LEBHART, WRITE YOUR ARTICLES IN A COLORFUL WAY

Jinwen XU
ProjLib@outlook.com

July 2021, Beijing

Abstract

lebhart is a member of the colorist class series. Its name is taken from German word "lebhaft" ("vividly"), combined with the first three letters of "artikel" ("article"). The entire collection includes colorart and lebhart for typesetting articles and colorbook and beaulivre for typesetting books. My original intention in designing this series was to write drafts and notes that look colorful yet not dazzling.

lebhart has multi-language support, including Chinese (simplified and traditional), English, French, German, Italian, Japanese, Portuguese (European and Brazilian), Russian and Spanish. These languages can be switched seamlessly in a single document. Due to the usage of custom fonts, lebhart requires XeLaTeX or LuaLaTeX to compile.

This documentation is typeset using lebhart (with the option `allowbf`). You can think of it as a short introduction and demonstration.

Contents

Before you start . . . 1
1 Usage and examples . . . 2
1.1 How to load it . . . 2
1.2 Example - A complete document . . . 2
2 On the default fonts . . . 5
3 The options . . . 5
4 Instructions by topic . . . 6
4.1 Language configuration . . . 6
4.2 Theorems and how to reference them . . . 6
4.3 Define a new theorem-like environment . . . 7
4.4 Draft mark . . . 8
4.5 Title, abstract and keywords . . . 9
5 Known issues . . . 10

Before you start

In order to use the package or classes described here, you need to:

- install TeX Live or MikTeX of the latest possible version, and make sure that `colorist` and `projlib` are correctly installed in your TeX system.
- be familiar with the basic usage of LaTeX, and knows how to compile your document with pdfLaTeX, XeLaTeX or LuaLaTeX.

Corresponding to: lebhart 2021/07/15

1 Usage and examples

1.1 How to load it

One only needs to put

```
\documentclass{lebhart}
```

as the first line to use the lebhart class. Please note that you need to use either XeLaTeX or LuaLaTeX engine to compile.

1.2 Example - A complete document

Let's first look at a complete document.

```
1  \documentclass{lebhart}
2  \usepackage{ProjLib}
3
4  \UseLanguage{French}
5
6  \begin{document}
7
8  \title{⟨title⟩}
9  \author{⟨author⟩}
10 \date{\PLdate{2022-04-01}}
11
12 \maketitle
13
14 \begin{abstract}
15     Ceci est un résumé. \dnf<Plus de contenu est nécessaire.>
16 \end{abstract}
17 \begin{keyword}
18     AAA, BBB, CCC, DDD, EEE
19 \end{keyword}
20
21 \section{Un théorème}
22
23 \begin{theorem}\label{thm:abc}
24     Ceci est un théorème.
25 \end{theorem}
26 Référence du théorème: \cref{thm:abc}
27
28 \end{document}
```

If you find this example a little complicated, don't worry. Let's now look at this example piece by piece.

1.2.1 Initialization

```
\documentclass{lebhart}
\usepackage{ProjLib}
```

Initialization is straightforward. The first line loads the document class lebhart, and the second line loads the ProjLib toolkit to obtain some additional functionalities.

1.2.2 Set the language

```
\UseLanguage{French}
```

This line indicates that French will be used in the document (by the way, if only English appears in your article, then there is no need to set the language). You can also switch the language in the same way later in the middle of the text. Supported languages include Simplified Chinese, Traditional Chinese, Japanese, English, French, German, Spanish, Portuguese, Brazilian Portuguese and Russian.

For detailed description of this command and more related commands, please refer to the section on the multi-language support.

1.2.3 Title, author information, abstract and keywords

```
\title{⟨title⟩}
\author{⟨author⟩}
\date{\PLdate{2022-04-01}}
\maketitle

\begin{abstract}
    ⟨abstract⟩
\end{abstract}
\begin{keyword}
    ⟨keywords⟩
\end{keyword}
```

This part begins with the title and author information block. The example shows the basic usage, but in fact, you can also write:

```
\author{⟨author 1⟩}
\address{⟨address 1⟩}
\email{⟨email 1⟩}
\author{⟨author 2⟩}
\address{⟨address 2⟩}
\email{⟨email 2⟩}
...
```

In addition, you may also write in the $\mathcal{AMS}$ fashion, i.e.:

```
\title{⟨title⟩}
\author{⟨author 1⟩}
\address{⟨address 1⟩}
\email{⟨email 1⟩}
\author{⟨author 2⟩}
\address{⟨address 2⟩}
\email{⟨email 2⟩}
\date{\PLdate{2022-04-01}}
```

```
\subjclass{*****}
\keywords{⟨keywords⟩}

\begin{abstract}
    ⟨abstract⟩
\end{abstract}

\maketitle
```

1.2.4 Draft marks

```
\dnf<⟨some hint⟩>
```

When you have some places that have not yet been finished yet, you can mark them with this command, which is especially useful during the draft stage.

1.2.5 Theorem-like environments

```
\begin{theorem}\label{thm:abc}
    Ceci est un théorème.
\end{theorem}
Référence du théorème: \cref{thm:abc}
```

Commonly used theorem-like environments have been pre-defined. Also, when referencing a theorem-like environment, it is recommended to use `\cref{⟨label⟩}` — in this way, there is no need to explicitly write down the name of the corresponding environment every time.

Tip

If you wish to switch to the standard class later, just replace the first two lines with:

```
\documentclass{article}
\usepackage[a4paper,margin=1in]{geometry}
\usepackage[hidelinks]{hyperref}
\usepackage[palatino,amsfashion]{ProjLib}
```

or to use the $\mathcal{AMS}$ class:

```
\documentclass{amsart}
\usepackage[a4paper,margin=1in]{geometry}
\usepackage[hidelinks]{hyperref}
\usepackage[palatino]{ProjLib}
```

Tip

If you like the current document class, but want a more "plain" style, then you can use the option `classical`, like this:

```
\documentclass[classical]{lebhart}
```

2 On the default fonts

By default, lebhart uses Palatino Linotype as the English font, FounderType's YouSong and YouHei GBK as the Chinese fonts[1], and partially uses Neo Euler as the math font. Among them, Neo Euler can be downloaded at `https://github.com/khaledhosny/euler-otf`. The other fonts are not free, you need to purchase and install them on your own.

When the corresponding font is not installed, fonts that comes with TeX Live will be used instead. In this case, the experience might be reduced.

3 The options

lebhart offers the following options:

- The language options `EN` / `english` / `English`, `FR` / `french` / `French`, etc.
  - For the option names of a specific language, please refer to ⟨*language name*⟩ in the next section. The first specified language will be used as the default language.
  - The language options are optional, mainly for increasing the compilation speed. Without them the result would be the same, only slower.
- `draft` or `fast`
  - The option `fast` enables a faster but slightly rougher style, main differences are:
    - Use simpler math font configuration;
    - Do not use hyperref;
    - Enable the fast mode of ProjLib toolkit.

> **Tip**
>
> During the draft stage, it is recommended to use the `fast` option to speed up compilation. When in `fast` mode, there will be a watermark "DRAFT" to indicate that you are currently in the draft mode.

- `a4paper` or `b5paper`
  - Paper size options. The default paper size is 8.5in × 11in.
- `palatino`, `times`, `garamond`, `biolinum` | `useosf`
  - Font options. As the name suggest, font with corresponding name will be loaded.
  - The `useosf` option is used to enable the old-style figures.
- `allowbf`
  - Allow boldface. When this option is enabled, the main title, the titles of all levels and the names of theorem-like environments will be bolded.
- `runin`
  - Use the "runin" style for `\subsubsection`
- `puretext` or `nothms`
  - Pure text mode. Does not load theorem-like environments.
- `nothmnum`
  - Theorem-like environments will not be numbered.

[1] For detail, pleaser visit FounderType's website: `https://www.foundertype.com`.

4 Instructions by topic

4.1 Language configuration

lebhart has multi-language support, including Chinese (simplified and traditional), English, French, German, Italian, Japanese, Portuguese (European and Brazilian), Russian and Spanish. The language can be selected by the following macros:

- `\UseLanguage{`⟨*language name*⟩`}` is used to specify the language. The corresponding setting of the language will be applied after it. It can be used either in the preamble or in the main body. When no language is specified, "English" is selected by default.
- `\UseOtherLanguage{`⟨*language name*⟩`}{`⟨*content*⟩`}`, which uses the specified language settings to typeset ⟨*content*⟩. Compared with `\UseLanguage`, it will not modify the line spacing, so line spacing would remain stable when CJK and Western texts are mixed.

⟨*language name*⟩ can be (it is not case sensitive, for example, `French` and `french` have the same effect):

- Simplified Chinese: `CN`, `Chinese`, `SChinese` or `SimplifiedChinese`
- Traditional Chinese: `TC`, `TChinese` or `TraditionalChinese`
- English: `EN` or `English`
- French: `FR` or `French`
- German: `DE`, `German` or `ngerman`
- Italian: `IT` or `Italian`
- Portuguese: `PT` or `Portuguese`
- Portuguese (Brazilian): `BR` or `Brazilian`
- Spanish: `ES` or `Spanish`
- Japanese: `JP` or `Japanese`
- Russian: `RU` or `Russian`

In addition, you can also add new settings to selected language:

- `\AddLanguageSetting{`⟨*settings*⟩`}`
  - Add ⟨*settings*⟩ to all supported languages.
- `\AddLanguageSetting(`⟨*language name*⟩`){`⟨*settings*⟩`}`
  - Add ⟨*settings*⟩ to the selected language ⟨*language name*⟩.

For example, `\AddLanguageSetting(German){\color{orange}}` can make all German text displayed in orange (of course, one then need to add `\AddLanguageSetting{\color{black}}` in order to correct the color of the text in other languages).

4.2 Theorems and how to reference them

Environments such as `definition` and `theorem` have been preset and can be used directly.

More specifically, preset environments include: `assumption`, `axiom`, `conjecture`, `convention`, `corollary`, `definition`, `definition-proposition`, `definition-theorem`, `example`, `exercise`, `fact`, `hypothesis`, `lemma`, `notation`, `observation`, `problem`, `property`, `proposition`, `question`, `remark`, `theorem`, and the corresponding unnumbered version with an asterisk `*` in the name. The titles will change with the current language. For example, `theorem` will be displayed as "Theorem" in English mode and "Théorème" in French mode.

When referencing a theorem-like environment, it is recommended to use `\cref{`⟨*label*⟩`}`. In this way, there is no need to explicitly write down the name of the corresponding environment every time.

EXAMPLE

```
\begin{definition}[Strange things] \label{def: strange} ...
```

will produce

DEFINITION 4.1 (Strange things) This is the definition of some strange objects. There is approximately an one-line space before and after the theorem environment, and there will be a symbol to mark the end of the environment.

`\cref{def: strange}` will be displayed as: **DEFINITION 4.1**.
After using `\UseLanguage{French}`, a theorem will be displayed as:

THÉORÈME 4.2 (Inutile) Un théorème en français.

By default, when referenced, the name of the theorem always matches the language of the context in which the theorem is located. For example, the definition above is still displayed in English in the current French mode : **DEFINITION 4.1** and **THÉORÈME 4.2**. If you want the name of the theorem to match the current context when referencing, you can add `regionalref` to the global options.

The following are the main styles of theorem-like environments:

THEOREM 4.3 Theorem style: theorem, proposition, lemma, corollary, ...

Proof | Proof style

Remark style

CONJECTURE 4.4 Conjecture style

EXAMPLE Example style: example, fact, ...

PROBLEM 4.5 Problem style: problem, question, ...

For aesthetics, adjacent definitions will be connected together automatically:

DEFINITION 4.6 First definition.

DEFINITION 4.7 Second definition.

4.3 DEFINE A NEW THEOREM-LIKE ENVIRONMENT

If you need to define a new theorem-like environment, you must first define the name of the environment in the language to use:

- `\NameTheorem[`⟨*language name*⟩`]{`⟨*name of environment*⟩`}{`⟨*name string*⟩`}`

For ⟨*language name*⟩, please refer to the section on language configuration. When ⟨*language name*⟩ is not specified, the name will be set for all supported languages. In addition, environments with or without asterisk share the same name, therefore, `\NameTheorem{envname*}{...}` has the same effect as `\NameTheorem{envname}{...}`.

And then define this environment in one of following five ways:

- `\CreateTheorem*{`⟨*name of environment*⟩`}`
  - Define an unnumbered environment ⟨*name of environment*⟩
- `\CreateTheorem{`⟨*name of environment*⟩`}`
  - Define a numbered environment ⟨*name of environment*⟩, numbered in order 1,2,3,…
- `\CreateTheorem{`⟨*name of environment*⟩`}[`⟨*numbered like*⟩`]`
  - Define a numbered environment ⟨*name of environment*⟩, which shares the counter ⟨*numbered like*⟩
- `\CreateTheorem{`⟨*name of environment*⟩`}<`⟨*numbered within*⟩`>`
  - Define a numbered environment ⟨*name of environment*⟩, numbered within the counter ⟨*numbered within*⟩
- `\CreateTheorem{`⟨*name of environment*⟩`}(`⟨*existed environment*⟩`)`
  `\CreateTheorem*{`⟨*name of environment*⟩`}(`⟨*existed environment*⟩`)`
  - Identify ⟨*name of environment*⟩ with ⟨*existed environment*⟩ or ⟨*existed environment*⟩`*`.
  - This method is usually useful in the following two situations:
    1. To use a more concise name. For example, with `\CreateTheorem{thm}(theorem)`, one can then use the name `thm` to write theorem.
    2. To remove the numbering of some environments. For example, one can remove the numbering of the `remark` environment with `\CreateTheorem{remark}(remark*)`.

Tip

This macro utilizes the feature of amsthm internally, so the traditional `theoremstyle` is also applicable to it. One only needs declare the style before the relevant definitions.

Here is an example. The following code:

```
\NameTheorem[EN]{proofidea}{Idea}
\CreateTheorem*{proofidea*}
\CreateTheorem{proofidea}<subsection>
```

defines an unnumbered environment `proofidea*` and a numbered environment `proofidea` (numbered within subsection) respectively. They can be used in English context. The effect is as follows:

Idea | The `proofidea*` environment.

Idea 4.3.1 | The `proofidea` environment.

4.4 Draft mark

You can use `\dnf` to mark the unfinished part. For example:

- `\dnf` or `\dnf<...>`. The effect is: `To be finished #1` or `To be finished #2: ...`.
  The prompt text changes according to the current language. For example, it will be displayed as `Pas encore fini #3` in French mode.

Similarly, there is `\needgraph`:

- `\needgraph` or `\needgraph<...>`. The effect is:

```
A graph is needed here #1
```

or

```
A graph is needed here #2: ...
```

The prompt text changes according to the current language. For example, in French mode, it will be displayed as

```
Il manque une image ici #3
```

4.5 Title, abstract and keywords

lebhart has both the features of standard classes and that of the $\mathcal{AMS}$ classes.

Therefore, the title part can either be written in the usual way, in accordance with the standard class article:

```
\title{⟨title⟩}
\author{⟨author⟩\thanks{⟨text⟩}}
\date{⟨date⟩}
\maketitle
\begin{abstract}
    ⟨abstract⟩
\end{abstract}
\begin{keyword}
    ⟨keywords⟩
\end{keyword}
```

or written in the way of $\mathcal{AMS}$ classes:

```
\title{⟨title⟩}
\author{⟨author⟩}
\thanks{⟨text⟩}
\address{⟨address⟩}
\email{⟨email⟩}
\date{⟨date⟩}
\keywords{⟨keywords⟩}
\subjclass{⟨subjclass⟩}
\begin{abstract}
    ⟨abstract⟩
\end{abstract}
\maketitle
```

The author information can contain multiple groups, written as:

```
\author{⟨author 1⟩}
\address{⟨address 1⟩}
\email{⟨email 1⟩}
\author{⟨author 2⟩}
\address{⟨address 2⟩}
\email{⟨email 2⟩}
...
```

Among them, the mutual order of \address, \curraddr, \email is not important.

5 Known issues

- The font settings are still not perfect.
- Since many features are based on the ProjLib toolkit, colorist (and hence colorart, lebhart and colorbook, beaulivre) inherits all its problems. For details, please refer to the "Known Issues" section of the ProjLib documentation.
- The error handling mechanism is incomplete: there is no corresponding error prompt when some problems occur.
- There are still many things that can be optimized in the code.