

## Contents

<b>1 Overview .....</b>	<b>2</b>
1.1 Installation and Usage .....	2
1.1.1 Installation .....	2
1.1.2 Usage .....	2
1.2 Design Goals and Purpose .....	3
1.2.1 Basic Concepts .....	3
1.2.2 Concepts from Object-Oriented Programming .....	4
1.2.3 Implementation of CoCoTeX Concepts in LaTeX Documents .....	4
1.3 Overview: Modules .....	5
1.3.1 User-Level Modules .....	5
1.3.2 Backend Modules .....	6
<b>Index .....</b>	<b>7</b>
Index of Properties .....	8
Index of L <sup>A</sup> T <sub>E</sub> X macros and environments .....	9

# 1 Overview

## 1.1 Installation and Usage

### 1.1.1 Installation

The current build of the package can be obtained from GitHub:

```
git clone https://github.com/transpect/CoCoTeX.git
```

The actual source files can be found in the `src` sub-folder.

The most recent stable version can be found in the `releases` folder. It contains the `cocotex.dtx` file, its corresponding `cocotex.ins` file, and this Manual as a pre-rendered PDF.

The package is installed via

```
latex cocotex.ins
```

This will create the `cocotex.cls` file, as well as some additional modules that follow the naming convention `coco-<module>.sty`. These modules will be explained in greater detail below in [section](#) .

The documentation of the framework's source code can be created via

```
pdflatex cocotex.dtx
```

**Note:** You *must* use `pdflatex` in order to create the source code documentation!

**Note:** The source code documentation is a technical breakdown of the framework's source code; it is not the same document as the more user-oriented Manual you are currently reading.

### 1.1.2 Usage

CoCoTeX follows a modular design. It comes with several `.sty` files that can be used independently from another. However, there is also a L<sup>A</sup>T<sub>E</sub>X Document Class file `cocotex.cls` which can be used to load the whole framework at once.

#### Using `cocotex.cls`

The `cocotex.cls` serves as stand-in for the L<sup>A</sup>T<sub>E</sub>X default document classes `article` and `book`. It is called with the usual L<sup>A</sup>T<sub>E</sub>X command:

```
\documentclass[<options>]{cocotex}
```

The actual document type can be set with the `pubtype` option:

```
\documentclass[pubtype=<mono|article|collection|journal>]{cocotex}
```

The allowed values are:

<code>mono</code>	for monographs, i.e., books that are written by one or multiple authors as a whole,
<code>article</code>	for single journal articles,
<code>collection</code>	for books that are collections of contributions of multiple authors, and
<code>journal</code>	for journals, i.e., collections of multiple journal articles.

### Using Single Modules

CoCoTeX is modular. That means you can use selected modules as packages together with L<sup>A</sup>T<sub>E</sub>X's default or other third-party document classes. Modules are included like any other package, e.g.,

```
\RequirePackage[<options>]{coco-floats}
\RequirePackage[<options>]{coco-headings}
\RequirePackage[<options>]{coco-title}
```

## 1.2 Design Goals and Purpose

CoCoTeX is a programming framework for L<sup>A</sup>T<sub>E</sub>X developers, who need to build and maintain a number of (not too) different publisher-specific style sheets and partly or fully automated typesetting processes. Its original purpose is to serve as a rendering backend for the typesetting tool *xerif*, but it is also usable as a standalone extension to plain L<sup>A</sup>T<sub>E</sub>X.

The following features are the main design goals of the CoCoTeX framework:

- Handling of different document types in the same stylesheet:
  - journal articles
  - whole journals
  - chapters by different authors in proceedings and collections,
  - text collections and proceedings, and
  - monographs by (a) single author(s).
- Handling of recurring complex elements that are difficult to set-up using standard-L<sup>A</sup>T<sub>E</sub>X, e. g.
  - headings of all levels with authors, subtitles, quotes, etc.;
  - a four-way distinction of material in a heading's title, its pendant in headers and footers, and their entry in the table(s) of contents, and in the PDF bookmarks; and
  - the possibility to provide classes of text components like headings and floats, similar to classes in HTML/CSS; and
  - the structured handling of meta-data, especially for titlepages.

The framework introduces some new concepts into L<sup>A</sup>T<sub>E</sub>X programming that are extensively influenced by object-oriented design principles. The name CoCoTeX is derived from two of those concepts, namely Containers and Components. In the next sections, those and other concepts are explained in more detail.

### 1.2.1 Basic Concepts

One design goal of the CoCoTeX Framework is to provide an easy and unified way to configure the typesetting of blocks of inter-connected data.

For instance, take *headings*: They always consist of a *Title*, but also may have some sort of *Numbering*, some have a *Subtitle*, some might have a dedicated *Author*, some are followed by a *Quote* or a *Motto*. They may re-appear (partly) in the head-line of a page, as well as in the table of contents, in some cases with slightly altered data.

#### Containers

Such a bundle of structured information in the CoCoTeX framework is referred to as a **Container**. In the aforementioned example, the information pieces Title, Subtitle, Author etc., together form a unit heading.

Containers can be derived from one another. For instance, the abstract concept heading might be extended to various levels, like section, chapter, paragraph, or part. Some of the abstract constituents, like Title or Number, are shared among all those derivations of the concept heading, while others might not. E.g. Author is usually used on chapters, but rarely on sections or even deeper levels. The mechanism to pass certain properties or constituents from one Container to another is called **Inheritance**.

### Components

The inter-connected pieces of information that constitute a Container are called **Components** in the **CoCoTeX** framework. Most basic components are simple  $\text{\LaTeX}$  macros that take one argument for the content that is to be stored inside that Component for the respective Container.

Some Components can be collections of other Components. An Author Component of a heading Container, for example, can contain a first, a middle and a last name, an academic title, an affiliation, or an email address, among other things. Those complex Components are called **Component Groups**.

Some Components may occur multiple times in the same parent Container. A good example are multiple Authors that contributed to the same chapter in a collection. Those Components are called **Counted Components**. Note that despite the name, it is not necessarily the case that those Counted Components are numbered or even ordered in any way. Rather, “Counted” refers to the way they are processed internally. Due to the way both concepts are implemented in **CoCoTeX**, Group Containers are always also Counted Containers and vice-versa, so both terms might be used interchangeably.

### Properties

While headings of the same level are usually rendered in the same way for a given publisher style, the actual typeface may vary depending on the Components that are actually filled with content for a given heading: A numbered heading might look slightly different than one without a number; a heading with a subtitle might have different spacing than one without a subtitle, and so on.

How Components are processed and ultimately rendered is controlled by so-called **Properties**. Properties are mostly short pieces of  $\text{\LaTeX}$  code that are usually set by the stylesheet developer according to a publisher’s guidelines and requirements. One of the design goals of **CoCoTeX** is to keep the code behind those Properties as comprehensible and pointed as possible.

### Types, Scope and Modular Inheritance

Properties and Components can be seen as Containerspecific Data **Types**. They are only defined within the scope of their parent Container and are usually not accessible from the outside. When a new Container is declared, it can inherit the Data Types from one or multiple other containers.

## 1.2.2 Concepts from Object-Oriented Programming

**Containers** are comparable to the concept of *classes* in object-oriented programming. A concrete heading in a document is an *instance* of that class. **Components** serve as *class variables*, **Properties** can be seen as *instance methods*. **Types** can include macros and control sequences that are somewhat comparable to *class methods*.

The **Inheritance** and **Type** mechanisms are comparable to *Mixins* in some object-oriented programming languages like Ruby.

## 1.2.3 Implementation of CoCoTeX Concepts in LaTeX Documents

In the **CoCoTeX** framework, *Containers* are realised in the document source as  $\text{\LaTeX}$  environments. *Simple Components* are  $\text{\LaTeX}$  commands that take one argument while Group Components are  $\text{\LaTeX}$  environments that hold the Commands for its constituent Components:

```
\begin{<Container>}[<options>]
  \<Component1>{<Content1>}
  \<Component2>{<Content2>}
  \begin{<GroupComponent>}
    \<Component3>{<Content3>}
    \<Component4>{<Content4>}
  \end{<GroupComponent>}
\end{<Container>}
```

The basic idea is that the Content in the Argument of the Component commands within a Container are collected, processed and the output is printed at the end of the corresponding Container environment. Containers allow Components with the same name to be used and processed independently in different Containers.

Components are only allowed within their corresponding Container environments. Outside, Container sensitive Components may have different meaning or even throw an `Undefined control sequence` error.

Components provided by the modules of the `CoCoTeX` framework usually start with the `tp1` prefix and the component's name begins with a capital letter. For instance,

```
\begin{tpFigure}
  \tpCaption{A~nice image.}
\end{tpFigure}
```

utilizes a Component named `tpCaption` with the `Content` *A nice image.* within the container `tpFigure`.

Properties consist of two parts, the property's name and its value. Some Properties provided by the `CoCoTeX` modules may have a fixed set of string values, while others are completely free to be set and used.

In this manual, the properties provided by the various modules are documented in the following way:

`<name> [<default value>] <allowed values>`

A property with the name `<name>` is set by default to `<default value>`. The user may chose to set it to any of the `<allowed values>`.

<code>&lt;any&gt;</code>	the user is completely free to set this property to any value she wants.
<code>&lt;dimen&gt;</code>	It is expected the property to be a dimension. This may be a length or dimension register, a fix value-unit pair that is understood by <code>TeX</code> , or a macro that expands to a dimension/length.
<code>&lt;num&gt;</code>	It is expected the property to be a numeric value. This may be a counter register, a fix value, or a macro that expands to a number.

`<allowed values>` without angles mean that those are fixes strings that have a special meaning. Those are explained in the descriptions below the property header.

The data type `<empty>` is used to indicate that the property is un-set or empty. This is the default for some of the properties provided by the `CoCoTeX` modules, but basicly all properties can be set to `<empty>`.

## 1.3 Overview: Modules

The following modules are included in `CoCoTeX`:

### 1.3.1 User-Level Modules

`coco-headings.sty` The `headings` module provides a new way to declare and use chapter, section and paragraph titles. It is described in greater detail in `?? ??`.

`coco-floats.sty` The `floats` module provides some extended handling for floating objects like tables or figures. It is described in greater detail in `?? ??`.

<sup>1</sup>That the prefix is `tp` has historic reasons: The earliest version of `CoCoTeX` was called `transpect-tex` after the XML conversion tool `transpect`, which constitutes the first major component of `xerif` (with `CoCoTeX` being the second).

<code>coco-title.sty</code>	The <code>title</code> module provides meta data handlers for title pages. It is described in greater detail in ?? ??.
<code>coco-frame.sty</code>	
<code>coco-notes.sty</code>	The <code>notes</code> module handles the easy switching between footnotes and endnotes, as well as the position where and in what way endnotes are printed. It is described in greater detail in ?? ??.

### 1.3.2 Backend Modules

<code>coco-kernel.sty</code>	The <code>kernel</code> module is the heart of the CoCoTeX framework. As such, it is a hard dependency for all other modules and loaded automatically.
<code>coco-common.sty</code>	The <code>common</code> module is a collection helper macros and functions, that are not per-se part of the CoCoTeX Framework, but utilised by multiple other modules. The common module is loaded automatically by some of the other modules, but not by all.
<code>coco-meta.sty</code>	The <code>meta</code> module collects methods and concepts that are used by both the <code>title</code> and <code>headings</code> modules. It is therefore auto-loaded by both modules.

# Index

*Italic index terms* refer to general design concepts. Page entries in **bold face** refer to main sections that describe the index term. Entries in **mono-type** refer to L<sup>A</sup>T<sub>E</sub>X commands and markup. The symbol ↑ stands for see; the symbol ↓ for see also.

## C

coco-common.sty .....	6
coco-floats.sty .....	5
coco-frame.sty .....	6
coco-headings.sty .....	5
coco-kernel.sty .....	6
coco-notes.sty .....	6
coco-title.sty .....	6
Component .....	3
Container .....	3
Counted Component .....	4

## G

Group Container .....	4
-----------------------	---

## I

Inheritance .....	3
-------------------	---

## P

Properties .....	4
------------------	---

## T

Type .....	4
------------	---

Index of Properties



## Index of $\LaTeX$ macros and environments

Since most  $\LaTeX$  macros in the  $\text{CoCoTeX}$  framework share the  $\text{tp}$  prefix, this index is sorted ignoring this prefix. Index terms with an asterisk (\*) represent  $\LaTeX$  environments.