# DoX – **D**oc, **o**nly e**X**tended*

Didier Verna
`mailto:didier@didierverna.net`
`http://www.lrde.epita.fr/~didier/`

v2.4 (2017/12/06)

**Abstract**

The `doc` package provides LaTeX developers with means to describe the usage and the definition of new commands and environments. However, there is no simple way to extend this functionality to other items (options or counters for instance). DoX is designed to circumvent this limitation, and provides some improvements over the existing functionality as well.

# Contents

---

*DoX homepage: `http://www.lrde.epita.fr/~didier/software/latex.php#dox`

# 1 Installation

## 1.1 Extraction

If you are building DoX from the tarball you need to execute the following steps in order to extract the necessary files:

```
[pdf]latex dox.ins
[pdf]latex dox.dtx
[pdf]latex dox.dtx
```

After that, you need to install the generated documentation and style files to a location where LaTeX can find them.

## 1.2 TDS-compliant layout

For a TDS-compliant layout, the following locations are suggested:

```
[TEXMF]/tex/latex/dox/dox.sty
[TEXMF]/doc/latex/dox/dox.[pdf|dvi]
```

## 1.3 AUCTeX support

AUCTeX is a powerful major mode for editing TeX documents in [X]Emacs. In particular, it provides automatic completion of command names once they are known. DoX supports AUCTeX by providing a style file named `dox.el` which contains AUCTeX definitions for the relevant commands. This file should be installed in a place where AUCTeX can find it. Please refer to the AUCTeX documentation for more information on this. See also section 3.

# 2 Usage

## 2.1 Initialization

### 2.1.1 Requirements

In order to work properly, DoX requires the presence of some LaTeX packages. You don't have to load them explicitly though. As long as LaTeX can locate them, they will be used automatically. DoX currently depends on `kvoptions`.

### 2.1.2 Loading the package

In order to load DoX, simply say `\usepackage[`⟨*options*⟩`]{dox}` in the preamble of your document. The package options will be discussed when appropriate.

## 2.2 Creating new documentation items

> ***Note:*** *we assume that you know about* `doc`*'s* `\DescribeMacro`*,* `\DescribeEnv` *and all other associated commands and environments.*

`\doxitem` `[`⟨*options*⟩`]{`⟨*name*⟩`}{`⟨*envname*⟩`}{`⟨*idxcat*⟩`}`
DoX provides a command named `\doxitem` to create new documentation items with functionalities equivalent to what `doc` provides for commands and environments. A whole API is created for every new item.

### 2.2.1 Example

Perhaps the simplest way to describe what it does is to give an example. Suppose you would like to describe package options explicitely. Here is what you need to do:

```
\usepackage{dox}
\doxitem{Option}{option}{options}
```

DoX then creates the following API for you:

- `\DescribeOption`
- the `option` environment
- `\PrintDescribeOption`
- `\PrintOptionName`
- `\SpecialMainOptionIndex`
- `\SpecialOptionIndex`

In order to comply with `doc`'s original behavior, the commands `\PrintDescribeOption` and `\PrintOptionName` will only be defined if they do not already exist.

### 2.2.2 Details

Here is a more precise description of the arguments to `\doxitem`.

- ⟨*name*⟩ (`Option` in our example) is used to construct the names of the commands in the new API. It usually starts with an upcase letter.
- ⟨*envname*⟩ (`option` in our example) is the name of the created environment. Be sure to avoid name clashes here! If you start experimenting odd behavior, you've probably overridden an existing command with your new environment.[1]

---

[1] It is a pity that LaTeX use the same namespace for commands and environments. The opening command for environment `env` should be named `\beginenv` and not just `\env`...

- ⟨*idxcat*⟩ is the index category under which your items will appear. In our example, all indexed options will be listed under the "options:" index entry.

### 2.2.3 Options to `\doxitem`

The first (optional) argument to `\doxitem` may contain a comma-separated list of options. The following ones are currently implemented.

`idxtype`

We saw earlier that individual items appear under the `idxcat` index entry, but items also appear as standalone index entries, alphabetically sorted with their type in parenthesis. For instance, the `final` option would appear like this under the `F` index entry: "`final` (option)".

By default, the index type is the same as the environment's name (the ⟨*envname*⟩ argument). However, you can change this by providing a value to the `idxtype` option.

For instance, you may find the word "environment" too long for an index type. In that case, you may redefine the `environment` API like this:

```
\doxitem[idxtype=env.]{Env}{environment}{environments}
```

`macrolike`

At some point, you may find that the "macro" category is too gross, and whish to further split it into different documentation items. A typical example of this would be to provide a specific category for LaTeX lengths.

The difference with options or counters is that the names of your items are actually control sequences, and so they need to be typeset and indexed in a specific way.

As of version 2.2, DoX provides a `macrolike` boolean option which instructs `\doxitem` to create new documentation items expecting control sequences instead of plain text names. If you want to create explicit documentation for lengths for instance, you can do this:

```
\doxitem[macrolike,idxtype=len.]{Length}{length}{lengths}
```

In turn this allows you do write things like that:

```
\DescribeLength{\partopsep}
\begin{length}{\partopsep}
  ...
\end{length}
```

## 2.3 Improvements over `doc`'s original API

*Please note that the improvements described in this section are also available in* `doc`*'s original command and environment* API*'s, because DoX redefines them.*

### 2.3.1 Additional (optional) argument

Compared to `doc`, the API's created by `\doxitem` are extended with a first optional argument containing a comma-separated list of options, such that, continuing with our initial example, the real prototypes are in fact the following:

```
\DescribeOption[opt,...]{name}

\begin{option}[opt,...]{name}
%% ...
\end{option}
```

### 2.3.2 Available options

The options currently supported are:

**noprint** Avoid printing ⟨*name*⟩ in the margin

**noindex** Avoid indexing ⟨*name*⟩

These options are useful if you don't want printing or indexing locally for one particular item. Without them, one would need to locally `\let` the relevant commands to `\@gobble` which is very inconvenient.

There is also another advantage in using the `noprint` option: in `doc`'s original implementation, a margin paragraph will still be created just to be empty, hence wasting float resources. If you're referencing a lot of items close to each other, this may lead to a "Too many unprocessed floats" error. With DoX, the `\marginpar` is avoided altogether.

### 2.3.3 Global effect

The `macrolike`, `noprint` and `noindex` options are also available to `\usepackage`. Their effect then becomes global. Because these options are boolean, it is still possible to counteract their global effect locally. For instance, one could do:

```
\usepackage[noprint]{dox}
```

and then later:

```
\DescribeOption[noprint=false]{french}
```

## 3 AUCTEX support for new documentation items

Recent versions of AUCTEX (in fact, `docTEX` mode) are aware of the `macro` and `environment` environments and give them a fixed indentation level of 0, meaning no indentation at all even when they are nested. This is considered more convenient than the usual indentation for environments when editing `dtx` files. If you have created new documentation items for your package, you may want to let them behave the same way. For that, the DoX style file provides two Lisp functions to let AUCTEX know of your new environments: `doxitem` and `doxitems`. The first one registers a new environment by name with AUCTEX, and the second one takes an arbitrary number of environment names and does the same with them. The environment names can in fact be regular expressions, allowing you to combine several names together or build complex ones.

Since these functions are located in the style file itself, a good place to use them is in `TeX-update-style-hook` which will be called after the file is parsed and the relevant style files applied. Note that the effect of calling these functions is always buffer-local.

Here is an example to make all of this clearer. The following code sample is what I have at the end of `fixme.dxt` (another package of mine), in the local variables section:

```
(add-hook 'TeX-update-style-hook
  (lambda () (doxitems "option" "counter" "lang" "face" "color")) nil t)
(add-hook 'TeX-update-style-hook
  (lambda () (doxitem "\\(env\\|target\\)?layout")) nil t)
```

# 4 Conclusion

If you want to see DoX in action, take a look at the documentation of the FiXme package (version 4.0 or later). In fact, I wrote DoX for it in the first place.

# 5 History

v2.4 Improve AUCTeX support.
Fix spurious spaces in index entries, reported by Eric Domenjoud.

v2.3 Support `doc`'s `\saved@indexname` internal command, thanks to Falk Hanisch.

v2.2 New `macrolike` option allowing to create control sequence based documentation items.

v2.1 New lisp functions `doxitem[s]` to register new documentation environments with AUCTeX.

v2.0 Optional argument to `\doxitem` (`idxtype` option to change the item's index type).
Optional argument to `\Describe`⟨*Item*⟩ and the ⟨*item*⟩ environment (`noprint` to avoid marginal printing and `noindex` to avoid indexing).
Extend `\DescribeMacro`, `\DescribeEnv` and their corresponding environments with the same features.

v1.0 First public version.

# 6 Implementation

## 6.1 Preamble

```
⟨dox⟩\NeedsTeXFormat{LaTeX2e}
⟨*header⟩
\ProvidesPackage{dox}[2017/12/06 v2.4 Extensions to the doc package]
```

```

⟨/header⟩
⟨*dox⟩
\RequirePackage{kvoptions}
\SetupKeyvalOptions{family=dox,prefix=dox@}

```

## 6.2 DoX options

These two options are available for use in `\usepackage` or in the generated item API's:

```
\DeclareBoolOption{noprint}
\DeclareBoolOption{noindex}
```

This one is for `\usepackage` and `\doxitem`:

```
\DeclareBoolOption{macrolike}
```

This one is for `\doxitem`:

```
\DeclareStringOption{idxtype}

```

## 6.3 DoX environments

`\@@doxenv` `{`⟨*item*⟩`}{`⟨*name*⟩`}`

In `doc.sty`, the `macro` and `environment` environments go through the `\m@cro@` macro which implements specific parts by testing a boolean condition as its first argument. This mechanism is not extensible, so I have to hack away a more generic version that would work for any new `dox` item, only which looks pretty much like the original one (with the addition of options management).

```
\long\def\@@doxenv#1#2{%
  \endgroup%
  \topsep\MacroTopsep%
  \trivlist%
    \edef\saved@macroname{\string#2}%
```

Since version 2.1g, `doc` creates a `\saved@indexname` command which in used by `\changes`. We now support that as well. The expansion of this command depends on whether the documented item is macrolike or not, which we don't know here (it's only know by `\doxitem`). That's why we need one specific command generating `\saved@indexname` the right way for every single item. These commands are named`\@Save`⟨*item*⟩`IndexName`; they are technically part of the generated API, only not meant for public use.

```
    \@nameuse{@Save#1IndexName}{\saved@macroname}%
    \def\makelabel##1{\llap{##1}}%
    \if@inlabel%
      \let\@tempa\@empty%
      \count@\macro@cnt%
      \loop\ifnum\count@>\z@%
        \edef\@tempa{\@tempa\hbox{\strut}}\advance\count@\m@ne%
      \repeat%
      \edef\makelabel##1{\llap{\vtop to\baselineskip{\@tempa\hbox{##1}\vss}}}%
      \advance\macro@cnt\@ne%
    \else%
      \macro@cnt\@ne%
    \fi%
    \ifdox@noprint%
      \item%
    \else%
      \edef\@tempa{%
        \noexpand\item[%
```

The second notable modification to the original macro involves dynamically constructing the name of the print macro:

```
        \expandafter\noexpand\csname Print#1Name\endcsname{\saved@macroname}]}%
      \@tempa%
    \fi%
    \ifdox@noindex\else%
      \global\advance\c@CodelineNo\@ne%
```

and the third one involves dynamically constructing the name of the index macro:

```
      \@nameuse{SpecialMain#1Index}{#2}\nobreak%
      \global\advance\c@CodelineNo\m@ne%
    \fi%
    \ignorespaces}
```

`\@doxenv` {⟨*item*⟩}[⟨*options*⟩]

Handle optional arguments and call `\@@doxenv`. Because environments can be nested, we can't rely on grouping for getting options default values. Hence, we need to reset the options at every call.

```
\def\@doxenv#1[#2]{%
  \@nameuse{dox@noprint\dox@noprintdefault}%
  \@nameuse{dox@noindex\dox@noindexdefault}%
  \setkeys{dox}{#2}%
  \begingroup%
    \catcode`\\12%
    \MakePrivateLetters%
    \@@doxenv{#1}}

```

## 6.4 DoX descriptions

`\@@doxdescribe` {⟨*item*⟩}{⟨*name*⟩}

The first closed group was the one opened to parse the ⟨*name*⟩ argument. The second one was opened to handle local options.

```
\def\@@doxdescribe#1#2{%
    \endgroup%
    \ifdox@noprint\else%
      \marginpar{\raggedleft\@nameuse{PrintDescribe#1}{#2}}%
    \fi%
    \ifdox@noindex\else%
      \@nameuse{Special#1Index}{#2}%
    \fi%
    \endgroup%
  \@esphack\ignorespaces}
```

`\@doxdescribe` {⟨*item*⟩}[⟨*options*⟩]

Handle optional arguments and call `\@@doxdescribe`.

```
\def\@doxdescribe#1[#2]{%
```

```
  \leavevmode\@bsphack%
  \begingroup%
    \setkeys{dox}{#2}%
    \begingroup%
      \MakePrivateLetters%
      \@@doxdescribe{#1}}

```

## 6.5 API construction

First, the two index name generation macros (macrolike or not).

`\@doxsavemacrolikeindexname` {⟨*saved macro name*⟩}

```
\def\@doxsavemacrolikeindexname#1{%
  \edef\saved@indexname{\expandafter\@gobble#1}}
```

`\@doxsaveindexname` {⟨*saved macro name*⟩}

```
\def\@doxsaveindexname#1{\let\saved@indexname#1}

```

`\@doxcreatespecialmainindex` {⟨*item*⟩}{⟨*idxtype*⟩}{⟨*idxcat*⟩}

`eatespecialmainmacrolikeindex` {⟨*item*⟩}{⟨*idxtype*⟩}{⟨*idxcat*⟩}

The "macrolike" version does something similar to `doc`'s `\SpecialIndex@` macro, but simplified. Let's just hope nobody will ever define `\␣` or nonletter macros as macrolike DoX items…

```
\def\@doxcreatespecialmainindex#1#2#3{%
  \expandafter\def\csname SpecialMain#1Index\endcsname##1{%
    \@bsphack%
    \special@index{##1\actualchar{\string\ttfamily\space##1} (#2)%
      \encapchar main}%
    \special@index{#3:\levelchar##1\actualchar{\string\ttfamily\space##1}%
      \encapchar main}%
    \@esphack}}
\def\@doxcreatespecialmainmacrolikeindex#1#2#3{%
  \expandafter\def\csname SpecialMain#1Index\endcsname##1{%
    \@SpecialIndexHelper@##1\@nil
    \@bsphack%
    \special@index{\@gtempa\actualchar
      \string\verb\quotechar*\verbatimchar\bslash\@gtempa\verbatimchar
      \space(#2)\encapchar main}%
    \special@index{#3:\levelchar\@gtempa\actualchar%
      \string\verb\quotechar*\verbatimchar\bslash\@gtempa\verbatimchar%
      \encapchar main}%
    \@esphack}}

```

`\@doxcreatespecialindex` {⟨*item*⟩}{⟨*idxtype*⟩}{⟨*idxcat*⟩}

`oxcreatespecialmacrolikeindex` {⟨*item*⟩}{⟨*idxtype*⟩}{⟨*idxcat*⟩}

Same comment as above.

```
\def\@doxcreatespecialindex#1#2#3{%
  \expandafter\def\csname Special#1Index\endcsname##1{%
    \@bsphack%
    \index{##1\actualchar{\protect\ttfamily##1} (#2)\encapchar usage}%
    \index{#3:\levelchar##1\actualchar{\protect\ttfamily##1}%
      \encapchar usage}%
    \@esphack}}
\def\@doxcreatespecialmacrolikeindex#1#2#3{%
  \expandafter\def\csname Special#1Index\endcsname##1{%
    \@SpecialIndexHelper@##1\@nil
    \@bsphack%
    \index{\@gtempa\actualchar
      \string\verb\quotechar*\verbatimchar\bslash\@gtempa\verbatimchar
      \space(#2)\encapchar usage}%
    \index{#3:\levelchar\@gtempa\actualchar
      \string\verb\quotechar*\verbatimchar\bslash\@gtempa\verbatimchar%
      \encapchar usage}%
    \@esphack}}

```

`\@doxcreatedescribe` {⟨*item*⟩}

```
\def\@doxcreatedescribe#1{%
  \expandafter\def\csname Describe#1\endcsname{%
    \@ifnextchar[%]
    {\@doxdescribe{#1}}{\@doxdescribe{#1}[]}}}

```

`\@doxcreateenv` {⟨*item*⟩}{⟨*envname*⟩}

```
\def\@doxcreateenv#1#2{%
  \expandafter\def\csname #2\endcsname{%
    \@ifnextchar[%]
    {\@doxenv{#1}}{\@doxenv{#1}[]}}
  \expandafter\let\csname end#2\endcsname\endtrivlist}

```

## 6.6 `Doc` overrides

### 6.6.1 Macro facilities

Making `\DescribeMacro` work the DoX way is straightforward. The only precaution we need is to provide an alias to `\SpecialUsageIndex` because it should really be named `\SpecialMacroIndex`.

```
\let\SpecialMacroIndex\SpecialUsageIndex
\@doxcreatedescribe{Macro}

```

Making the `macro` environment work the DoX way is straightforward. The only precaution we need is to to provide a `\SpecialMainMacroIndex` macro that does the job originally done in `doc`'s `\m@cro@`.

```
\def\SpecialMainMacroIndex#1{%
  \SpecialMainIndex{#1}\nobreak%
  \DoNotIndex{#1}}
\let\@SaveMacroIndexName\@doxsavemacrolikeindexname
\@doxcreateenv{Macro}{macro}

```

### 6.6.2 Environment facilities

Making `\DescribeEnv` and the `environment` environment work the DoX way is even more straightforward.

```
\@doxcreatedescribe{Env}
\@doxcreateenv{Env}{environment}
\let\@SaveEnvIndexName\@doxsaveindexname

```

## 6.7 API creation

The whole user interface is created in one macro call.

`\doxitem` [⟨*options*⟩]{⟨*name*⟩}{⟨*envname*⟩}{⟨*idxcat*⟩}

```
\newcommand\doxitem[4][]{%
  \@nameuse{dox@macrolike\dox@macrolikedefault}%
  \def\dox@idxtype{#3}%
  \setkeys{dox}{#1}
```

`\Print...Name` {⟨*name*⟩}

```
  \@ifundefined{Print#2Name}{%
    \ifdox@macrolike
      \expandafter\def\csname Print#2Name\endcsname##1{%
        \strut\MacroFont\string ##1\ }
    \else
      \expandafter\def\csname Print#2Name\endcsname##1{%
        \strut\MacroFont ##1\ }
    \fi}{}
```

`\SpecialMain...Index` {⟨*name*⟩}

```
  \ifdox@macrolike
    \def\@doxexpr{\@doxcreatespecialmainmacrolikeindex{#2}}%
  \else
    \def\@doxexpr{\@doxcreatespecialmainindex{#2}}%
  \fi
  \expandafter\@doxexpr\expandafter{\dox@idxtype}{#4}%
```

`\PrintDescribe...` {⟨*name*⟩}

```
  \@ifundefined{PrintDescribe#2}{%
    \ifdox@macrolike
      \expandafter\def\csname PrintDescribe#2\endcsname##1{%
        \strut\MacroFont\string ##1\ }
    \else
      \expandafter\def\csname PrintDescribe#2\endcsname##1{%
        \strut\MacroFont ##1\ }
    \fi}{}
```

`\Special...Index` {⟨*name*⟩}

```
  \ifdox@macrolike
    \def\@doxexpr{\@doxcreatespecialmacrolikeindex{#2}}%
  \else
    \def\@doxexpr{\@doxcreatespecialindex{#2}}%
  \fi
  \expandafter\@doxexpr\expandafter{\dox@idxtype}{#4}%
```

`\Describe...` [⟨*options*⟩]{⟨*name*⟩}

```
\@doxcreatedescribe{#2}
```

`item` [⟨*options*⟩]{⟨*name*⟩}

```
\@doxcreateenv{#2}{#3}
```

`\@Ssave...IndexName`

```
\ifdox@macrolike
  \expandafter\let\csname @Save#2IndexName\endcsname%
    \@doxsavemacrolikeindexname%
\else
  \expandafter\let\csname @Save#2IndexName\endcsname\@doxsaveindexname%
\fi}
```

## 6.8 Finale

We need to save the default value for every option because DoX environments need to reset them at every call.

```
\ProcessKeyvalOptions*
\ifdox@noprint
  \def\dox@noprintdefault{true}
\else
  \def\dox@noprintdefault{false}
\fi
\ifdox@noindex
  \def\dox@noindexdefault{true}
\else
  \def\dox@noindexdefault{false}
\fi
\ifdox@macrolike
  \def\dox@macrolikedefault{true}
\else
  \def\dox@macrolikedefault{false}
\fi

⟨/dox⟩
```