CoCoTEX

A Commmon LATEX Stylesheet Framework

Version 0.4.0

(2024/01/29)

Contents

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 the core modules that follow the naming convention `coco-<module>.sty`. These modules will be explained in greater detail below in Sect. 1.3, "Overview: Modules".

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

```
pdflatex cocotex.dtx
```

Note 1: You *must* use `pdflatex` in order to create the source code documentation!

Note 2: The source code documentation is a technical breakdown of the framework's source code; it is not the same document as the more end-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 LaTeX Document Class file `cocotex.cls` which can be used to load the whole framework at once.

1.1.2.1 Using cocotex.cls

The `cocotex.cls` serves as stand-in for the LaTeX default document classes `article` and `book`. It is called with the usual LaTeX commmand:

```
\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:

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

1.1.2.2 Using Single Modules

CoCoTEX is modular. That means you can use selected modules as packages together with LATEX'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 first and foremost a **programming framework** designed for LATEX **developers**, who need to build and maintain a number of (not too) different publisher-specific style sheets and partly or fully automatted typesetting processes. CoCoTEX has been developed to serve as a the **rendering backend** for the typesetting tool ***xerif*** [1], but it is also usable as a standalone extension to plain LATEX 2_ε projects.

The following **features** were the initial 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 few or single authors.
- Handling of **recurring complex typographic elements** that are difficult to set-up using standard-LATEX, 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**;
  - 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 LATEX programming that are extensivley influenced by object-oriented design principles. The name CoCoTEX is derived from two of those concepts, namely **Co**ntainers and **Co**mponents. In the next sections, those and other concepts are exlained 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.

1.2.1.1 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

[1] see https://www.le-tex.de/en/xerif.html

constituents, like Title or Number, are shared among all those derivations of the concept “heading”, while others might not. E.g. “Author” is usualy 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**.

1.2.1.2 Components

The inter-connected pieces of information that constitute a Container are called **Components** in the CoCoTEX framework. Most basic components are simple 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 mutliple Authors that contibuted 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.

1.2.1.3 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 ultimatly rendered is controlled by so-called **Properties**. Properties are mostly short pieces of 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.

1.2.1.4 Types, Scope and Modular Inheritance

Properties and Components can be seen as Containerspecific Data **Type**s. 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 LATEX environments. *Simple Components* are LATEX commands that take one argument while Group Components are LATEX environments that hold the Commands for its consitutent 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 `tp`[2] 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>.

<any>	the user is completely free to set this property to any value she wants.
<dimen>	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 TEX, or a macro that expands to a dimension/length.
<num>	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:

[2]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).

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 **??**, "**??**".

`coco-title.sty` The `title` module provides meta data handlers for title pages. It is described in greater detail in **??**, "**??**".

`coco-frame.sty`

`coco-notes.sty` The `notes` 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

`coco-kernel.sty` The `kernel` module is the heart of the CoCoTEX framwork. As such, it is a hard dependency for all other modules and loaded automatically.

`coco-common.sty` The `common` 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.

`coco-meta.sty` The `meta` module collects methods and concepts that are used by both the `title` and `headings` modules. It is therefore auto-loaded by both modules.

1.3.3 Experimental Modules

WARNING!
The modules in this section are to be considered highly experimental!
Use with caution!

`coco-script.sty` This module provides support for non-latin scripts utilizing Google's Noto fonts. It is described in greater detail in **??**, "**??**".

`coco-lists.sty` This module provides support for lists of various types

Part I
CoCoTEX Core Modules

Part II
Customization