ProjLib Toolkit

User Manual

Jinwen XU
ProjLib@outlook.com

August 2021, Beijing

Abstract

The ProjLib toolkit is designed to simplify the preparation before writing LaTeX documents. With the package ProjLib loaded, you no longer need to set up the theorem-like environments nor configure the appropriate multilingual settings. Additionally, a series of auxiliary functionalities are introduced.

Contents

Before you start

In order to use the toolkit, you need to:

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

1 Introduction

The name ProjLib can be regarded as the abbreviation of *Project Library* in English or *Projet Libre* in French (the author prefers the French interpretation). Its main purpose is to provide multi-language support and theorem-like environments with clever references. In addition, some additional features are provided, such as the enhanced author block, draft marks, mathematical symbols and shortcuts, etc.

The ProjLib toolkit is composed of the main package ProjLib and a series of components whose names begin with the abbreviation ”PJL”. You can learn how to use it through the usage examples in the next section.

Corresponding to: ProjLib 2021/08/19

2 Usage example

2.1 How to load it

Just add the following line to your preamble:

```
\usepackage{ProjLib}
```

Attention

Since cleveref is used internally, ProjLib needs to be placed after varioref and hyperref.

2.2 Example - A complete document

Let's first look at a complete document.

```
\documentclass{article}
\usepackage[a4paper,margin=.75in]{geometry}
\usepackage[hidelinks]{hyperref}
\usepackage[palatino]{ProjLib} % Load the toolkit and use font Palatino

\UseLanguage{French} % Use French from here

\begin{document}

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

\maketitle

\begin{abstract}
    ⟨abstract text⟩ \dnf<⟨some hint⟩>
\end{abstract}

\section{Un théorème}

\begin{theorem}\label{thm:abc}
    Ceci est un théorème.
\end{theorem}

Référence du théorème: \cref{thm:abc} % It is recommended to use clever reference

\end{document}
```

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

2.2.1 Initialization

```
\documentclass{article}
\usepackage[a4paper,margin=.75in]{geometry}
\usepackage[hidelinks]{hyperref}
\usepackage[palatino]{ProjLib}
```

In standard classes, one usually only need to configure the page size, hyperlinks and load ProjLib before actually start writing the document. The font option `palatino` of ProjLib is used here. For all available options of ProjLib, please refer to the next section.

Of course, you can also use the document class amsart, the configurations are the same.

2.2.2 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[1].

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

2.2.3 Title and author information

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

This part is the title and author information block. The example shows the most 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, if the $\mathcal{AMS}$ simulation is enabled, you can also write in the $\mathcal{AMS}$ fashion (the original way still works). For this, you should add the package option `amsfashion`[2], *i.e.*, the line that introduces ProjLib should be written as:

```
\usepackage[amsfashion,palatino]{ProjLib}
```

And correspondingly, you will also be able to use these macros:

```
\dedicatory{⟨dedicatory⟩}
```

[1]However, you need to add the encoding support and fonts of the corresponding language by yourself. For example, for Chinese, you may need to load the ctex package and set the fonts. As a sidenote, you can try the author's document classes einfart or lebhart, in which the corresponding settings have been completed. For the details, run `texdoc minimalist` or `texdoc colorist` in the command line.

[2]Since this option modifies some internal macros of LaTeX, it may conflict with some packages or document classes, and thus it is not enabled by default.

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

In addition, you can also write the abstract before `\maketitle`, as the way required in the $\mathcal{AMS}$ classes:

```
\begin{abstract}
    ⟨abstract text⟩
\end{abstract}
\maketitle
```

2.2.4 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.

2.2.5 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.

3 Options of the main package

ProjLib offers the following options:

- `draft` or `fast`
  - Fast mode. The functionality will be appropriately reduced to get faster compilation speed, recommended to use during the writing stage.
- `palatino`, `times`, `garamond`, `noto`, `biolinum` | `useosf`
  - Font options. As the names suggest, font with corresponding name will be loaded.
  - The `useosf` option is used to enable the old-style figures.
- `nothms`, `delaythms`, `nothmnum`, `thmnum` or `thmnum=`⟨*counter*⟩, `regionalref`, `originalref`
  - Options from the component PJLthm used for setting theorem-like environments, please refer to the section on this package for details.
- `author`
  - Load the component PJLauthor used to enhance the author information block. For more information about its functionality, see the section on this package.
- `amsfashion`
  - Allow the user to write document in the $\mathcal{AMS}$ fashion. In the mean time, the option `author` will be automatically turned on.

In addition, there are also some options of the components that should be passed as global options of your document class, such as the language options `EN` / `english` / `English`, `FR` / `french` / `French` etc. of PJLlang, and `paperstyle`, `preview` of PJLpaper. For more information, please refer to the corresponding sections.

4 The components

4.1 Main functions

4.1.1 PJLauthor: enhanced author block

PJLauthor offers \address, \curraddr and \email, and allows you to enter multiple groups of author information. The standard usage is like this:

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

The mutual order of \address, \curraddr and \email is not important.

In addition, you can use the option `amsfashion` to enable the $\mathcal{AMS}$ fashion. More specifically, the effect is:

- Provides the macros \dedicatory, \keywords and \subjclass;
- \thanks can be written outside \author;
- The `abstract` environment can be placed before \maketitle.

Attention

These modifications would only take place in standard classes. In the $\mathcal{AMS}$ classes, PJLauthor does not have any effect.

4.1.2 PJLlang: multi-language support

PJLlang offers multi-language support, including simplified Chinese, traditional Chinese, English, French, German, Japanese, and Russian (among them, Chinese, Japanese, and Russian require appropriate TEX engines and fonts to support).

PJLlang provides language options. The names of these options have three types, which are abbreviations (such as `EN`), lowercase (such as `english`), and capital letters (such as `English`). For the option names of a specific language, please refer to ⟨*language name*⟩ below. Among them, the first specified language ⟨*first language*⟩ will be used as the default language, which is equivalent to specifying \UseLanguage{⟨*first language*⟩} at the beginning of your document.

Tip

It is recommended to use these language options and pass them as global options. In this way, only the specified languages are set, thus saving the TEX memory and significantly improving the compilation speed.

The language can be selected by the following macros:

- \UseLanguage{⟨*language name*⟩} is used to specify the language. The corresponding settings 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).

4.1.3 PJLthm: theorem-like environments with clever reference and multilingual support

PJLthm offers the configuration of theorem-like environments. It has the following option:

- `nothms`
  - Theorem-like environments will not be defined. You may use this option if you wish to apply your own theorem styles.
- `delaythms`
  - Defer the definition of theorem-like environments to the end of the preamble. Use this option if you want the theorem-like environments to be numbered within a custom counter.
- `nothmnum`, `thmnum` or `thmnum=`⟨*counter*⟩
  - Theorem-like environments will not be numbered / numbered in order 1, 2, 3... / numbered within ⟨*counter*⟩. Here, ⟨*counter*⟩ should be a built-in counter (such as `subsection`) or a custom counter defined in the preamble (with the option `delaythms` enabled). If no option is used, they will be numbered within `chapter` (book) or `section` (article).
- `regionalref`, `originalref`
  - When referencing, whether the name of the theorem-like environment changes with the current language. The default is `regionalref`, *i.e.*, the name corresponding to the current language is used; for example, when referencing a theorem-like environment in English context, the names ”Theorem, Definition...” will be used no matter which language context the original environment is in. If `originalref` is enabled, then the name will always remain the same as the original place; for example, when referencing a theorem written in the French context, even if one is currently in the English context, it will still be displayed as “Théorème”.
  - In `fast` mode, the option `originalref` will have no effect.

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. For details on how to select a language, please refer to the section on PJLlang.

Tip

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.

If you need to define a new theorem-like environment, you must first define the name of the environment in the language to use. There are two ways for this:

- Simple settings: `\NameTheorem[⟨language name⟩]{⟨name of environment⟩}{⟨name string⟩}`
  - This approach only sets one main name, the other names, such as those used for clever reference, are set to be the same (in particular, for clever reference, the singular and plural form will not be distinguished). 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}` .
- Detailed settings (recommended):

```
\NameTheorem{⟨name of environment⟩}{
    ⟨language name 1⟩={
        name=⟨Name⟩,
        crefname={⟨name⟩}{⟨names⟩},
        Crefname={⟨Name⟩}{⟨Names⟩},
        autorefname=⟨name⟩,
        theoremheading=⟨Name⟩,
    },
    ⟨language name 2⟩={...},
}
```

or

```
\NameTheorem[⟨language name⟩]{⟨name of environment⟩}{
    name=⟨Name⟩,
    crefname={⟨name⟩}{⟨names⟩},
    Crefname={⟨Name⟩}{⟨Names⟩},
    autorefname=⟨name⟩,
    theoremheading=⟨Name⟩,
}
```

  - This approach sets all the names. When *⟨language name⟩* is not specified, the full interface will be enabled; when it is specified, only the names of the corresponding language are set. Similarly, environments with or without asterisk share the same name, therefore, `\NameTheorem{envname*}` has the same effect as `\NameTheorem{envname}` .

Tip

In addition, you can also name a theorem-like environment while defining it, see the description of `\CreateTheorem` later.

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.

You can also name a theorem-like environment while defining it, by adding afterwards a group of parentheses containing the settings:

```
\CreateTheorem{⟨name of environment⟩}{
    ⟨language name 1⟩={
        name=⟨Name⟩,
        crefname={⟨name⟩}{⟨names⟩},
        Crefname={⟨Name⟩}{⟨Names⟩},
        autorefname=⟨name⟩,
        theoremheading=⟨Name⟩,
    },
    ⟨language name 2⟩={...},
}
```

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 (the actual style is related to the document class):

Idea | The `proofidea*` environment. ☐

Idea 4.1.1 | The `proofidea` environment. ☐

Of course, you can also use a set of more detailed name:

```
\NameTheorem{proofidea}{
    EN = {
        name = Idea,
        crefname = {idea}{ideas},
        Crefname = {Idea}{Ideas},
    }
}
\CreateTheorem*{proofidea*}
\CreateTheorem{proofidea}<subsection>
```

or set the names while defining them (for `proofidea*` and `proofidea`, set once suffices):

```
\CreateTheorem*{proofidea*}
\CreateTheorem{proofidea}<subsection>{
    EN = {
        name = Idea,
        crefname = {idea}{ideas},
        Crefname = {Idea}{Ideas},
    }
}
```

4.2 Secondary functions

4.2.1 PJLdate: date-time processing

PJLdate offers the `\PLdate`⟨*yyyy-mm-dd*⟩ (or `\PJLdate`⟨*yyyy-mm-dd*⟩) macro to convert ⟨*yyyy-mm-dd*⟩ into the date format of the currently selected language. For example, in current English context, `\PLdate {2022-04-01}` would become "April 1, 2022", while in French context as "1$^{\text{er}}$ avril 2022".

For details on how to select a language, please refer to the section on PJLlang.

4.2.2 PJLdraft: draft marks

PJLdraft offers the following macros:

- `\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.
- `\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`

For details on how to select a language, please refer to the section on PJLlang.

4.2.3 PJLlogo: the logo ProjLib

PJLlogo offers the macro `\ProjLib` to draw the logo, which looks like ProjLib. It is similar to ordinary text macros and can be used with different font size macros:

`\tiny`:	ProjLib
`\scriptsize`:	ProjLib
`\footnotesize`:	ProjLib
`\normalsize`:	ProjLib
`\large`:	ProjLib
`\Large`:	ProjLib
`\LARGE`:	ProjLib
`\huge`:	ProjLib
`\Huge`:	ProjLib

4.2.4 PJLmath: math symbols and shortcuts

PJLmath offers the following shortcuts:

i) `\mathfrak{·}` ⟶ `\mf·` or `\frak·` . For example, `\mfA` (or `\mf{A}`) has the same effect as `\mathfrak{A}`. This works for both upper and lower case, producing:

$$\mathfrak{abcdefghijklmnopqrstuvwxyz}$$
$$\mathfrak{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

ii) `\mathbb{·}` $\longrightarrow$ `\bb·` . This only works for uppercase alphabet and the number 1.

$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ1}$$

There are also special command for well-known algebraic structures: `\N`, `\Z`, `\Q`, `\R`, `\C`, `\F`, `\A`.

$$\mathbb{NZQRCFA}$$

iii) `\mathcal{·}` $\longrightarrow$ `\mc·` or `\cal·` . This only works for uppercase alphabet.

$$\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

iv) `\mathscr{·}` $\longrightarrow$ `\ms·` or `\scr·` . This only works for uppercase alphabet.

$$\mathscr{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

In addition, PJLmath also provides some math symbols that are not by default included in LaTeX.

`\abs`	`\abs{a}` $\to$ $\lvert a\rvert$	absolute value symbol
`\norm`	`\norm{a}` $\to$ $\lVert a\rVert$	norm symbol
`\injection`	`\injection` $\to$ $\hookrightarrow$	arrow symbol for injection
`\surjection`	`\surjection` $\to$ $\twoheadrightarrow$	arrow symbol for surjection
`\bijection`	`\bijection` $\to$ $\xrightarrow{\sim}$	arrow symbol for bijection

These shortcuts and symbols are defined in such a way that they will not conflict with existing or user-defined commands. Thus, even if you do not use these shortcuts or symbols, there is no need to worry that their existence will bring errors.

4.2.5 PJLpaper: paper configuration

PJLpaper is mainly used to adjust the paper color. It has the following options:

- `paperstyle = ⟨`*paper style name*`⟩`
  - Set the paper color style. The options available for ⟨*paper style name*⟩ are: `yellow`, `dark` and `nord`.
- `yellowpaper`, `darkpaper`, `nordpaper`
  - Same as `paperstyle` with the corresponding ⟨*paper style name*⟩ specified.
- `preview`
  - Preview mode. Crop the white edges of pdf file for the convenience of reading.

It is recommended to use them as global options of the document class. In this way, the paper settings would be clear at a glance.

5 Known issues

- PJLauthor is still in its preliminary stage, its effect is not as good as the relatively mature authblk.
- PJLlang: It is still quite problematic with the configuration of polyglossia, so main features are implemented through babel for now.
- PJLpaper: the `preview` option is mainly implemented with the help of package geometry, so it does not work quite as well in the KOMA document classes.
- PJLthm: The numbering and theorem-style settings of the theorem-like environments cannot be accessed by the user at present.
- PJLthm: The localization of cleveref is not yet complete for all supported languages of PJLlang, especially for Chinese, Japanese and Russian.
- Error handling mechanism is incomplete: no corresponding error prompt when some problems occur.
- There are still many things that can be optimized in the code. Some takes too long to run, especially the setup of theorem-like environments in PJLthm.