The packdoc package

v0.1

Jander Moreira
moreira.jander@gmail.com
https://github.com/jandermoreira/packdoc

2025/01/26

Contents

Change History

0.1 (2024/12/04)
Initial version.

1 Introduction

The packdoc package was developed to assist in the writing of documents or manuals that use LaTeX. The intention is to simplify the documentation process by providing standardized formatting for key components. This includes the ability to describe the syntax and

functionality of macros and environments, as well as available options and their usage. Additionally, the package facilitates the inclusion of elements such as index entries, ensuring a clear and consistent structure throughout the text.

This package is not intended to replace or compete with the use of .dtx and .ins files, which is an important strategy for writing .sty files. The development of a package and its documentation should be done independently. A very personal reason for this approach is that my background is in Computer Science, and, as such, I have a strong inclination towards writing well-formatted and well-documented code. Therefore, in the end, while the style file created from .dtx and .ins files is functional, it often lacks attention to the elegance in the look of the code. For this reason, I write .sty files and their documentation completely separately.

This package offers the capability to document commands and environments, in addition to supporting versioning through a change log.

2 Package usage and options

To use this package, it must be loaded with `\usepackage`.

`\usepackage[`⟨*options*⟩`]{packdoc}`

So far there is only one option to the package: `presets`. This is covered in Section 3.4.

3 Documentation

3.1 Basic commands

A set of useful macros is provided to facilitate the creation of consistent documents and ensure uniform formatting across the text.

`\PackageName[`⟨*options*⟩`]{`⟨*name*⟩`}`

This macro is designed to format ⟨*name*⟩ as the name of LaTeXpackages and classes, such as amsmath or article. The optional argument ⟨*options*⟩ allows for local adjustments to the style used for package names by modifying the `package style`.

```
Examples of useful packages include \PackageName{graphicx} and \PackageName{xcolor}.
↪ Additionally, \PackageName[package style=\scshape\color{blue}]{babel} and
↪ \PackageName{inputenc} are also important.
```

Examples of useful packages include graphicx and xcolor. Additionally, BABEL and inputenc are also important.

`\Argument[`⟨*options*⟩`]{`⟨*name*⟩`}`

The `\Argument` macro is used to format generic arguments.
The optional `argument color` allows the color of the argument to be customized. However, the font, size, and shape of arguments are currently hardcoded and cannot be adjusted.

```
The \PackageName{article} class supports \Argument{options}, including settings like
↪ paper size and the number of columns.
```

The article class supports ⟨*options*⟩, including settings like paper size and the number of columns.

`\MArg[`⟨*options*⟩`]{`⟨*name*⟩`}`

"MArg" means *mandatory argument*, and the result is the same as `\Argument` enclosed in braces. The same ⟨*options*⟩ available for `\Argument` also apply.

`\OArg[`⟨*options*⟩`]{`⟨*name*⟩`}`

"OArg" stands for *optional argument*, and the result is the same as `\Argument` enclosed in square brackets. The same ⟨*options*⟩ available for `\Argument` also apply.

```
Mandatory argument: \MArg{arg}.\par
Optional argument: \OArg{arg}.
```

Mandatory argument: {⟨*arg*⟩}.
Optional argument: [⟨*arg*⟩].

Additionally, macros for arguments between angle brackets (e.g., `<color = blue>`) and plain text (e.g., `{newcounter}`) are also available.

`\AArg[`⟨*options*⟩`]{`⟨*name*⟩`}`

"AArg" stands for *optional argument between angular brackets*. The same ⟨*options*⟩ for `\Argument` also apply.

```
Class \PackageName{beamer} can use overlays in slides. For example, \AArg[argument
↪ color = red]{range} can be used in a itemized list and \Argument{range} can be
↪ set to \PDInline{2} (only on slide 2) or \PDInline{2-5} (from slide 2 to 5), for
↪ example.
```

Class beamer can use overlays in slides. For example, <⟨*range*⟩> can be used in a itemized list and ⟨*range*⟩ can be set to 2 (only on slide 2) or 2-5 (from slide 2 to 5), for example.

`\PArg{`⟨*name*⟩`}`

"PArg" stands for *mandatory plain text argument* and is an `\Argument` between brackets without any special format.

```
Plain argument: \PArg{article}
```

Plain argument: {article}

3.2 Elements

An *element* in the scope of this document refers to an item that can be highlighted and referenced, such as macros, options and environments, for example.

To instance an element, the `\PDNewElement` macro must be used.

`\PDNewElement{`⟨*element name*⟩`}{`⟨*element options*⟩`}`

This macro creates a new element named ⟨*element name*⟩ and several other macros to use it. The ⟨*element options*⟩ are a key/value list of options to change how the item will look like.

```
\PDNewElement{EnumItemOption}{color = red!75!black}
```

```
I like to use the \PackageName{enumitem} package. It makes easier to fine tune the
↪ lists appearance, such as using \EnumItemOption{itemsep} or
↪ \EnumItemOption{parsep} to change the spaces between the items.
```

I like to use the enumitem package. It makes easier to fine tune the lists appearance, such as using itemsep or parsep to change the spaces between the items.

When an element is created, several other macros are created for different needs.

Macro	Description	Ex0.1
`\`⟨*element name*⟩	Formats to element style.	`\MyElement{a4paper}`
`\`⟨*element name*⟩`Def`	Formats to element style, sets a label and index the element.	`\MyElementDef{left}`
`\`⟨*element name*⟩`Ref`	Formats to element style and hyperlinks to the definition.	`\MyElementRef{no align}`
`\`⟨*element name*⟩`Ind`	Formats to element style and index the element.	`\MyElementInd{showframe}`
`\`⟨*element name*⟩`RefInd`	Formats to element style, hyperlinks to the definition and add an entry to the index.	`\MyElementRefInd{a4paper}`
`\`⟨*element name*⟩`Index`	Index the element without any typeset.	`\MyElementIndex{element}`

`\`⟨*element name*⟩`[`⟨*options*⟩`]{`⟨*item*⟩`}`

A macro named after ⟨*element name*⟩ is created to typeset ⟨*item*⟩ in a consistent way. The appearance will follow that defined when ⟨*element name*⟩ was created with `\PDNewElement`, but can be overridden with ⟨*options*⟩ (see Section 3.3).

```
I like to use the \PackageName{enumitem} package. It makes easier to fine tune
↪ lists, such as using \EnumItemOption{itemsep} or \EnumItemOption{parsep} to
↪ change the spaces between them items.
```

I like to use the enumitem package. It makes easier to fine tune lists, such as using itemsep or parsep to change the spaces between them items.

\⟨*element name*⟩Def[⟨*options*⟩]{⟨*item*⟩}

A macro \⟨*element name*⟩Def is used to define an ⟨*item*⟩, so it can be cross-referenced and have index entries. The definition can be referenced by the \⟨*element name*⟩Ref macro.
The appearance will follow that defined when ⟨*element name*⟩ was created with \PDNewElement, but can be overridden with ⟨*options*⟩ (see Section 3.3).
The definition of an ⟨*item*⟩ can be stated with an environment called ⟨*element name*⟩def instead.

```
I wrote some code to extend the \PackageName{enumitem} package. Now
↪  \EnumItemOptionDef{float} can be used to insert a list in a float.
% The name 'float' has an anchor (label) and entries in the index.
```

I wrote some code to extend the enumitem package. Now float can be used to insert a list in a float.

\⟨*element name*⟩Ref[⟨*options*⟩]{⟨*text*⟩}

The macro \⟨*element name*⟩Ref typesets the ⟨*item*⟩ and creates a link to its definition.
The appearance will follow that defined when ⟨*element name*⟩ was created with \PDNewElement, but can be overridden with ⟨*options*⟩ (see Section 3.3).

```
Remember that the \EnumItemOptionRef{float} cannot be used if the list is already in
↪  a float.
% 'float' is a link to the definition
```

Remember that the float cannot be used if the list is already in a float.

\⟨*element name*⟩Ind[⟨*options*⟩]{⟨*item*⟩}

The \⟨*element name*⟩ macro defines ⟨*item*⟩ and inserts entries to the index. Sometimes a secondary index entry is desired, so \⟨*element name*⟩Ind does the job. A reference to the definition is not created.
The appearance will follow that defined when ⟨*element name*⟩ was created with \PDNewElement, but can be overridden with ⟨*options*⟩ (see Section 3.3).

```
Here we describe some other important information about the
↪  \EnumItemOptionInd{float} option.
% 'float' has now new index entries, but it's not a link
```

Here we describe some other important information about the float option.

`\⟨element name⟩RefInd[⟨options⟩]{⟨text⟩}`

The `\⟨element name⟩RefInd` performs the job of both `\⟨element name⟩Ind` and `\⟨element name⟩Ind`, so the index is affected and a reference to the definition is created.
The appearance will follow that defined when ⟨*element name*⟩ was created with `\PDNewElement`, but can be overridden with ⟨*options*⟩ (see Section 3.3).

```
Here we describe some other important information about the
↪  \EnumItemOptionRefInd{float} option.
% 'float' has now new index entries and is also a link
```

Here we describe some other important information about the `float` option.

An element can be defined, as previously stated, by calling `\⟨element name⟩Def`. This is handy for inline definitions. An alternative way to define an element is to use an environment also created by `\PDNewElement`. This environment is named ⟨*element name*⟩`def`.

```
\begin{⟨element name⟩def}[⟨options⟩]{⟨item⟩}{⟨arguments⟩}{⟨complement⟩}
   ⟨element description⟩
\end{⟨element name⟩def}
```

This environment uses ⟨*element name*⟩'s styles to define an instance named ⟨*item*⟩, along with its ⟨*arguments*⟩ and a ⟨*complement*⟩. The ⟨*complement*⟩ is any additional text.
The header of the definition will use the following format:

⟨*item*⟩⟨*args prefix*⟩⟨*arguments*⟩⟨*complement prefix*⟩⟨*complement*⟩

The values for ⟨*args prefix*⟩ and ⟨*complement prefix*⟩ are set by `arguments prefix` and `complement prefix` options respectively.
This environment will create an anchor to ⟨*item*⟩ and add it to the index.

```
% args prefix is \texttt{~=~} and complement prefix is \hfill
\begin{EnumItemOptiondef}{float}{\PDInline{true} | \PDInline{false}}{Default:
↪  \PDInline{true}; initially: \PDInline{false}}
    By adding \EnumItemOption{float} to a list, it will be inserted in a float
    ↪  environment.
\end{EnumItemOptiondef}
% This definition can be linked with \<element>Def and item is indexed
```

`float = true | false` Default: `true`; initially: `false`

By adding `float` to a list, it will be inserted in a float environment.

Another environment is available to just typeset an item, without creating an anchor and not adding entries to the index.

```
\begin{⟨element name⟩*}[⟨options⟩]{⟨item⟩}{⟨arguments⟩}{⟨complement⟩}
   ⟨element description⟩
\end{⟨element name⟩*}
```

The ⟨*element name*⟩`env*` environment has the same behavior as ⟨*element name*⟩`env`, but no anchor is created and no entry is added to the index.

```
\begin{EnumItemOption*}{float}{\PDInline{true} | \PDInline{false}}{Default:
↪  \PDInline{true}; initially: \PDInline{false}}
    By adding \EnumItemOption{float} to a list, it will be inserted in a float
    ↪  environment.
\end{EnumItemOption*}
```

`float` = `true` | `false` — Default: `true`; initially: `false`

By adding `float` to a list, it will be inserted in a float environment.

3.3 Options for elements

Several options can be used to customize each element. These options are typically specified when the element is created with `\PDNewElement`, but can also be modified with `\PDSetElement`. Options not specified at creation assume predefined default values, which can also be changed with `\PDSet`.

```
\PDSetElement{⟨element name⟩}{⟨option list⟩}
```

After created with `\PDNewElement`, options can be changed *a posteriori* with `\PDSetElement`

```
\PDNewElement{MyItem}{color = magenta, no single index, no group index}
An example of MyItem is \MyItem{PDExample}.\par
\PDSetElement{MyItem}{color = blue!80!black, font = \slshape}
This is another one: \MyItem{instance}.
```

An example of MyItem is PDExample.
This is another one: *instance*.

`package style` = ⟨*commands*⟩ — Initially: `\sffamily`

Sets how `\PackageName` will typeset classes and package names.

`argument color` = ⟨*color*⟩ — Inititally: `orange!50!black`

Sets the color to typeset arguments (see `\Argument`).

`prefix` = ⟨*text*⟩ — Initially empty

When an element is typeset, ⟨*text*⟩ is added before the item's name. For example, if an element is created for macros, `prefix` can be set to `\textbackslash`.

`arguments prefix` = ⟨*text*⟩ Initially empty

This options sets the text to be put between the item name and its arguments. For macros, for example, it must be empty; for options it can be set to `=`.
This element is only typeset if the ⟨*arguments*⟩ are not empty (meaning anything with width equal to zero).

`complement prefix` = ⟨*text*⟩ Initially: `\hfill`

The contents of ⟨*text*⟩ will be inserted between the ⟨*arguments*⟩ and the ⟨*complement*⟩.
This element is only typeset if the ⟨*complement*⟩ is not empty (meaning anything with width equal to zero).

`font` = ⟨*commands*⟩ Initially: `\ttfamily`

These ⟨*commands*⟩ are prepended to every ⟨*item*⟩.

`color` = ⟨*color*⟩ Initially: `.!75`

This sets the color to be used with the ⟨*item*⟩.

`index heading` = ⟨*text*⟩ Initially: ⟨*element name*⟩

When an item is defined (`\`⟨*element name*⟩`Def` macro or ⟨*element name*⟩`env`), index entries will be grouped under a main entry named ⟨*text*⟩.
Grouped index entries can be disabled with `no group index`.
This option is element-specific and will not work as a global option.

`no group index` = `true` | `false` Default: `true`; initially `true`

This option suppresses adding entries as groups to the index. Single entries are not affected.

`index remark` = ⟨*text*⟩ Default: `{~(`⟨*element name*⟩`)}`

Every index entry will be appended with ⟨*text*⟩ the item name.
Single entries can be removed with `no single index`.
This option is element-specific and will not work as a global option.

`no single index` = `true` | `false` Default: `true`; initially `true`

This option suppresses adding single entries to the index. Group entries are not affected.

3.4 Preset elements

`presets = true | false` Default: `true`; initially: `false`

When packdoc is loaded with the `presets` option, some useful elements are automatically created.

Element name	Description
`Option`	To use with options (as those passed within brackets).
`Macro`	For macros, preceding them with a backslash.
`Environment`	For general environments.

This document used these presets.

```
The preset elements include \Option{option}, \Macro{macro} and
↪ \Environment{environment}.

For example, \OptionRef{presets} is a package option. The \MacroRef{PDNewElement}
↪ macro is used to create new elements and \Environment{tabular} is a well known
↪ environment.
```

The preset elements include `option`, `\macro` and `environment`.
For example, `presets` is a package option. The `\PDNewElement` macro is used to create new elements and `tabular` is a well known environment.

3.5 Supplementary resources

Code examples can be displayed with `PDListing`, while examples along with their corresponding results can be shown using `PDExample`.

```
\begin{PDListing}
    ⟨code⟩
\end{PDListing}
```

This environment is used to display LaTeX code.

```
This is an example code:

\begin{PDListing}
    \usepackage{packdoc}
\end{PDListing}
```

This is an example code:

```
\usepackage{packdoc}
```

```
\begin{PDExample}
   ⟨code⟩
\end{PDExample}
```

This environment is used to present LaTeX code along with its output.

```
This is an example of use:

\begin{PDExample}
    Resources are macros, such as \Macro{Option}, and environments, such as
    ↪  \Environment{PDExample}.
\end{PDExample}
```

This is an example of use:

```
Resources are macros, such as \Macro{Option}, and environments, such as
↪  \Environment{PDExample}.
```

Resources are macros, such as \Option, and environments, such as PDExample.

Inline code can use \PDInline.

```
\PDInline{⟨code⟩}
```

This macro is used to display LaTeX code. If braces are balanced, the use \PDInline{{example}} ({example}) holds; when unbalanced, \PDInline!{example! ({example) can be used. The use is equivalent to \verb.

```
Someone can use \PDInline{\usepackage[presets]{packdoc}} instead of just
↪  \PDInline{\usepackage{packdoc}}.
```

Someone can use \usepackage[presets]{packdoc} instead of just \usepackage{packdoc}.

```
\PDTilde
```

The \PDTilde generates a more visually appealing and accurately positioned single tilde (~) for representing a non-breaking space.
In context, some tildes can be compared:

Code	Result	Result (monotype)
ab\PDTilde cd	ab~cd	ab~cd
ab\~{}cd	ab˜cd	ab˜cd
ab\texttildelowcd (textcomp)	ab∼cd	ab∼cd
ab\textasciitildecd	ab˜cd	ab˜cd

4 Change history support

This package provides a straightforward yet flexible set of tools for tracking and managing changes across different versions. Each version is uniquely identified by its version number and has its release date.

A sample document, packdoc-change-history-example, is included with this package to demonstrate the use of versions and change markings.

4.1 Creating versions and changes

Creating versions and changes is straightforward. A version is created with \PDNewVersion, and each individual change is logged using \PDAddChange. The change history is then produced with \PDPrintChanges.

\PDNewVersion{⟨*version number*⟩}{⟨*version date*⟩}

The \PDNewVersion macro creates a new version entry in the change log. The first required parameter, ⟨*version number*⟩, is used to reference and group the changes made in that version. The version number can follow standard formats, such as `1.0` or `2.5.1`, for instance. For the ⟨*version date*⟩, a date in the *YYYY-MM-DD* format is typically used. This date is purely for display in the log, so the specific text format is flexible.

```
\PDNewVersion{1.0}{2025-01-01}
```

\PDAddChange{⟨*version number*⟩}{⟨*description*⟩}[⟨*box options*⟩]

The mandatory parameters for \PDAddChange include the ⟨*version number*⟩, which must have been previously defined using \PDNewVersion, and a comma-separated ⟨*description*⟩ list that outlines the specifics of the change.
The primary component of the ⟨*description*⟩ is, of course, `description`. In addition to this, several other options are available, which are outlined in Section 4.2.2.
The change boxes utilize the snaptodo package, meaning that the final optional parameter, ⟨*box options*⟩, can be used to adjust the appearance or modify other properties of the box.

```
\PDAddChange{1.0}{
    updated,
    description = {\Macro{SomeMacro} now allows floating point calculations.},
}
```

The change log is generated using the \PDPrintChanges macro, which functions similarly to other macros, such as \printindex.

\PDPrintChanges[⟨*options*⟩]

\PDPrintChanges generates the change log using a fixed, predefined format. Its position within the document is not important and can be determined based on the author's preference.

The ⟨*options*⟩ allow for customization of the `version prefix`, the `header style`, and the `entry style`. (See Section 4.2.)

```
\PDPrintChanges[version prefix = {V}]
```

4.2 Options for the Change History

This section outlines the options available for the change history. These are categorized into general options, which apply to the entire document, and specific options for the change record, which are limited to the particular change being marked.

4.2.1 General options

The following are the general options. All of them are defined using `\PDSet` and can be set either in the preamble or within the body of the text. They can also be used locally as options for `\PDPrintChanges` and `\PDAddChange`.

`version prefix` = ⟨*text*⟩ — Initially empty

The `version prefix` option sets a ⟨*text*⟩ that is added before the version number, appearing both in the change listing and in the margin boxes.

`header style` = ⟨*format*⟩ — Initially: `\bfseries\footnotesize`

This option defines the formatting commands for the style to be applied to each header line in the change history.

`entry style` = ⟨*format*⟩ — Initially: `\footnotesize\RaggedRight`

This option defines the formatting commands for the style to be applied to each change in the change history.

4.2.2 Change options

The options specific to changes are used to define each individual change and therefore have a local effect. However, some can be applied globally with `\PDSet`, as noted in their description.

`description` = ⟨*text*⟩

The description of a change refers to the ⟨*text*⟩ that will be included in the change history.
The `description` key can be omitted when the `no listing` option is applied.

`type` = ⟨*type*⟩

⟨*type*⟩ defines the type of change. Its value can be one of the following: `new`, `update`, `change`, `removal`, or `deprecation`. In practice, the key `type` is optional and one can specify directly the values of ⟨*type*⟩.
If no type is specified, the change defaults to `new`.

```
% The three change markers are equivalent.
\PDAddChange{1.0}{
    description = {A new feature has been implemented.},
}
\PDAddChange{1.0}{
    new,
    description = {A new feature has been implemented.},
}
\PDAddChange{1.0}{
    type = new,
    description = {A new feature has been implemented.},
}
```

`title` = ⟨*text*⟩

Using `title`, a ⟨*text*⟩ can be added to the box to provide additional relevant information.

```
\PDAddChange{1.0}{
    update,
    title = {Paragraphs},
    description = {A modification has been implemented regarding paragraphs.},
}
```

`no page` = `true` | `false` Default: `true`

The `no page` option is used to omit the page from the change log listing.
This option can be set globally with `\PDSet`.

```
\PDAddChange{1.0}{
    deprecation,
    description = {\Macro{OldThing} is no longer supported.},
    no page,
}
```

`no listing` = `true` | `false` Default: `true`

The `no listing` option prevents the change from being added to the change log.
This option can be set globally with `\PDSet`.

```
\PDAddChange{1.0}{
    deprecation,
```

```
    description = {\Macro{OldThing} is no longer supported.},
    no listing,
}
```

`no box` = `true` | `false` Default: `true`

The `no box` option prevents the change box from being displayed in the left margin. This option can be set globally with `\PDSet`.

`page` = ⟨*text*⟩

By default, the page number where a change was recorded is included in the change log. This can be modified using the `page` option, which allows for an alternative ⟨*text*⟩.
In a special case, an empty ⟨*text*⟩ is equivalent to `no page`.

```
\PDAddChange{1.0}{
    update,
    description = {A substantial amount of changes have been made.},
    page = {Chapt.~5.},
}
```

5 \PDSet

Certain options, both general and specific to the changes, can be defined globally using `\PDSet`.

`\PDSet{`⟨*options*⟩`}`

This macro enables certain settings for the change history and the changes themselves to be applied globally across the entire text, starting with the use of `\PDSet`.

```
\PDSet{
    version prefix = {V},  % 0.1 is displayed V0.1
    argument color = blue,  % color for <argument>
    no page,  % all pages are suppressed from the Change History
}
```

6 Issues

As this is the initial version, it is premature to provide a list of known issues. Therefore, any problems or suggestions may be submitted directly to me via email or by opening an issue on github (`https://github.com/jandermoreira/packdoc`).