnum\ccCurCount<\ccTotalCount\ccUseProperty{counted-name-sep}\fi}% How each item in the Component <Role>PDFInfo is formatted
\ccSetProperty{role-block-correspondence-format}{%
  \ccIfAttrIsSet{\cc@cnt@grp\the\ccCurCount}{corresp}
    {\ifx\is@first@corresp\relax
       \ccUseProperty{corresp-sep}%
     \else
       \global\let\is@first@corresp\relax
     \fi
     \ccUseComp{CorrespondenceAs}%
   }{}}% How each item in the Component <Role>Correspondence is formatted
% Aliasses
% for Role "Author":
\ccPropertyLet{author-cite-name-format} {role-cite-name-format}%
\ccPropertyLet{author-short-cite-name-format} {role-short-cite-name-format}%
\ccPropertyLet{author-full-name-format} {role-full-name-format}%
\ccPropertyLet{author-pdfinfo-name-format} {role-pdfinfo-name-format}%
\ccPropertyLet{author-correspondence-as-format} {role-correspondence-as-format}%
%
\ccPropertyLet{author-list-print-format} {role-block-print-format}%
\ccPropertyLet{author-list-cite-format} {role-block-cite-format}%
\ccPropertyLet{author-list-short-cite-format} {role-block-short-cite-format}%
\ccPropertyLet{author-list-pdfinfo-format} {role-block-pdfinfo-format}%
\ccPropertyLet{author-list-correspondence-format} {role-block-correspondence-format}%
%
\ccSetProperty{counted-name-sep}{,\space}%
\ccSetProperty{name-and}{\space and\space}%
\ccSetProperty{name-etal}{\space et~al.}%
\ccSetProperty{name-sep}{,\space}%
\ccSetProperty{corresp-mark}{*}%
\ccSetProperty{corresp-sep}{,\space}%
%
% Affiliation Properties
%
\ccSetProperty{affiliation-format}{% Format of the affiliation block
  \ccWhenComp{Institute}{\ccUseComp{Institute}}%
  \ccWhenComp{Department}{, \ccUseComp{Department}}%
  \ccWhenComp{Address}{, \ccUseComp{Address}}%
}%
\ccSetProperty{affil-sep}{\par}
\ccSetProperty{affil-block-item-face}{}% Font of a single item in the affiliation list
\ccSetProperty{affil-block-item-format}{% Format of a single item in the affiliation list
  \textsuperscript{\ccUseComp{AffilID}}%
  \bgroup
```

```
      \ccUseProperty{affil-block-item-face}%
      \ccUseComp{Affiliation}
    \egroup%
    \ifnum\ccCurCount<\ccTotalCount\relax\ccUseProperty{affil-sep}\fi%
  }
  \ccSetProperty{affil-block-face}{\small\normalfont}%
  \ccSetProperty{affil-block-format}{%
    \ccWhenComp{AffilBlock}
      {\bgroup
         \ccUseProperty{affil-block-face}%
         \ccUseComp{AffilBlock}%
       \egroup
       \par
     }}
  %
  % Labeled Meta Properties
  %
  \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}}%
  }
  \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
    }}
  \ccSetProperty{labeled-meta-label-face}{\bfseries}
  \ccSetProperty{labeled-meta-label-sep}{:\enskip}
  \ccSetProperty{labeled-meta-face}{}
  \ccSetProperty{labeled-meta-before}{}
  \ccSetProperty{labeled-meta-after}{\par}
}
```

```
%</meta>
```

Part II

Document Level Structures

Modul 6

coco-headings.dtx

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

```
%<*headings>
```

```
%%
%% module for CoCoTeX that extends heading objects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive >= 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-headings}
    [2024/03/23 0.4.1 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

`Container Heading`

```
\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.

```
    \cch@provide@comp{Title}%
    \cch@provide@comp{Subtitle}%
    \cch@provide@comp{Number}%
    \cch@provide@comp{LicenceLogo}%
    \cch@provide@comp{LicenceName}%
    \ccDeclareComponent{RefLabel}{}{}%
    \cch@provide@quotes
  }%
  \ccDeclareType{Properties}{}%
  \ccDeclareEnv{\cch@heading}{\cch@end@heading}%
}
```

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

- `#1` (optional) inherit-from: load all properties from that heading level, first.
- `#2` level: used for toc entries. -1 for part, 0 for chapter, 1 for section, etc.
- `#3` name: part, chapter, section, etc, to be used in toc, head lines, bookmarks, etc.
- `#4` Property definitions and switches

```
\long\def\ccDeclareHeading{\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}%
```

\cch@2@unique is a unique name for the heading's level. Is is always the name of the *first* heading that is defined with a given heading level counter.

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

```
      \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}
  }% \ifcsdef cc@container@#3 fi
```

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

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

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

```
  \ifnum#2<\cch@min@level\relax
    \global\cch@min@level=\csname cch@#3@level\endcsname\relax
  \fi
  \ifnum#2>\cch@max@level\relax
    \global\cch@max@level=\csname cch@#3@level\endcsname\relax
  \fi
}% \cc@declare@heading
```

Each new heading level needs some configuration with the `ltpdfa` package in order to automatically close heading tags with the beginning of a new heading.

`\cch@add@autoclose` adds the new heading level to `ltpdfa`'s autoclose mechanism. #1 is the numeric level, #2 is the name of the heading. We do this inside the `cca/before/begin/document` hook, since we need to know *all* locally defined heading levels beforehand in order to build the Sectioning tree correctly.

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

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

```
    \ccaAddRolemap{#2}{Sect}%
```

Then we determine the hierarchical heading level we need to assign to the PDF tags. H1 is always reserved for the entire document's title, so we need to calculate the difference of the lowest used value and 2 and add this to the actual level of the current heading.

```
    \cch@tempcnta=\numexpr\tw@-\cch@highest@level\relax
    \advance\cch@tempcnta by #1\relax
    \ccaAddRolemap{#2head}{H\the\cch@tempcnta}%
    \ifnum\cch@tempcnta>6\relax
      \ccaAddRolemap{H\the\cch@tempcnta}{H}%
    \fi
```

Next, we tell `ltpdfa` for each heading level which other heading level is the next down the Sectioning hierarchy. For that, we first put the current heading level in a calculable counter.

```
    \cch@tempcnta=#1\relax
```

Then we catch the heading with the highest level (from the aux file) and set the *document* layer in the `ltpdfa`'s `Sectioning` table to have that heading as its child

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

Then, we catch the lowest level to tell ltpdfa's Sectioning table that this level has no children. Another switch is made to distinguish first-born heading levels from aliases, since the Sectioning table can only hold one heading per level. All other hadings of the same level are, per definition, Aliases of the one that has been defined first.

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

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

```
      \@tempswatrue
      \loop
```

by incrementing the heading level counter by one

```
        \advance\cch@tempcnta\@ne\relax
```

and checking the variable repeat condition:

```
      \if@tempswa
```

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

```
        \ifnum\cch@tempcnta>\cch@lowest@level\relax
          \@tempswafalse
        \else
```

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

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

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

```
            \@tempswatrue
          \else
```

If the heading level is defined, we configure autoclose such that the level with the iterator counter is set to be the child of the current heading level in ltpdfa's Sectioning table. As above, we distinguish between original headings and Aliases.

```
            \ifcsstring{cch@#1@unique}{#2}
              {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:\csname cch@
                  \the\cch@tempcnta @unique\endcsname}{Egroup:false}}}\x}
              {\edef\x{\noexpand\ccaAddToConfig{autoclose}{#2={Type:Sectioning}{Child:\csname cch@
                  \the\cch@tempcnta @unique\endcsname}{Egroup:false}{Alias:\csname cch@#1@unique\
                  endcsname}}}\x}%
            \@tempswafalse
          \fi
        \fi
```

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

```
      \repeat
    \fi
  }}
```

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

```
\newcount\cch@min@level \cch@min@level=99\relax
\newcount\cch@max@level \cch@max@level=-99\relax
\ifx\cch@highest@level\@undefined \def\cch@highest@level{99}\fi
\ifx\cch@lowest@level\@undefined \def\cch@lowest@level{-99}\fi
\AtEndDocument{%
  \immediate\write\@mainaux{\string\gdef\string\cch@highest@level{\the\cch@min@level}}%
  \immediate\write\@mainaux{\string\gdef\string\cch@lowest@level{\the\cch@max@level}}%
}%
```

`\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-hook}{#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{before-print-hook}{#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
        \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}}{}%
      \egroup%
      \cch@make@run% Running headers
      \ccUseProperty{after-heading-block}%
    }%
```

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
    \edef\@cch@parent{#1-\csname cc@parent@#2\endcsname}%
    \expandafter\ccUseHook\expandafter{\@cch@parent}%
  \fi
  \ccUseHook{#1-\ccCurSecName}%
}
```

\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{%
  \ccDeclareHook{toc-before-hook-#1}% Expanded before the toc entry is printed
  \ccDeclareHook{toc-after-hook-#1}% Expanded after the toc entry is printed
  \ccDeclareHook{before-hook-#1}% Expanded before before-heading property is expanded
  \ccDeclareHook{before-print-hook-#1}% Expanded at the very beginning of the local definition of \
      @svsec
}
```

\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}%
       {\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}%
   }}
```

`\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
    \ccUseHook{toc-before-hook-#1}%
    \ccUseProperty{toc-before-entry}%
    \ccUseProperty{toc-format}%
```

```
    \ccUseHook{toc-after-hook-#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}%
  \ccIfProp{interline-para}
    {\global\setbox\cch@inline@sec@box\hbox{\ifvoid\cch@inline@sec@box\else\unhbox\
        cch@inline@sec@box\ccUseProperty{interline-para-sep}\fi\@svsec}}%
    {\global\setbox\cch@inline@sec@box\hbox{\@svsec}}
  \@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{%
```

Adding start tags for the contents that "belong" to a heading. **Warning**, the following code is incredibly ugly. In principle, we close the semi-group opened by `begin`, add the tagging, and then re-build the rest of the code from older and more modern LaTeX's standard definitions of `begin`.

This is necessary, because otherwise we would need to either manually add the starting sectioning tag outside the `\ccPrefix Heading` environment, or, if we want to keep `ltpdfa`'s `autoclose` mechanism, the sectioning tag is auto-ended at `\end{Heading}`. Using the `env/Heading/before` hook won't work either, because at the time of its expansion, the level of the heading isn't known, yet. So, we need to take the ugly road, for now.

```
  \ccIfAlly
    {\global\let\cch@currenvir\@currenvir
     \endgroup
     \ccaVstructStart{#2}%
     \ifnum\luatexversion>111\relax\UseHook{env/\ccPrefix Heading/before}\fi
     \@ignorefalse
     \begingroup
       \@endpefalse
       \let\@currenvir\cch@currenvir
       \edef\@currenvline{\on@line}%
       \ifnum\luatexversion>111\relax
         \@execute@begin@hook{\ccPrefix Heading}%
       \fi
    }{}%
```

Some LaTeX kernel macros are saved, the namespace is set and counted groups from previous headings are reset.

```
  \cch@reserve
```

Handling of the optional argument

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

and treatment of heading-level specific style classes.

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

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

```
  \edef\ccCurSecName{#2}%
```

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

```
  \ccEvalType[#2]{Components}%
}
```

`\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}%
  }
```

`\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
  }
```

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

```
\def\cch@provide@quotes{%
  \ccDeclareComponent{QuoteBlock}{}{}%
  \ccDeclareComponentGroup{Quote}{%
    \ccDeclareCountedComponent{QuoteText}%
    \ccDeclareCountedComponent{QuoteSource}%
  }%
}
```

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

```
\def\cch@provide@authors{%
  \ccAddToRole{Author}{%
    \ccDeclareCountedComponent{AuthorContact}%
  }%
  \ccDeclareRoleBlock{Author}{ContactBlock}{author-contact-block-format}%
  \ccDeclareGroupHandler{Author}{%
    \ccIfComp{AuthorContact}{}{\ccComponent{AuthorContact}{\ccUseProperty{author-contact-format
        }}}{}%
  }%
  \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 (\ccPrefix#1), *ii* data sent to toc (\ccPrefix Toc#1), (iii) data sent to the page styles (\ccPrefix Run#1), and (iv) the data sent to the PDF bookmarks (\ccPrefix BM#1).

```
\def\cc@provide@overrides#1{%
  \ccDeclareComponent{Toc#1}{}{}% toc overrides
  \ccDeclareComponent{Run#1}{}{}% running overrides
  \ccDeclareComponent{BM#1}{}{}% bookmark overrides
}
```

5 Defaults

```
\ccAddToProperties{Heading}{%
  \ccSetProperty{interline-para}{}%
  \ccSetProperty{interline-para-sep}{\space}
  \ccSetProperty{heading-par}{%
    \ccIfProp{interline-para}{\if@noskipsec \leavevmode \fi}{}%
    \par
    \global\@afterindenttrue
  }%
  \ccSetProperty{after-heading-par}{\par \nobreak}% par commands at the end of non-inline headings
  \ccSetProperty{before-heading}{}%
  \ccSetProperty{title-face}{\bfseries}%
  \ccSetProperty{subtitle-face}{\normalfont}%
  \ccSetProperty{author-face}{\normalfont}%
  \ccSetProperty{quote-face}{\raggedleft}%
  \ccSetProperty{quote-source-face}{}%
  \ccSetProperty{quote-block-format}{%
    \bgroup
      \ccUseProperty{quote-face}%
      \ccUseComp{QuoteText}\par
      \ccIfComp{QuoteSource}{{\ccUseProperty{quote-source-face}--\space\ccUseComp{QuoteSource}}\
          par}{}%
    \egroup}
  \ccSetProperty{heading-block}
    {\ccUseProperty{main-title-format}%
     \ccIfComp{Subtitle}{{\ccUseProperty{subtitle-face}\ccUseComp{Subtitle}}\par}{}%
     \ccIfComp{AuthorNameList}{{\ccUseProperty{author-face}\ccUseComp{AuthorNameList}}\par}{}%
     \ccIfComp{QuoteBlock}{\ccUseComp{QuoteBlock}}{}%
     \ccIfComp{AffilBlock}{{\ccUseProperty{affil-block-face}\ccUseComp{AffilBlock}}\par}{}%
    }%
  \ccSetProperty{main-title-format}{%
    \ccUseProperty{title-face}%
    \ccaVstructStart{\ccCurSecName head}%
    \ccIfComp{Number}%
    {\ccUseProperty{hang-number}}%
    {\leftskip0pt}%
```

```
    \ccUseComp{Title}
    \ccaVstructEnd{\ccCurSecName head}%
    \par
  }
  \ccSetProperty{extended-heading}{%
    \ccIfComp{Abstract}
      {\par\vskip\baselineskip
       {\bfseries\ccIfComp{AbstractLabel}{\ccUseComp{AbstractLabel}}{Abstract}}\par
       {\itshape\small\ccUseComp{Abstract}}\par}
      {}%
    \ccIfComp{Keywords}
      {\par\vskip\baselineskip
       {\bfseries\ccIfComp{KeywordsLabel}{\ccUseComp{KeywordsLabel}}{Keywords}}\par
       {\itshape\small\ccUseComp{Keywords}\par}}
     {}%
   }%
  \ccSetProperty{before-skip}{\z@skip}% TODOC: values < 0pt use \minusvspace, else \addvspace. LaTeX's default behaviour of @afterindent is relocated to the after-indent property.
  \ccSetProperty{after-heading-block}{}%
  \ccSetProperty{before-heading-block}{\parindent\z@ \parskip\z@}%
  \ccSetProperty{toc-hook}{}% Called, after ToC and BM entries have been written to the .aux file
  \ccSetProperty{after-indent}{}%
  \ccSetProperty{margin-left}{}%
  \ccSetProperty{margin-right}{\@flushglue}%
  \ccSetProperty{after-skip}{1sp}%
  \ccSetProperty{indent}{auto}%
  \ccSetProperty{number-width}{}%
  \ccSetProperty{number-sep}{\space}%
  \ccSetProperty{number-align}{left}%
  \ccSetProperty{number-format}{%
    \bgroup
      \ccUseProperty{title-face}%
      \ccUseProperty{number-face}%
      \ccUseComp{Number}%
      \ccUseProperty{number-sep}%
    \egroup}
  \ccSetProperty{numbering}{auto}%
  %% running header
  \ccSetProperty{running-level}{}% override level for running title, name
  \ccSetProperty{running-heading}{%
    \ccIfComp{RunAuthorNameList}{\ccUseComp{RunAuthorNameList}:\space}{}%
    \ccUseComp{RunTitle}%
  }%
  %% ToC
  \ccSetProperty{no-toc}{false}% toc entries are generally disabled iff true
  \ccSetProperty{no-BM}{false}% bookmark entries are generally disabled, iff true
  \ccSetProperty{toc-margin-top}{\z@}% left indent of the whole entry
  \ccSetProperty{toc-margin-bottom}{\z@}% bottom margin of the whole entry
  \ccSetProperty{toc-margin-left}{auto}% left indent of the whole entry
  \ccSetProperty{toc-margin-right}{\@pnumwidth}% right margin of the whole entry
  \ccSetProperty{toc-title-face}{}% appearance of title
  \ccSetProperty{toc-indent}{auto}% offset of the first line of the entry. auto: hang indent by max-number-width for the level
  \ccSetProperty{toc-number-width}{}% current width of the TocNumber
  \ccSetProperty{toc-number-align}{left}% alignment of TocNumber within the hbox when hanging
  \ccPropertyLet{toc-number-face}{toc-title-face}% appearance of the TocNumber
  \ccSetProperty{toc-number-sep}{\enskip}% thing between TocNumber and TocTitle
  \ccSetProperty{toc-number-format}{% Format of the TocNumber
    \bgroup
      \ccUseProperty{toc-number-face}%
      \ccUseComp{TocNumber}%
```

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

6 Miscellaneous

6.1 Alternative paragraph separation

\ccPrefix NewPar is a user-level macro to have a vertical skip between two local paragraphs and no indent in the second one. The amount of vertical space between the paragraphs can be adjusted with the optional argument. If #1 is omitted, \ccnewparskip is inserted, which defaults to 1\baselineskip if the dimension isn't set to something other than 0pt in the preamble. This macro is intended to be used at the end of the first of the paragraphs.

```
\newdimen\ccnewparskip \AtBeginDocument{\ifdim\ccnewparskip=\z@\relax \ccnewparskip=1\
    baselineskip\relax\fi}
\csdef{\ccPrefix NewPar}{\@ifnextchar[{\cc@newpar}{\cc@newpar[\the\ccnewparskip]}}%]
\def\cc@newpar[#1]{%
  \ifhmode\par\fi
  \vskip#1\relax
  \@afterheading
}
```

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

\TitleBreak

```
\letcs\TitleBreak{\ccPrefix Break}
```

```
%</headings>
```

Modul 7

coco-notes.dtx

This file contains the code for foot- and endnote handling. It provides a switch between endnotes and footnotes as well as options to handle the resetting of footnote/endnote counters.

```
%<*endnotes>
```

```
%%
%% module for CoCoTeX that handles footnote/endnote switching.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-notes}
   [2024/03/23 0.4.1 le-tex coco notes module]
```

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

```
\newif\if@ccn@use@en \@ccn@use@enfalse
\newif\if@ccn@en@links \@ccn@en@linksfalse
```

package options:

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

```
\DeclareOption{endnotes}{\global\@ccn@use@entrue}
\DeclareOption{ennotoc}{\global\let\ccn@en@no@toc\relax}
\DeclareOption{resetnotesperchapter}{\global\let\ccn@reset@notes@per@chapter\relax}
\DeclareOption{endnoteswithchapters}{\global\@ccn@use@entrue\global\let\ccn@en@with@chapters\
    relax}
\DeclareOption{endnotelinks}{\global\@ccn@en@linkstrue}
\ProcessOptions
```

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

```
\RequirePackage{footnote}
```

Handling of endnotes:

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

```
  % Allow linking endnotes to their respective occurrence in the document.
  \if@ccn@en@links
    \global\newcount\endnoteLinkCnt \global\endnoteLinkCnt\z@
    \def\@endnotemark{%
      \leavevmode
      \ifhmode\edef\@x@sf{\the\spacefactor}\nobreak\fi
      \phantomsection%
      \label{endnote-\the\endnoteLinkCnt}%
      \hyperref[endnotetext-\the\endnoteLinkCnt]{\makeenmark}%
      \ifhmode\spacefactor\@x@sf\fi%
      \relax%
    }
  \fi
  \let\footnote=\endnote
  \def\enotesize{\normalsize}%
  \def\enoteformat{%
    % Create the label right at the start of the endnote text to prevent erroneous pointing to the next page
        .
    \if@ccn@en@links%
      \phantomsection%
      \label{endnotetext-\currentEndnote}%
    \fi
    \noindent
    \leavevmode
    \hskip-2em\hb@xt@2em{%
      \if@ccn@en@links
        \hyperref[endnote-\currentEndnote]{\@theenmark}\hss%
      \else
        \@theenmark\hss%
      \fi%
    }%
    \expandafter\parindent\ccn@parindent\relax\expandafter%
  }%
  \gdef\enoteheading{%
    \leftskip2em
  }%
  \def\printnotes{%
    \ifx\ccn@en@with@chapters\relax
      \ifnum\c@endnote>\z@
        \expandafter\global\expandafter\let\csname enotes@in@\the\realchap\endcsname\@empty
      \fi
    \fi
    \if@enotesopen
      \global\c@endnote\z@%
      \bgroup
      %\parindent\z@
      \parskip\z@
      \theendnotes
      \egroup
    \fi}
\else
  \newcount\c@endnote \c@endnote\z@
  \let\printnotes\relax
\fi
\newcount\realchap \realchap\z@
\ifx\ccn@en@with@chapters\relax
  \AtBeginDocument{%
    \ccAddToHook[heading]{before-hook-chapter}{%
      \ifnum\c@endnote>\z@\relax
        \expandafter\global\expandafter\let\csname enotes@in@\the\realchap\endcsname\@empty
      \fi
```

```
      \global\advance\realchap\@ne
      \global\c@endnote\z@
      \def\ccn@par@title{\ccIfComp{TocTitle}{\ccUseComp{TocTitle}}{\ccUseComp{Title}}}%
      \def\ccn@par@runtitle{\ccIfComp{RunTitle}{\ccUseComp{RunTitle}}{\ccUseComp{Title}}}%
      \addtoendnotes{%
        \noexpand\expandafter\noexpand\ifx\noexpand\csname enotes@in@\the\realchap\noexpand\
            endcsname\noexpand\@empty
          \bgroup
            \noexpand\leftskip\noexpand\z@
            \noexpand\begin{heading}\ifx\ccn@en@no@toc\relax[notoc]\fi{section}%
              \noexpand\ccComponent{Title}{\ccn@par@title}%
              \noexpand\ccComponent{RunTitle}{\ccn@par@runtitle}%
            \noexpand\end{heading}%
          \egroup
        \noexpand\fi}%
    }%
  }
\fi
\ifx\ccn@reset@notes@per@chapter\relax
  \AtBeginDocument{%
    \ccAddToHook[heading]{before-hook-chapter}{%
      \global\c@footnote\z@
      \global\c@endnote\z@
    }%
  }%
\fi
```

Here we make a small adjustment to the `\fn@fntext` macro from the `footnote` package by making it `\long` and therefore allowing `\par` inside its argument.

```
\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%
    }%
  }%
}
```

Adding artifact tagging to the footnoterule:

```
\pretocmd\footnoterule{\ccaVstructStart[document]{footnoterule}}{}{}
\apptocmd\footnoterule{\ccaVstructEnd{footnoterule}}{}{}
```

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:

```
\def\@xfootnote[#1]{%
   \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%
}
```

patching \@footnotemark

```
\pretocmd\@footnotemark{%
  \ccaStructStart{Span}\protected@xdef\@lt@fn@parent{\ccaGetCurStruct{idx}}%
  \ccaStructStart{footnotemark}%\addAltText{\@thefnmark}
}{}{}
\apptocmd\@footnotemark{%
  \ccaStructEnd{footnotemark}\ccaStructEnd{Span}%
}{}{}
```

patching \@makefntext

```
\pretocmd\@makefntext{%
  \ccaStructStart{footnotetext}%
  \ifx\@lt@fn@parent\@empty\relax\else\addToStruct{\@lt@fn@parent}\fi%
}{}{}
\apptocmd\@makefntext{%
  \ccaAddID{auto}\ccaStructEnd{footnotetext}%
}{}{}
```

Adding footnotemark and footnotetext PDF tags to the rolemap

```
\ccaAddRolemap{footnotemark}{Reference}
\ccaAddRolemap{footnotetext}{Note}
```

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{%
    \if@ccn@en@links
```

```
      \string\def\string\currentEndnote{\the\endnoteLinkCnt}%
    \fi%
    \@doanenote{\@theenmark}%
  }%
  \begingroup
    \def\next{#1}%
    \newlinechar='40
    \immediate\write\@enotes{\meaning\next}%
  \endgroup
  \immediate\write\@enotes{\@endanenote}%
  \if@ccn@en@links
    \global\advance\endnoteLinkCnt\@ne%
  \fi%
}
```

```
%</endnotes>
```

Modul 8

coco-script.dtx

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

```
%<*script>
```

```
%% module for CoCoTeX that handles script switching.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-script}
    [2024/03/23 0.4.1 CoCoTeX script module]
```

The argument of the `usescript` option is a list of script systems that are used in the document. It is used to determine the additional fonts that are to be loaded via the babel package.

```
\let\usescript\relax
\define@key{coco-script.sty}{usescript}{\def\usescript{#1}}
\ProcessOptionsX
\RequirePackage[quiet]{fontspec}
\RequirePackage[bidi=basic,silent]{babel}
\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
\message{^^J [coco-script Fonts loaded: \meaning\usescript]^^J}
```

1 Default fallback font

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]
\DeclareTextFontCommand\textfallback{\fallbackfont}
\DeclareTextFontCommand\textsffallback{\sffallbackfont}
```

2 Generic Fonts Declaration Mechanism

#1 Options passed to \babelprovide
#2 language
#3 argument(s) passed to \babelfont{rm}
#4 argument(s) passed to \babelfont{sf}

```
\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

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

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 Hebrew script

```
\ccDeclareBabelFont{hebrew}{[%
   Scale=MatchUppercase,%
```

```
    Path=./fonts/Noto/Hebrew/,%
    Ligatures=TeX,%
    BoldFont = NotoSerifHebrew-Bold.ttf]{NotoSerifHebrew-Regular.ttf}%
}{[%
    Scale=MatchUppercase,%
    Path=./fonts/Noto/Hebrew/,%
    Ligatures=TeX,%
    BoldFont = NotoSansHebrew-Bold.ttf]{NotoSansHebrew-Regular.ttf}%
}
```

3.5 Support for Arabic script

```
\ccDeclareBabelFont{arabic}{[%
    BoldFont = NotoNaskhArabic-Bold.ttf,%
    Path = ./fonts/Noto/Arabic/%
    ]{NotoNaskhArabic-Regular.ttf}}
  {[%
    BoldFont = NotoSansArabic-Bold.ttf,%
    Path = ./fonts/Noto/Arabic/%
    ]{NotoSansArabic-Regular.ttf}%
  }
```

3.6 Support for Greek script

```
\ccDeclareBabelFont{greek}{[%
    BoldFont = NotoSerif-Bold.ttf,%
    ItalicFont = NotoSerif-Italic.ttf,%
    BoldItalicFont = NotoSerif-BoldItalic.ttf,%
    Path = ./fonts/Noto/Serif/,%
    WordSpace = 1.25
    ]{NotoSerif-Regular.ttf}}
  {[BoldFont = NotoSans-Bold.ttf,%
    ItalicFont = NotoSans-Italic.ttf,%
    BoldItalicFont = NotoSans-BoldItalic.ttf,%
    Path = ./fonts/Noto/Sans/,%
    WordSpace = 1.25%
    ]{NotoSans-Regular.ttf}%
  }
```

3.7 Support for Syrian script

Since Babel does not support the Syrian script natively, we create a `babel-syriac.ini` file and include it, if it is needed. If we don't, the kerning and ligatures of Syriac text will be off.

Please note that due to the restrictions of the `listings`-Package, some Unicode characters cannot be displayed correctly in the documentation of the following code. Therefore, Syriac letters appear as "x" in the following source code listing.

```
\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 = xxxxxxxxxx
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 =
derive = 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}%
  }
```

3.8 Support for medieval scripts and special characters

only rm!

```
\babelfont{mdv}[%
Path=fonts/Junicode/,%
ItalicFont = Junicode-Italic.ttf,%
BoldFont = Junicode-Bold.ttf,%
BoldItalicFont = Junicode-BoldItalic.ttf,%
]{Junicode.ttf}
\def\mdvfont#1{{\mdvfamily#1}}
```

```
%</script>
```

Modul 9

coco-title.dtx

This file provides macros and facilities for title pages.

```
%<*title>
```

```
%%
%% module for CoCoTeX for maketitle.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-title}
   [2024/03/23 0.4.1 CoCoTeX title module]
\RequirePackage{coco-meta}
```

1 Top-Level Interface

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

```
\ccDeclareContainer{titlepage}{%
  \ccInherit {Components,Properties}{CommonMeta}%
  \ifarticle\ccInherit{Components}{article-meta}\fi
  \ccDeclareType{Components}{%
    \cct@simple@comps
```

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

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

```
    \cct@fundings@comp
    \cct@role@handlers{author}{Author}%
    \cct@declare@role{editor}{Editor}%
    \cct@declare@role{series-editor}{SeriesEditor}%
  }%
  \ccDeclareType{Properties}{}%
  \ccDeclareEnv[Meta]{\cct@meta}{\endcct@meta}%
}
```

`\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}}
```

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

```
\def\cct@meta{%
  \ccEvalType{Components}%
}
```

`\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{}
```

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

```
\def\endcct@meta{%
  \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
  \ccUseHook{document-meta-hook}%
  \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` is used to transfer the DocumentInfo meta date to the pdf writer.

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

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

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

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

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

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

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

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

```
        \ccIfAlly
```

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

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

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

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

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@bka
}
```

2.1 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
  \pdfstringdef\@title{\ccUseComp{Title}}%
  \cct@write@pdf@meta{pdftitle}{Title}{\ccUseComp{Title}}%
  \ccpgdefFromProperty{RunBookTitle}{run-book-title}%
}
```

2.2 Processing of the Document's Author

`\cct@title@process@bka` processes the document's main author or, if that doesn't exist, the main editor, or throws a warning if neither exist.

```
\def\cct@title@process@bka{%
  \@tempswatrue
  \begingroup
    \ccGobble
    \renewcommand\foreignlanguage[2]{{##2}}%
    \ccIfComp{AuthorPDFInfo}
      {\ccpgdefFromProperty{RunBookName}{AuthorPDFInfo}}
      {\ccIfComp{EditorPDFInfo}
         {\ccpgdefFromProperty{RunBookName}{EditorPDFInfo}}
         {\ifnum\ccAuthorCnt>\z@
            \@setpar{\@@par}%
            \ccggdefFromCountedComp{RunBookName}{Author}{author-list-pdfinfo-format}%
          \else
            \ifnum\ccEditorCnt>\z@
              \ccpgdefFromCountedComp{RunBookName}{Editor}{editor-list-pdfinfo-format}%
            \else
              \ccPackageWarning{transcript-title}{Meta Data}{No author or editor given!}%
              \@tempswafalse
            \fi
          \fi}}%
    \if@tempswa
      \pdfstringdef\@author{\csname\ccPrefix RunBookName\endcsname}%
      \cct@write@pdf@meta{pdfauthor}{Author}{\csname\ccPrefix RunBookName\endcsname}%
    \fi
  \endgroup
}
```

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

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

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

2.4 Including the XMP Meta Data

`\cct@title@insert@xmp` inserts the contents of the XMP meta data file into the pdf, if it exists. There are two versions, depending on whether coco-accessibility is active or not.

```
\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

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

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

`\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}%
  \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
    \ccUseHook[titlepage]{before-maketitle-hook}%
    \bgroup
      \ccSetContainer{titlepage}%
      \ccEvalType{Properties}%
      \ifarticle
        \cct@article@titlepage
      \else
        \ifjournal
          \cct@journal@titlepage
        \else
          \cct@book@titlepage
        \fi
      \fi
    \egroup
    \ccUseHook[titlepage]{after-maketitle-hook}%
  }%
}
```

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}%
}}
```

`\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{%
  \ccDeclareGlobalComponent[\jobname]{XmpFile} % File basename of the XMP file ('.xmp' is added
      automatically)
  %% Cover
  \ccDeclareGlobalComponent{Cover} % Path to Cover Image(!)
  %% Titles
  \ccDeclareGlobalComponent{Title} % Main Title
  \ccDeclareGlobalComponent{ShortTitle} % Shortened main title
  \ccDeclareGlobalComponent{RunTitle} % Shortened main title override for headers
  \ccDeclareGlobalComponent{AltTitle} % Alternative main title (e.g. for bastard title page)
  \ccDeclareGlobalComponent{Subtitle} % Sub Title
  \ccDeclareGlobalComponent{TitleNote} % Additional Title Information (contributor list)
  \ccDeclareGlobalComponent{RunNames} % Shortened list of names (authors and/or publishers)
  \ccDeclareGlobalComponent{AltNames} % Alternative list of names (e.g. for bastard title page)
  %% Series
  \ccDeclareGlobalComponent{Series} % Series Title
  \ccDeclareGlobalComponent{SubSeries} % Series Subtitle
  \ccDeclareGlobalComponent{SeriesNote} % Series Notes
  \ccDeclareGlobalComponent{Volume} % Series Volume
  \ccDeclareGlobalComponent{Number} % Series Number
  \ccDeclareGlobalComponent{EditorNameList} % Editor Text Line
  \ccDeclareGlobalComponent{SeriesEditorNameList} % Series Editor Text Line
  %% Publisher
  \ccDeclareGlobalComponent{Publisher} % Publisher Name
```

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

```
  \ccDeclareGlobalComponent{AddNoteI} % Additional information, title page I
  \ccDeclareGlobalComponent{AddNoteII} % Additional information, title page II
  \ccDeclareGlobalComponent{AddNoteIII} % Additional information, title page III
  \ccDeclareGlobalComponent{AddNoteIV} % Additional information, title page IV
}
```

4 Default Settings

```
\ccAddToProperties{titlepage}{%
  \ccSetProperty{article-title}{}%
  % 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}%
      \ccIfComp{FundName}{\ccUseComp{FundName}\\[1ex]}{}%
      \includegraphics[width=\ccUseProperty{fund-width}]{\ccUseComp{FundLogo}}}%
    \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}{%
    \ifmonograph
      \ccUseComp{AuthorNameList}%
    \else
      \ccUseProperty{EditorNameList}%
    \fi%
    \vskip\baselineskip
    \bgroup
      \ccUseProperty{title-face}\ccUseComp{Title}%
    \egroup
  }%
  \ccSetProperty{titlepage-ii}{%
    \ccGetComp{Editorial}%
    \ccGetComp{SeriesNote}%
    \ccGetComp{GenericMetaBlock}%
    \vfill
    \ccUseProperty{bio-output}%
  }%
  \ccSetProperty{titlepage-iii}{%
    \ifmonograph
```

```
      \ccUseComp{AuthorNameList}%
    \else
      \ccUseProperty{EditorNameList}%
    \fi%
    \par
    \ccUseProperty{title-format}
    \ccGetComp{Edition}%
    \ccGetComp{EditionNote}%
    \vfill
    \clearpage
  }%
  \ccSetProperty{titlepage-iv}{%
    \ccGetComp{Dedication}% Dedication
    \ccGetComp{Acknowledgements}% Dedication
    \ccUseProperty{imprint-format}%
    \ccUseProperty{funding-block}%
    \vfill
    \bgroup
      \ccUseProperty{imprint-face}%
      \ccIfComp{Biblio}{{\bfseries\ccGetComp{BiblioTitle}}\ccGetComp{Biblio}}{}%
      \ccUseProperty{imprint-sep}%
      \ccUseProperty{imprint}%
    \egroup
    \clearpage
  }%
  %% predefined face and format Properties
  \ccSetProperty{title-face}{\Huge\sffamily\bfseries}%
  \ccSetProperty{title-format}{%
    \bgroup
      \ccVstructStart{Title}% PDF 2.0
      \ccUseProperty{title-face}%
      \ccUseComp{Title}\par
      \ccVstructEnd{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}{%
    \ccIfComp{Copyright}
      {\ccUseComp{Copyright}\par}
      {\textcopyright\space\ccUseComp{Year}\space\ccUseComp{Publisher},\space\ccUseComp{PubPlace
          }\par}%
    }%
  \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}%
  }%
}
```

5 Accessibility Features

5.1 Output Intent and ICC Profiles

```
\ccWhenAlly{%
```

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

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

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

```
    \ccDeclareGlobalComponent{IccProfileFile}
```

`Compoent titlepage::IccComponents` holds the number of components in the color profile

```
    \ccDeclareGlobalComponent{IccComponents}
```

`Compoent titlepage::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):

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

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

`Property titlepage::output-intent` sends the output intent information to the ltpdfa package. It must contain of three data fields:

`profile` with the name of the to-be-embedded `.icc` file,
`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 *document-meta-hook*:

```
  \ccAddToHook[titlepage]{document-meta-hook}{\edef\x{\noexpand\ccaAddToConfig{intent}{\
      ccUseProperty{output-intent}}}\x}
```

5.2 Encoding of the PDF-A Conformance

As before, the parameters for the PDF conformity level are encoded via specific Components in the titlepage Container:

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

`Compoent titlepage::PDFAID` defines the PDF/A ID (Default: 2, meaning: PDF/A-2)

```
  \ccDeclareGlobalComponent[2]{PDFAID}%
```

`Compoent titlepage::PDFALevel` defines the PDF/A Level (Default: A, meaning PDF/A-2A)

```
  \ccDeclareGlobalComponent[A]{PDFALevel}%
```

`Compoent titlepage::PDFUAID` defines the PDF standard (Default: 1, meaning: PDF/UA-1). Use `\ccPrefix PDFUAID{}` (i.e. set it to nothing) to make the document conform to the PDF/A standard, but **not** to the PDF/UA standard.

```
  \ccDeclareGlobalComponent[1]{PDFUAID}}%
```

The checking if the values are valid, and the separation of the various parts of the standard is done via a lua script in the *document-meta-hook*. The `conformance` DocumentInfo nodes are only written, if *neither* `PDFAID`, *nor* `PDFALevel` is empty.

```
\ccAddToHook[titlepage]{document-meta-hook}{%
  \ccIfCompEmpty{PDFAID}{}{\ccIfCompEmpty{PDFALevel}{}{%
     \edef\x{\noexpand\ccaSetDocinfo{conformance}{%
        pdfaid=\ccUseComp{PDFAID};%
        level=\ccUseComp{PDFALevel}%
```

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

5.3 Titlepage Specific Role Maps

According to the "Tagged PDF Best Practice Guide" page by the PDF Association, the main title of the document should be mapped to <P> until the more appropriate <Title> tag becomes widely accepted with the PDF 2.0 Standard.

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

```
637 }%ccWhenAlly
```

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

Modul 10

coco-floats.dtx

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

```
%<*floats>
```

```
%%
%% module for CoCoTeX that extends floating objects.
%%
%% Maintainer: p.schulz@le-tex.de
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-floats}
    [2024/03/23 0.4.1 CoCoTeX floats module]
\DeclareOptionX{nofigs}{\global\let\ccf@no@figs\relax}
\ProcessOptionsX
```

1 Package Setup

1.1 Hard requirements

```
\RequirePackage{coco-common}
\RequirePackage{rotating}
\RequirePackage{grffile}
\RequirePackage{footnote}
\RequirePackage[Export]{adjustbox}
\usepackage{stfloats}
\setcounter{dblbotnumber}{5}
```

1.2 Document Class Option overrides

for automatic typesetting and float positioning, we set very high tolerances in macros from LATEX's standard

2 .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
```

2.1 Internal registers

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

```
\newbox \ccf@floatbox
\newbox \ccf@sub@box
```

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

```
\newcount\ccSubFloatCnt \ccSubFloatCnt=\z@\relax
\newcount\ccf@int@cnt \ccf@int@cnt\z@
\newcount\ccf@int@sub@flt@cnt \ccf@int@sub@flt@cnt\z@
```

Various dimension registers that store dimensions and spaces of floats and sub-floats:

- \ccf@sub@maxheight stores and self-updates the height of the largest sub-float inside a float
- \ccf@sub@sep is the space between sub-floats
- \ccf@total@width stores the cumulated overall width of the entire float
- \ccf@calc@width is an internal dimension used to calculate the ratio between mutiple sub-floats that should be scaled to the same height
- \ccf@total@height is the overall height of a float
- \ccf@total@depth is the overall depth of a float

```
\newdimen\ccf@sub@maxheight \ccf@sub@maxheight=\z@\relax
\newdimen\ccf@sub@sep \ccf@sub@sep=\fboxsep\relax
\newdimen\ccf@total@width \ccf@total@width=\textwidth\relax
\newdimen\ccf@total@height \ccf@total@height=\textwidth\relax
\newdimen\ccf@total@depth \ccf@total@depth=\textwidth\relax
\newdimen\ccf@calc@width \ccf@calc@width=\ccf@total@width\relax
```

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

```
\newskip\ccf@sep@top \ccf@sep@top=\z@\relax
\newskip\ccf@sep@bottom \ccf@sep@bottom=\z@\relax
```

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

```
\newdimen\ccf@margin@r \ccf@margin@r=\z@\relax
\newdimen\ccf@margin@l \ccf@margin@l=\z@\relax
\newdimen\ccf@margin@i \ccf@margin@i=\z@\relax
\newdimen\ccf@margin@o \ccf@margin@o=\z@\relax
```

Locally adjustable switch to allow captions to break across pages

```
\newif\if@ccf@break@capt \@ccf@break@captfalse
```

String definitions for Property value comparisons

```
\def\ccf@str@bottom{bottom}
\def\ccf@str@top{top}
```

2.2 AtBeginDocument hook

```
\AtBeginDocument{%
```

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

```
  \ifx\ccf@no@figs\relax
    \renewcommand\includegraphics[2][]{}%
  \fi
  \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%
  }%
}%
```

3 Internal macros

3.1 Generic resetter

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

`#1` the caption type (e.g., `figure`, `table`)
`#2` abbreviation of the caption list (e.g., standard LaTeX uses `lof` for the List of Figures, `lot` for the List of Tables)

```
\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
}
```

3.2 Internal macros that handle Attributes

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

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

```
\def\ccf@get@attr#1#2{%
  \if!#1!\else
    \ccParseAttributes{#2}{#1}%
    \ccIfAttr{#2}{class}
      {\global\letcs\ccf@current@class{cc@#2@attr@class}%
       \ccUseStyleClass{default}{\ccfCapType}%
       \expandafter\ccUseStyleClass\expandafter{\csname cc@#2@attr@class\endcsname}{\ccfCapType}}
      {}%
      \ccIfAttr{#2}{break-caption}{\@ccf@break@capttrue}{}%
  \fi
  \ccf@get@pos{#2}}
```

`\ccf@get@pos` is the handler for determining the floating position. Some float Properties and Attributes restrict and override the explicit float positions, e.g., fully rotated floats must be positioned in `p` mode (i.e., as float page). #1 is the caption type.

```
\def\ccf@get@pos#1{%
  \ccIfAttr{#1}{float-pos}
    {\letcs\ccf@floatpos{cc@#1@attr@float-pos}}
    {\let\ccf@floatpos\@empty}%
  \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
  \ccIfAttrIsStr{#1}{orientation}{landscape}
    {\linewidth\textheight
     \hsize\linewidth
     \def\ccf@floatpos{p}}
    {}}
```

`\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
    %\ifhmode\par\fi
```

```
  \else
    \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
      {\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}}}
      {\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}
```

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

```
\def\ccf@debug#1{%
  \ccIfAttr{#1}{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}}{}}
```

`\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
    \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}{}
      {\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}}
```

`\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}}
```

4 Float Container and Component Declarations

\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}}
```

\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
}
```

Container float

```
\ccDeclareContainer{float}{%
  \ccDeclareType{Components}{%
    \def\cc@counted@comp@scheme##1{##1-\the\ccSubFloatCnt}%
    \ccfMakeCompL{Caption}%
    \ccfMakeCompL{Legend}%
    \ccfMakeCompL{Source}%
    \ccfMakeCompL{Number}%
    \ccfMakeComp{RefLabel}%
    \ccfMakeComp{AltText}% neu: 2023-06-08; TODO: muss noch implementiert werden
    \ccfMakeComp{ListofEntry}%
  }%
  \ccDeclareType{Properties}{}%
}
```

\ccDeclareFloat is the user-level macro used to (re-)declare a (new) ccFloat environment.

#1 Name of the float Container from which the declared Container should inherit Properties (*optional*)
#2 top-level name of the float environment (e.g., \ccPrefix Table, \ccPrefix Figure)
#3 the tagging Role (*optional*, defaults to Div)
#4 caption type (e.g., table, figure)
#5 list (e.g., lot, lof)
#6 Property list

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

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

```
  \ifcsdef{cc@container@#2}{%
    \ccPackageInfo{Floats}{}{Appending to '#2'}%
    \ifx\ccf@parent\@empty\else
      \ccPackageError{Float}{Type}
        {Attempt to change parent of pre-existing float^^JContainer '#2'}
        {You cannot use the optional argument of \string\ccDeclareFloat\space for pre-existing^^J
            %
float containers!^^J^^J%
Use \string\ccAddToType{<Type>}{#2}{<code>}\space to alter the #2 container!}
    \fi
    \ccAddToType{Properties}{#2}{#6}%
```

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

```
    \ccAddToType{FloatEnvInfo}{#2}{%
      \def\ccfCapType{#4}%
      \def\ccf@cap@list@type{#5}%
    }%
  }{%
```

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

```
    \ccDeclareContainer{#2}{%
      \ccPackageInfo{Floats}{}{Declaring float '#2'}%
      \ifx\ccf@parent\@empty
        \ccInherit{Properties,Components}{float}
      \else
        \ccInherit{Properties,Components}{\ccf@parent}
      \fi
      \ccDeclareType{FloatEnvInfo}{%
        \ccSetContainer{#2}%
        \def\ccfCapType{#4}%
        \def\ccf@cap@list@type{#5}%
      }% /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}{%
        \ccUseProperty{float-handler}%
      }%
```

Generating the Handlers for the list-of entries and define the corresponding `l@` macros

```
      \ccf@generate@listof@handlers{#5}{#4}{#2}%
      \bgroup
        \def\cc@cur@cont{#2}%
        \cc@init@l@[list-of]{#4}{0}{#4}% Generate listof-Entries for first level floats
        \cc@init@l@[list-of]{#5}{1}{sub#4}% Generate listof-Entries for sub-floats
      \egroup
      \ccDeclareType{Properties}{#6}%
    }% /container
  }% /ifcsdef{cc@container@#2}
  \ifstrequal{Table}{#3}{}
    {\ifstrequal{Figure}{#3}{}
       {\ccaAddRolemap{#2}{#3}}}%
}
```

`\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@<list>@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@#1@extract@data\endcsname##1##2##3##4{%
    \ccSetContainer{#3}%
    \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}%
  }%
```

`\csname cc@<list>@print@entry\endcsname` The second dynamically defined macro is the entry renderer. It applies the Listof properties and selects the components to be printed. ##1 is the caption type of the float.

```
  \expandafter\gdef\csname cc@#1@print@entry\endcsname##1{%
    \bgroup
      \ccUseHook{list-of-before-hook-##1}%
      \ccUseProperty{list-of-before-entry}%
      \ccUseProperty{list-of-block}%
      \ccUseHook{list-of-after-hook-##1}%
      \ccUseProperty{list-of-after-entry}%
```

```
    \egroup}%
}
```

`\ccf@addcontentsline` fork of LaTeX's `\addtocontents` macro.

```
\def\ccf@addcontentsline{%
  \ccWhenComp{ListofEntry}{%
    \protected@write\@auxout
      {\ccGobble}%
      {\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}}
```

`\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\ccCompoennt{ListofSource}{\
      ccUseComp{ListofSource}}}}%
  \ifx\ccf@listof@entry\relax\else
    \bgroup
      \ccGobble
      \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}%
}
```

5 Label and Referencing mechanisms

5.1 Generation of Number Components

\ccf@create@counter checks for the various parameters that control whether or not a Number component is auto-generated for each sub-float.

```
\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{%
  \ccSetPropertyVal{float-number}{\csname cc@\cc@cur@cont @Number-0\endcsname}%
  \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}}%
}
```

5.2 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}}
```

6 Processing the Float

6.1 Common Float and Sub-Float Environments

`\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}%
    \ccf@get@attr{#1}{\ccfCapType}%
    \ccf@set@hsize
    \ccf@get@seps
    \ccEvalType{Components}%
    \ccUseProperty{before-float}%
    \ccf@set@env
    \ifx\ccf@floatpos\@empty\else\savenotes\fi
    \ignorespaces
    \@cc@is@finaltrue
  }
```

`\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
      \ccaStructStart{\cc@cur@cont}%
      \edef\ccf@parstruct@id{\ccaGetCurStruct{idx}}%
      \hsize\ccf@total@width
      \ccf@process
      \ccaStructEnd{\cc@cur@cont}%
      \par
    \egroup
    \ccSavePage
    \ccf@set@bot@sep
  \ccf@end@env
  \ccf@debug{\ccfCapType}%
  \ifx\ccf@floatpos\@empty\else\spewnotes\fi
\endgroup
\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}%
  }}%
\global\let\ccf@current@class\relax
}
```

`\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{%
  \ccUseProperty{subfloat-handler}%
  \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
}
```

6.2 Processing the Contents of the Float Environment

`\ccf@process` prints the contents of a float environment.

```
\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
    \@cc@is@finalfalse
    \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%
    \@cc@is@finaltrue
    \ccUseProperty{subfloat-render}%
    \let\ccf@prefix\@empty
  \fi
  \ccf@make@outer@caption{bottom}%
}
```

6.3 Caption mechanism

`\ccf@test@caption` tests if the current sub-float has any top or bottom caption that needs to be printed.

`#1` is the value of the sub-float counter
`#1` indicates if the caption belongs to the whole float (`capt`) or a sub-float (`subcapt`)
`#1` `top` or `bottom`

We compare the caption of the current `\SubCounter` level with a caption of a non-existing, negative, float level in case there is non-expandable material hard-coded into the `caption-#3` Property. If we were to compare the width of the `\hbox` with `\z@`, this scenario would give us false positives.

Warning: Long captions can cause the hbox's width to exceed `\maxdimen`. To avoid LaTeX errors in this case, we compare `sp` instead of `pt`. This, however, means that if the difference is less than 1pt, the test fails and no caption is printed!

```
\def\ccf@test@caption#1#2#3{%
  \@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
  \@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{%
      \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
        {\setbox\@tempboxa\vbox\bgroup\hsize\textheight}
        {\hskip\ccf@margin@l%
         \setbox\@tempboxa\vbox\bgroup\advance\hsize-\ccf@margin@l}%
```

```
    }%
  \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}%
    \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
      \@cc@is@finaltrue
      \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}{}%
  \fi}
```

`\ccfRenderSubFloats` iterates through the single sub-floats and renders them in a nice row. #1 is the subfloat counter, #2 is the Component name that contains the actual contents of the sub-float, for `\ccPrefix Figure` it is `Fig`, for `\ccPrefix Table` it is `Content`.

```
\long\def\ccfRenderSubFloats#1#2{%
  \leavevmode
  \savenotes
  \ifnum#1>\@ne\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
}
```

depending

```
\def\ccf@make@subcaption#1{%
  \expandafter\ifx\csname cc@has@\ccf@prefix capt@#1\endcsname\@empty
    \ccf@make@caption{#1}{\ccUseProperty{\ccf@prefix caption-valign-#1}}%
  \fi}
```

\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}}
```

\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 ratio between each sub-float's height and the height of the largest sub-float

```
\def\ccf@calc@sameheight{%
  \@tempdima=\z@\relax
  \@tempcnta=\z@\relax
  \ccf@calc@width=\ccf@total@width\relax
  \advance\ccf@calc@width-\ccf@margin@l\relax
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \edef\@tempa{\CalcRatio{\ccf@sub@maxheight}{\csname ccf@\cc@cur@cont @height-\the\@tempcnta\
        endcsname}}%
    \ifnum\the\@tempcnta>\@ne\relax
```

```
      \advance\ccf@calc@width-\ccf@sub@sep\relax%
    \fi
    \expandafter\@tempdimc\csname ccf@\cc@cur@cont @width-\the\@tempcnta\endcsname\relax
    \@tempdimb=\@tempa\@tempdimc\relax
    \expandafter\edef\csname cc@\cc@cur@cont @adj@width-\the\@tempcnta\endcsname{\the\@tempdimb}%
    \advance\@tempdima\@tempdimb
  }%
  \@tempcnta=\z@\relax
  \@tempdimb=\z@\relax
  \@tempdimc=\z@\relax
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \edef\@tempa{\CalcRatio{\csname cc@\cc@cur@cont @adj@width-\the\@tempcnta\endcsname}{\
        @tempdima}}%
    \expandafter\edef\csname cc@\cc@cur@cont @res@width-\the\@tempcnta\endcsname{\dimexpr\@tempa
        \ccf@calc@width\relax}%
    \@tempdimc\dimexpr\csname ccf@\cc@cur@cont @height-\the\@tempcnta\endcsname\relax
    \@tempdimc\dimexpr\@tempa\@tempdimc\relax
    \ifdim\@tempa\@tempdimb<\@tempdimc\@tempdimb\@tempdimc\relax\fi
  }%
  \expandafter\edef\csname cc@\cc@cur@cont @res@height\endcsname{\the\@tempdimb}%
}
```

7 Handlers for different float types

7.1 Handlers for generic floats

`\ccfGenericRender` is the Component that contains the contents of a generic float.

```
\def\ccfGenericRender{\ccUseComp{Content}}
```

`\ccfGenericHandler` is the generic content handler of a float

```
\def\ccfGenericHandler{\ccfMakeComp{Content}}
```

`\ccfSubGenericHandler` is the generic handler of a sub-float.

```
\def\ccfSubGenericHandler{}
```

7.2 Handlers for figures

`\ccfFigureHandler` tells the float module the name, main namespace, and main content Container of `tpFigure` type floats.

```
\def\ccfFigureHandler{\ccfMakeComp{Fig}}
```

`\ccf@create@natural` is the actual handler for sub-figures.

```
\def\ccf@create@natural{\ccUseComp{Fig}}
```

`\ccfSubFigureHandler` is the User-level macro that defines the handler for sub-figures. It also contains code for the `nofigs` package option.

```
\def\ccfSubFigureHandler{%
  \ifx\ccf@no@figs\relax
    \setbox\ccf@sub@box\hbox{\rule{0pt}{1pt}\rule{1pt}{0pt}}%
  \else
    \setbox\ccf@sub@box\hbox{\ccGobble\ccf@create@natural}%
  \fi}
```

`\ccfFigureRender` tells the module how `tpFigures` are to be rendered.

```
\def\ccfFigureRender{%
  \bgroup
    \ccIfAttrIsStr{\ccfCapType}{orientation}{landscape}
      {\hsize\dimexpr\textwidth-\ccf@margin@r-\ccf@margin@l\relax}%
      {}%
    \let\includegraphics\ccf@includesubgraphics
    \hskip\ccf@margin@l
    \ccWhenComp{AltText}{\ccaAddAltText{\ccUseComp{AltText}}}
    \strut\ccUseComp{Fig}\strut
  \egroup}
```

`\ccfSubFigureRender` tells the module how sub-floats of `tpFigure` type floats are to be rendered.

```
\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.

```
\def\ccf@includesubgraphics{\cc@opt@empty\@ccf@includesubgraphics}%
\def\@ccf@includesubgraphics[#1]#2{%
  \ifx\ccf@current@class\relax
    \def\@igopts{max width=\hsize,max height=\vsize}%
  \else
    \def\@igopts{width=\hsize}%
  \fi
  \if!#1!\else
    \def\@igopts{#1,width=\hsize}%
  \fi
  \gdef\ccf@fig@path{#2}%
  \if@cc@is@final\ccaAddPlacement{Block}\fi%
  \expandafter\ccf@ltx@includegraphics\expandafter[\@igopts]{#2}%
}
```

7.3 Handlers for tables

`\ccf@reserve@tabular` is a shell macro that stores the default macro definitions for various tabular mechanisms (currently, only plain *tabular*, `tabulary`, `tabularx`, and `htmltabs` are supported as content Component of `\ccPrefix Table`)

```
\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{%
  \expandafter\expandafter\expandafter\let\expandafter\csname orig@tabular#1\expandafter\
      endcsname\csname tabular#1\endcsname
  \expandafter\expandafter\expandafter\let\expandafter\csname orig@endtabular#1\expandafter\
      endcsname\csname endtabular#1\endcsname
  \expandafter\def\csname tabular#1\endcsname{%
    \global\setbox\ccf@floatbox
    \vbox\bgroup
      \if!#1!\else
        \let\tabular\orig@tabular
        \let\endtabular\orig@endtabular
      \fi
      \csname orig@tabular#1\endcsname}%
  \expandafter\def\csname endtabular#1\endcsname{\csname orig@endtabular#1\endcsname\egroup}%
}
```

\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 \ccPrefix Float\the\ccf@int@cnt Captop\endcsname\relax\else
          \htInitSkip\csname \ccPrefix Float\the\ccf@int@cnt Captop\endcsname
          \advance\htInitSkip\ccf@sep@top%
        \fi
        \expandafter\ifx\csname \ccPrefix Float\the\ccf@int@cnt Capbottom\endcsname\relax\else
          \htAddToBottom\csname \ccPrefix Float\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}%
}
```

\ccfTableHandler defines the content handler for \ccPrefix Table.

```
\def\ccfTableHandler{%
  \ccfMakeComp{Content}%
  \ccf@reserve@tabular
  }
```

\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}
```

\ccfSubTableHandler is the handler for sub-tables. So far, coco-floats.sty does not support tables to be sub-floats, so we just generate an Error message.

```
\def\ccfSubTableHandler{%
  \PackageError{coco-floats.sty}{ccSubFloat does not support sub-tables (yet)!}{You cannot yet
      use a tables within the 'ccSubFloat'!}%
}
```

\ccfTableRender defines the Renderer for \ccPrefix Table content Components

```
\def\ccfTableRender{%
  \ccfGetTableContent
  \ccComponent{Content}{\unvbox\ccf@floatbox}%
  \ccUseComp{Content}%
  \par\if@ccf@break@capt\else\nopagebreak\fi
  \vskip\dp\strutbox
}
```

\ccfSubTableRender Is the Renderer for table sub-floats (which we don't allow yet, so this definition is un-used at the moment)

```
\def\ccfSubTableRender{%
  \cc@iterate{\@tempcnta}{\@ne}{\ccSubFloatCnt}{%
    \ccfGetTableContent
    \@cc@is@finalfalse
    \ccComponent{Content}{\unvbox\ccf@floatbox}%
    \@cc@is@finaltrue
    \ccfRenderSubFloats{\the\@tempcnta}{Content}%
  }}
```

7.4 Helpers

\ccFloatBarrier can be used to force all pending floats to be printed at the next shipout.

```
\def\ccFloatBarrier{\AtBeginShipoutNext{\clearpage}}
```

8 Default Settings

```
\ccAddToType{Properties}{float}{%
  \ccSetProperty{auto-number-prefix}{\csname\ccfCapType name\endcsname}%% Prefix for auto-generated
      Number components
  \ccSetProperty{auto-number-prefix-sep}{~}%% Prefix for auto-generated Number components
  \ccSetProperty{intext-skip-top}{\intextsep}%% non-float sep top
  \ccSetProperty{intext-skip-bottom}{\intextsep}%% non-float sep bottom
  \ccSetProperty{float-skip-top}{\z@}%% float sep top
  \ccSetProperty{float-skip-bottom}{\z@}%% float sep bottom
  \ccSetProperty{sub-float-sep}{\ccf@sub@sep}%% space between sub-floats
  \ccSetProperty{margin-inner}{\z@}%% left margin on odd pages/right margin on even pages
  \ccSetProperty{margin-outer}{\z@}%% right margin on odd pages/left margin on even pages
```

```
760  \ccSetProperty{margin-left}{\z@}%% left margin
761  \ccSetProperty{margin-right}{\z@}%% right margin
762  \ccSetProperty{before-float}{\parindent\z@}%% executed before content is evaluated
763  \ccSetProperty{float-handler}{\ccfGenericHandler}% Alias for the caption type specific content
         handler
764  \ccSetProperty{subfloat-handler}{\ccfSubGenericHandler}% Alias for the caption type specific
         content handler
765  \ccSetProperty{float-render}{\ccfGenericRender}% Alias for the caption type specific content printer
766  \ccSetProperty{subfloat-render}{\ccfGenericRender}% Alias for the caption type specific content
         printer for sub-floats
767  \ccSetProperty{subfloat-same-height}{}% if true, the subfloat must/can be adjusted to the same
         heights
768  %% captions
769  \ccSetProperty{caption-face}{}% style applied to top and bottom captions
770  \ccSetProperty{caption-face-top}{}%% style applied to top captions
771  \ccSetProperty{caption-face-bottom}{}%% style applied to bottom captions
772  \ccSetProperty{source-face}{}% Format of source, additional to caption-format
773  \ccSetProperty{legend-face}{}% Format of legend, additional to caption-format
774  \ccSetProperty{caption-sep-top}{\z@}%% vertical space between top caption and content
775  \ccSetProperty{caption-sep-bottom}{\z@}%% vertical space between content and bottom caption
776  \ccSetProperty{caption-top}{%
777    \ccIfComp{Number}{{\ccUseProperty{number-face}\ccUseComp{Number}\ccUseProperty{number-sep
           }}}{}%
778    \ccUseComp{Caption}%
779  }%
780  \ccSetProperty{caption-bottom}{%
781    \ccIfComp{Legend}{{\ccUseProperty{legend-face}\ccUseComp{Legend}}}{}%
782    \ccIfComp{Source}{%
783      \ccIfComp{Legend}{\par\nopagebreak}{}%
784      {\ccUseProperty{source-face}%
785       \ccUseComp{Source}}}{}}%
786  \ccPropertyLet{subcaption-face}{caption-face}% style applied to top and bottom captions
787  \ccSetProperty{subcaption-face-top}{\ccUseProperty{caption-face-top}}%% style applied to top
         captions
788  \ccSetProperty{subcaption-face-bottom}{\ccUseProperty{caption-face-bottom}}%% style applied to
         bottom captions
789  \ccSetProperty{subcaption-add-sep-top}{\z@}%% additional vertical space between top caption and top
         sub-caption
790  \ccSetProperty{subcaption-add-sep-bottom}{\z@}%% additional vertical space between bottom sub-
         caption and bottom caption
791  \ccSetProperty{subcaption-sep-top}{\ccUseProperty{caption-sep-top}}%% vertical space between top
         sub-caption and content
792  \ccSetProperty{subcaption-sep-bottom}{\ccUseProperty{caption-sep-bottom}}%% vertical space
         between content and bottom sub-caption
793  \ccSetProperty{subcaption-top}{\ccUseProperty{caption-top}}% in case, sub-float captions diverge
         from main caption
794  \ccSetProperty{subcaption-bottom}{\ccUseProperty{caption-bottom}}% in case, sub-float captions
         diverge from main caption
795  \ccSetProperty{subcaption-valign-top}{top}%% vertical alignment of neighboring top-placed sub-
         captions
796  \ccSetProperty{subcaption-valign-bottom}{top}%% vertical alignment of neighboring bottom-placed sub-
         captions
797  %% Numbers
798  \ccSetProperty{numbering}{auto}%% automatic numbering for missing Number component
799  \ccSetProperty{sub-numbering}{}%% automatic numbering for missing Number component in real(!) sub-
         floats
800  \ccSetProperty{number-sep}{\enskip}% Separator between label and caption
801  \ccSetProperty{number-face}{\bfseries}% Format of number, additional to caption-format
802  \ccSetProperty{sub-number-sep}{\,}%% when sub-captions, this is placed between the float counter and
         the sub-float counter
803  \ccSetProperty{sub-number-style}{alph}%% counting style of subcaption counters
```

```
  \ccSetProperty{sub-number-face}{}%% format of subcaption counters
  \ccSetProperty{sub-number-before}{(}% stuff that is put immediately before the sub counter
  \ccSetProperty{sub-number-after}{)}% stuff that is put immediately after the sub counter
  \ccSetProperty{sub-number-format}{% Format of the sub number
    \ccUseProperty{float-number}%
    \ccUseProperty{sub-number-sep}%
    \ccUseProperty{sub-number}}%
  %% Labels
  \ccSetProperty{label-pos}{top}% The position of the caption, before which the \ref{} target should be
      placed
  \ccSetProperty{sublabel-pos}{top}% The position of the subcaption, before which the \ref{} target
      should be placed
  %% List-of entries
  \ccSetProperty{list-of-page-sep}{\dotfill}%
  \ccPropertyLet{list-of-number-face}{list-of-caption-face}%
  \ccSetProperty{list-of-number-sep}{\enskip}%
  \ccSetProperty{list-of-number-align}{left}%
  \ccSetProperty{list-of-number-format}{%
    \bgroup
      \ccUseProperty{list-of-number-face}%
      \ccUseComp{ListofNumber}%
      \ccUseProperty{list-of-number-sep}%
    \egroup}%
  \ccSetProperty{list-of-parfillskip}{-\rightskip}%
  \ccSetProperty{list-of-margin-right}{\@pnumwidth \@plus 1fil}%
  \ccSetProperty{list-of-margin-left}{auto}%
  \ccSetProperty{list-of-indent}{auto}% list-of-float appearance
  \ccSetProperty{list-of-block}{%
    \ccUseProperty{list-of-caption-face}%
    \ccIfComp{ListofNumber}
      {\ccUseComp{list-of-hang-number}}
      {\leftskip0pt}%
    \ccUseComp{ListofCaption}%
    \ccUseProperty{list-of-page-sep}\ccUseComp{ListofPage}%
  }% list-of-float appearance
  \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
  \ccSetProperty{list-of-after-entry}{\par}% list-of-float appearance
}
```

`Container Figure` defines the defaults for the `\ccPrefix Figure` Container.

```
\ccDeclareFloat{Figure}[Figure]{figure}{lof}{%
  \ccSetProperty{subfloat-same-height}{true}% if true, the subfloat must/can be adjusted to the same
      heights
  \ccSetProperty{float-handler}{\ccfFigureHandler}%
  \ccSetProperty{subfloat-handler}{\ccfSubFigureHandler}%
  \ccSetProperty{float-render}{\ccfFigureRender}%
  \ccSetProperty{subfloat-render}{\ccfSubFigureRender}%
}
```

`Container Table` defines the default Properties of the `\ccPrefix Table` Container.

```
\ccDeclareFloat{Table}[Table]{table}{lot}{%
  \ccSetProperty{subcaption-valign-top}{bottom}%
  \ccSetProperty{float-handler}{\ccfTableHandler}%
  \ccSetProperty{subfloat-handler}{\ccfSubTableHandler}%
  \ccSetProperty{float-render}{\ccfTableRender}%
  \ccSetProperty{subfloat-render}{\ccfSubTableRender}%
}
```

```
%</floats>
```

Modul 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
%%
%% lualatex - texlive > 2019
%%
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-frame}
    [2024/03/23 0.4.1 coco-frame]\relax
```

1 Top-Level Interface

```
\let\cc@frame@mode n
\define@choicekey{coco-frame.sty}{frame}[\cc@frame@mode\nr]{none,crop,frame}{%
  \ifcase\nr\relax% none
    \let\cc@frame@mode n
  \or% crop
    \let\cc@frame@mode p
  \else% frame
    \let\cc@frame@mode w
  \fi
}%
\ProcessOptionsX\relax
```

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>
```

Modul 12

coco-lists.dtx

This module provides handlers for list-like environments like item lists, enumerations, glossaries and descriptions.

```
%<*lists>
```

Note: The `coco-lists` module diverges somewhat from the other CoCoTEX modules insofar as that its main Container does not follow the CoCoTEX's usual "collect all–process later" approach, but all Properties are processed at the beginning of each Container's instances and the contents are processed as they are parsed by the `\LaTeX` interpreter, just like "reguar" LATEX lists. Configuration of lists, however, follows the CoCoTEX playbook.

1 Preamble

```
\NeedsTeXFormat{LaTeX2e}[2018/12/01]
\ProvidesPackage{coco-lists}
   [2024/03/23 0.4.1 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
\DeclareOptionX{replace}{\global\@ccl@replacetrue}%
```

The option inherit defines how nested lists inherit their properties. Currently, there are two ways: `common`: All nested lists of the same type inherit only from the same, generic type definition; `conseq`: nested lists of the same type inherit from the next-higher level list of the same type, and from the generic type definition.

For example, if `inherit=common`, 3rd level *itemize* and 2nd level *itemize* both inherit only the property values of the same generic *itemize* list type. If `inherit=conseq`, 3rd level inherits the property lists from 2nd level *itemize*.

Since inheritance is a transitive relation, 3rd level *itemize* will ultimately also inherit the Properties from generic *itemize*, but in contrast to `common`, `conseq` allows 2nd level *itemize* to override some Properties of generic *itemize*, which will be propagate down to 3rd level *itemize*, while with `inherit=common`, the override on 2nd level *itemize* would have no effect on 3rd level *itemize*.

```
\def\ccl@ih@common{common}
\def\ccl@ih@conseq{conseq}%
\let\ccl@inherit\ccl@ih@common
\define@choicekey{coco-lists.sty}{inherit}[\@ccl@inherit\nr]{conseq,common}{%
  \ifcase\nr\relax% conseq: nested lists of the same type inherit only from the previous level
    \global\let\ccl@inherit\ccl@ih@conseq
  \fi
}
```

The nesting option sets whether the nesting level of a list should be counted list-specific (value `local`), or globally (value `global`, default).

```
\def\ccl@str@local{local}%
\def\ccl@str@global{global}%
\let\ccl@nesting\ccl@str@global
\define@choicekey{coco-lists.sty}{nesting}[\@ccl@nesting\nr]{local,global}{%
  \ifcase\nr\relax% local
    \global\let\ccl@nesting\ccl@str@local
  \fi
}
\ProcessOptionsX
```

2 The List Container

The `List` Container is the most abstract Container for lists.

```
\ccDeclareContainer{List}{%
  \ccDeclareType{Properties}{%
    %% list formatting
    \ccSetProperty{before-list}{% at the very beginning of each (nested) list
      \if@noskipsec \leavevmode \fi
      \ifvmode\else
        \unskip \par
      \fi
      \ccaStructStart{L}% Start Tag for the (nested) list
    }%
    \ccSetProperty{after-list}{% after each (nested) list
      \ccUseProperty{after-item}%
      \ccaStructEnd{L}% end tag for the (nested) list
    }%
    %% list margins
    \ccSetProperty{margin-top}{\z@}% vertical space after the list.
    \ccSetProperty{margin-bottom}{\z@}% vertical space before the list.
    \ccSetProperty{margin-left}{\csname leftmargin\@roman\cclCurDepth\endcsname-\ccUseProperty{
        label-sep}+\ccUseProperty{prev-margin-left}}% horizontal space to the left of each item, from
        left boundary of the page area (auto=width of widest label + prev-margin-left, top-level-list-wise)
    \ccSetProperty{max-label-width}{.33\textwidth}% maximum margin reserved for list labels
    \ccSetProperty{margin-right}{\z@}% horizontal space to the right of each list item
    %% between list items
    \ccSetProperty{item-sep}{\z@}% vertical space between two adjacent list items (real: this value +
        par-skip)
    \ccSetProperty{after-indent}{false}% whether the paragraph after the list should have an indent (
        true) or not (false)
    \ccSetProperty{at-begin-item-body}{\ccaVstructStart{LBody}}% right at the beginning of a new item
         body
    \ccSetProperty{at-end-item-body}{\ccaVstructEnd{LBody}}% at the very end of an item body, but
        before \par
    \ccSetProperty{after-item}{% material after each item
      \ccUseProperty{at-end-item-body}%
      \ccaVstructEnd{LI}% Close list item tags
      \par}%
    \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
    }%
    \ccSetProperty{item-offset}{% Setting the label indent and first-line 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
    }%
    %% inside list items
    \ccSetProperty{par-indent}{\parindent}% indent of the first line of a *new* paragraph inside a list item
    \ccSetProperty{par-fill-skip}{\@flushglue}% skip at the end of the last line of each paragraph inside a list item
    \ccSetProperty{par-skip}{\z@}% vertical space between two adjacent paragraphs inside a list item
    %% label formatting
    \ccSetProperty{label}{\ccUseComp{Label}}% The Label Component is set via the optional argument of \Item, otherwise it is generated
    \ccSetProperty{indent}{-\dimexpr\csname leftmargin\@roman\cclCurDepth\endcsname-\ccUseProperty{label-sep}\relax}% indent of each list item's first line (relative to margin-left) NOTE: auto-global is valid, but it causes *all* lists -- despite the nesting level -- to have the same left margin and indent!
    \ccSetProperty{label-sep}{.5em}% horizontal skip between each item's label and its content
    \ccSetProperty{label-face}{}% font of the item's label
    \ccSetProperty{label-align}{left}% alignment of label within its \hbox
    \ccSetProperty{label-format}{% format of the label itself
      \ccUseProperty{label-face}%
      \ccaVstructStart{Lbl}% Start Tag for the item's label
      \ccUseProperty{label}%
      \ccaVstructEnd{Lbl}% End tag for the item's label
    }%
    \ccSetProperty{label-box}{% hbox that contains and aligns the label
      \hbox to \cclItemIndent{%
        \ccIfPropVal{label-align}{left}{}{\hss}%
        \ccUseProperty{label-format}%
        \ccIfPropVal{label-align}{right}{}{\hss}}%
    }%
    \ccSetProperty{item-format}{% material at the beginning of a new item
      \ccUseProperty{before-item}%
      \ccUseProperty{item-offset}%
      \ccUseProperty{label-box}%
      \hskip\ccUseProperty{label-sep}%
    }%
  }%
  \ccDeclareType{Components}{%
    \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}{%
      \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@}{#3}%
}
```

`\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<\@ne\relax
    \ccPackageError{lists}{Nesting}{Invalid nesting level!}{You cannot declare nesting levels
        less than 1!}%
  \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
```

```
}
\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. 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\cclID \cclID\z@\relax
\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:

```
  \ccSetPropertyX{prev-margin-left}{\the\leftskip}%
  \ccSetPropertyX{prev-margin-right}{\the\rightskip}%
  \ccEvalType[\ccl@cur@cont]{Properties}%
  \edef\ccl@list@type{\ccUseProperty{list-type}}%
```

Processing of the optional argument.

```
  \cclUseAttributeHandler{#1}%
```

The macro that separates the items of the list is defined locally so that we can use Properties:

```
  \cclCalculateMarginLeft%
  \cclCalculateVMargin{top}%
  \cclCalculateVMargin{bottom}%
  \csdef{\ccPrefix Item}{\cc@opt@empty\ccl@item}%
  \def\ccl@item[##1]{%
    \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%
  }%
```

If default LaTeX macros are replaced per package option, `\item` is made into a copy of the local definition of `\ccPrefix Item`.

```
  \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:

```
  \ccUseProperty{before-list}%
  \ccUseProperty{int-margin-top}%
  \leftskip\dimexpr\ccUseProperty{margin-left}+\ccUseProperty{label-sep}\relax%
  \rightskip\dimexpr\ccUseProperty{margin-right}\relax%
}
```

`\endcc@list` is expanded at the end of each List Container's respective environment. It basicly calls the after-list Property one last time and decrements the depth counter(s).

```
\def\endcc@list{%
  \ccUseProperty{after-list}%
```

```
  \ccl@decr@count%
  \ccl@advance@depth\m@ne%
  \ccUseProperty{int-margin-bottom}%
  \ifnum\cclCurDepth=\z@\relax
    \ccIfPropVal{after-indent}{false}{%
      \global\@afterindentfalse
      \global\everypar{%
        \if@afterindent \else
          {\setbox\z@\lastbox}%
        \fi
        \global\everypar{}}}{}%
  \fi
}
```

`\cclCalculateVMargin` generates a macro that realizes the internal vertical margin 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

```
\ccDeclareListType{unnumbered}{%
```

`\ccl@make@label@unnumbered` creates the label 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

For unnumbered lists there is one new Property, `default-label` which defines a fallback label.

```
}{\ccSetProperty{default-label}{-}}
```

Itemize-Style Default Lists

```
\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
```

```
\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

The new Property `enum-type` controls how the item counter is rendered when it is not given explicitly with the optional argument of `\item`. The default values are borrowed from LaTeX's default enumerate types and defined below.

```
  \ccSetProperty{enum-type}{arabic}%
```

Properties with Deviating Default Values

By default, numeric labels are followed by a period to accommodate LaTeX customs.

```
  \ccSetProperty{label}{\ccUseComp{Label}.}}
```

Available Counting Styles

`\ccl@label@type@arabic`, `\ccl@label@type@roman`, `\ccl@label@type@Roman`, `\ccl@label@type@alph`, `\ccl@label@type@Alph` are wrappers for all the available counter transformators built-in the `\LaTeX` kernel.

```
\def\ccl@label@type@arabic{\@arabic}
\def\ccl@label@type@roman{\@roman}
\def\ccl@label@type@Roman{\@Roman}
\def\ccl@label@type@alph{\@alph}
\def\ccl@label@type@Alph{\@Alph}
```

Enumerate-Style Default Lists

```
\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

```
\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

There is a new Property, `label-growth`, which can be set to `up` or `down` and which controls the direction labels "grow" into when they need more space than `max-label-width`. On 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.

Properties with Deviating Default Values

To accommodate for the new option, the `label-box` has a conditional that switches between regular `\hbox` labels and the two `\vbox` variants described above..

The Properties `margin-left` and `indent` of text-type lists are by default set to `auto`.

```
}{%
  \ccSetProperty{label-growth}{up}% or 'down'; where a multi-line label should "grow" to.
  \ccSetProperty{indent}{auto}%
  \ccSetProperty{margin-left}{auto}%
  \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

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.

```
393 \if@ccl@replace
394   \letcs\itemize{\ccPrefix Itemize}
395   \letcs\enditemize{end\ccPrefix Itemize}
396   \letcs\enumerate{\ccPrefix Enumerate}
397   \letcs\endenumerate{end\ccPrefix Enumerate}
398   \letcs\description{\ccPrefix Description}
399   \letcs\enddescription{end\ccPrefix Description}
400 \fi
```

```
401 %</lists>
```