The **skdoc** document class*†

Simon Sigurdhsson [sigurdhsson@gmail.com]

Version 1.0

Abstract The skdoc class provides macros to document the functionality and implementation of LaTeX packages and document classes. It is loosely based on the ydoc and ltxdoc classes, but has a number of incompatible differences.
The class defines a `MacroCode` environment which substitutes the usual docstrip method of installing packages. It has the ability to generate both documentation and code in a single run of a single file.

Contents

*Available on `http://www.ctan.org/pkg/skbundle`.
†Development version available on `https://github.com/urdh/skdoc`.

1 Introduction

This document class, inspired by a question on the TEX Stack Exchange[1], aims to provide an alternative to the standard docstrip method of literate programming for LATEX packages. It also aims to provide a more modern, appealing style for LATEX package documentation.

In order to achieve this, it builds upon already existing features of the expl3, verbatim and ydoc packages as well as the KOMA-script document classes.

So far it is mainly intended to be an experiment to explore a less cumbersome way of writing LATEX packages, and as such I give no guarantee that this package will continue to exist in a working state (*i.e.* future users may not be able to extract code from package documentation based on skdoc) or that its public API (commands and environments described by this documentation; consider undocumented macros part of a private API) will be stable.

The documentation of skdoc is in fact typeset using the class itself. It does not, however, make use of the main feature of this class (the `MacroCode` environment), because bootstrapping the class to generate itself is more complicated than it is useful.

2 Documentation

Since skdoc is based on ydoc many of the macros and environment present in that package are also available in skdoc, in a possibly redefined

[1] Lazarides 2012.

incanation. However, any macros or environment present in ydoc but not described in this documentation should be considered part of the private API of skdoc. In the future, the removal of the ydoc dependency may result in such macros being unavailable, and at present changes made by skdoc may break such macros without notice.

2.1 Options

`load` ⟨*package*⟩ (`\jobname`)

The `load` option declares that if the package specified exists, it should be loaded. This is intended to load any package provided in the implementation, but requires that the documentation provides stub variants of the macros used in the documentation so that LaTeX still completes its first run.

> **Warning:** The `load` option will likely disappear in a future version.

`highlight` `true,false` (`true`)

The `highlight` option enables or disabled syntax highlighting of the implementation code. Highlighting is performed using minted, and falls back to no highlighting if there is no `\write18` access or if the `pygmentize` binary can't be found. (***Note:*** *On non-unix platforms, the test for* `pygmentize` *will likely fail. Therefore, syntax highlighting is not supported on such platforms.*)

Generally, there should be no reason to disable syntax highlighting unless the documentation describes a very large package, and the repeated calls to `pygmentize` take too long.

2.2 General macros

The document class defines a number of general macros intended for use in parts of the document not strictly considered 'documentation' or 'implementation', in addition to being used in those parts. These 'general' macros include macros that define metadata, generate the title page, typeset notices or warnings and those that refer to macros, environments, packages and such.

2.2.1 Metadata

Several macros for defining metadata (*i.e.* information about the package and its documentation) are made available. These mostly set an internal variable which is used to typeset the title page, and to insert PDF metadata whenever pdfLaTeX is used to generate the documentation.

`\package` `[ctan=`⟨*identifier*⟩`,vcs=`⟨*url*⟩`]{`⟨*package name*⟩`}`

The `\package` macro defines the package name used by `\thepkg`, `\maketitle` and similar macros. It also calls `\title` to set a sensible default title based on the package name. The optional key-value argument takes two keys: `ctan` and `vcs`. The first one accepts an optional value ⟨*identifier*⟩, which should be the identifier the package has on CTAN (the default is `\jobname`), while the other takes a mandatory argument ⟨*url*⟩ specifying the URL of a VCS repository where development versions of the package are available. The two optional keys imply calls to the `\ctan` and `\repository` macros, respectively.

`\version` `{`⟨*version*⟩`}`

Sets the version number of the package the documentation describes. Here, ⟨*version*⟩ should not include the initial 'v', *i.e.* the argument should be the same as that given to *e.g.* the LaTeX3 `\ProvidesExplPackage` or the standard ltxdoc `\changes`.

`\ctan` `{`⟨*identifier*⟩`}`

As detailed above, this macro defines the CTAN identifier of the package, which is (optionally) used in the `\maketitle` macro.

`\repository` `{`⟨*url*⟩`}`

Again, as detailed above this macro defines the URL of a source code repository containing a development version of the package, which is optionally used by `\maketitle`.

`\author` `{`⟨*name*⟩`}`

Defines the name of the package author. This is used by `\maketitle` and is mandatory if `\maketitle` is used.

`\email` `{`⟨*email*⟩`}`

Defines the email of the package author. This is used by `\maketitle` and is mandatory if `\maketitle` is used.

`\title` `{`⟨*title*⟩`}`

Defines the package title. By default, the `\package` macro sets a sensible title that should suit most packages, but using `\title` will override this title (useful for *e.g.* document classes or BibTeX styles).

Three macros retrieving the set metadata are also available. They can be used to typeset the current version of the package, and the package name, respectively.

`\theversion`

Returns the version as defined by `\version`, with a leading 'v'. That is, issuing `\version{1.0}` makes `\theversion` print 'v1.0'.

`\thepackage`
`\thepkg`

The `\thepackage` and `\thepkg` macros return the package name as defined by the `\package` macro, enclosed in `\pkg*`. That is, the package name is typeset as a package but not indexed.

2.2.2 The preamble

The preamble of any documentation most often consists of a title page containing an abstract, and possibly a table of contents. The skdoc package provides macros and environments that typeset such things, and these should be fully compatible with most other document classes.

`\maketitle`

The `\maketitle` macro typesets a title page. This title page uses the metadata defined by the macros described earlier, and typesets them in a manner which is illustrated by the documentation of this class. This style is inspired by skrapport, which is in turn inspired by the title pages of the PracTEX Journal.

`\begin{abstract}`
⟨*package abstract*⟩
`\end{abstract}`

The `abstract` environment typesets an abstract of the package. Again, its style is illustrated by this document and it is inspired by the skrapport package as well as the PracTEX Journal.

`\tableofcontents`

Finally, a Table of Contents may be printed. The actual table of contents is provided by the scrartcl document class, but skdoc redefines a couple of the internal macros to style the Table of Contents in a manner similar to that of the microtype manual.

2.2.3 The LPPL license

`\PrintLPPL`

If the LPPL license is present in a directory where LATEX can find it, in a file called `lppl.tex`, then `\PrintLPPL` will include the entire LPPL license in the document, and typeset it in a fairly compact manner.

2.2.4 Notices and warnings

The document class provides macros to indicate information that may be of extra importance in the documentation. Such information is categorized as either notices or warnings, which are treated differently.

`\Notice` {⟨*notice*⟩}

A notice is a short piece of text that contains information that may explain some unexpected but unharmful behaviour of a macro or similar. It is typeset inline, emphasized and in parantheses — as such, the sequence `\Notice{a notice}` yields (***Note:*** *a notice*).

`\Warning` {⟨*warning*⟩}

A warning is a short comment that conveys information that the user must be aware of. This includes unexpected potentially harmful behaviour, deprecation notices and so on. It is typeset in its own `\fbox` — the sequence `\Warning{a warning}` yields the following:

Warning: a warning

`\LongWarning` {⟨*warning*⟩}

The `\LongWarning` macro is a variant of `\Warning` that has been adapted for longer texts, possibly including paragraph breaks. Like `\Warning`, it is typeset in a box:

Warning:
a long warning

2.2.5 Referential macros

The family of macros originating from `\cs` are used to typeset various concepts in running text. In addition to adhering to the general format of the corresponding concept, they index their argument. Each of these macros have a starred variant which does not index its argument; use these when appropriate.

`\cs` {⟨*command sequence*⟩}

Typesets a command sequence, or macro. The argument should be provided without the leading backslash, and the command sequence will be typeset in a monospaced font.

`\env` {⟨*environment name*⟩}

Typesets an environment name, which will be typeset in a monospace font.

`\pkg` {⟨*package name*⟩}

Typesets a package, document class or bundle name. The name will be typeset in a sans-serif font.

`\opt` {⟨*option*⟩}

Typesets a package or document class option. As of v1.0, options are typeset using a monospace font.

`\bib` {⟨*BibTeX entry type*⟩}

Typesets a BibTeX entry type. The agument should be provided without the leading `@` sign. The entry type will be typeset in a monospace font.

`\thm` {⟨*theme name*⟩}

Typesets a theme name. As of v1.0, the theme name will be typeset in an upright serif font.

`\file` {⟨*filename*⟩}

Typesets a filename. As of v1.0, the filename will be typeset in a monospace font.

2.3 Documenting the package

The documentation part of any LaTeX manual is arguably the most important one, and to facilitate proper typesetting of the documentation

Table 1: *Typesetting arguments*

Invokation	Result
`\marg{argument}`	{⟨*argument*⟩}
`\oarg{argument}`	[⟨*argument*⟩]
`\parg{argument}`	(⟨*argument*⟩)
`\aarg{argument}`	<⟨*argument*⟩>
`\sarg`	*

skdoc povides a number of different macros, all inspired by or inherited from ydoc. The first of these macros that will be discussed are the macros that typeset differen kinds of arguments in running text.

`\meta` {⟨*meta text*⟩}

The `\meta` macro typesets a placeholder to be placed in an argument. This can be used to refer to arguments and contents of macros and environments described by commands discussed later in this documentation. It is typeset in brackets: ⟨*meta text*⟩.

`\marg` {⟨*mandatory argument*⟩}
`\oarg` {⟨*optional argument*⟩}
`\parg` {⟨*picture-style argument*⟩}
`\aarg` {⟨*beamer-style argument*⟩}
`\sarg`

These macros typeset different kinds of arguments (mandatory, optional, picture-style, beamer-style and star arguments, respetively). These can be used to describe arguments, but are mostly used internally. See table 1 for examples of how these macros are typeset.

2.3.1 Options

Package options are of course important to describe, and to this end four macros are provided. They aid in describing options of both regular boolean and the more modern key-value syntax. They are intended to

be used in a sequence:
`\Option{...}\WithValues{...}\AndDefault{...}`

`\Option` `{`⟨*option*⟩`}`
`\Options` `{`⟨*option*⟩`,...}`

These macros typeset an option, and may be followed by the `\WithValues` macro (***Note:*** *the with `\Options`, only the first option in the list will work with `\WithValues`*).

`\WithValues` `{`⟨*value*⟩`,...}`

This macro typesets a comma-separated list of values a specific option can take. It may be followed by the `\AndDefault` macro.

`\AndDefault` `{`⟨*default value*⟩`}`

This macro typesets the default value of an option. It may follow either `\Options` or `\WithValues`.

Common constructs using these macros include:

- `\Option{`⟨*option*⟩`}\WithValues{`⟨*value*⟩`,...}\AndDefault{`⟨*default*⟩`}`
- `\Options{`⟨*option*⟩`,no`⟨*option*⟩`}\AndDefault{no`⟨*option*⟩`}`

2.3.2 Macros

The skdoc class inherits a number of macros for describing the package macros from the ydoc package. Only four of them are to be considered stable.

Warning:
The macros `\MakeShortMacroArgs` and `\DeleteShortMacroArgs` and the environments `DescribeMacros` and `DescribeMacrosTab` provided by ydoc are unsupported as of skdoc v1.0. They may work, but this is not a guarantee and they are most likely broken or may break other features of skdoc.

`\DescribeMacro` ⟨`\`*macro*⟩⟨*macro arguments*⟩

The `\DescribeMacro` macro documents a macro along with its arguments. Any number of ⟨*macro arguments*⟩ may follow the macro, and `\DescribeMacro` will stop reading arguments on the first non-argument token. The macro will be indexed.

Warning:
Although ⟨`\`*macro*⟩ can include `@` signs, it is not possible to document LaTeX3-style macros (with underscores and colons) without the following hack:

```
\ExplSyntaxOn
\cs_set_protected_nopar:Npn\ExplHack{
  \char_set_catcode_letter:n{ 58 }
  \char_set_catcode_letter:n{ 95 }
}
\ExplSyntaxOff
\ExplHack
```

`\Macro` ⟨`\`*macro*⟩⟨*macro arguments*⟩

This is simply a variant of `\DescribeMacro` for use in running text. It is equivalent to `\MacroArgs\AlsoMacro`.

`\MacroArgs` ⟨*macro aguments*⟩

This macro formats ⟨*macro arguments*⟩ the same way `\DescribeMacro` does. As with `\Macro`, it is used in running text.

`\AlsoMacro` ⟨`\`*macro*⟩⟨*further arguments*⟩

This macro should be used inside ⟨*macro arguments*⟩ of the macros described above, and typesets an additional macro as part of the syntax of the described macro. For instance, the `\csname` macro could be described with the sequence `\Macro\csname<command sequence name>\AlsoMacro\endcsname`, which would be rendered as `\csname`⟨*command*

sequence name⟩\endcsname .

2.3.3 Environments

In addition to the macros describing macros, skdoc also inherits one environment and one macro to describe environments. These are similar to the macros described previously in both form and function, but lack equivalents for running text.

\DescribeEnv [⟨*body content*⟩]{⟨*name*⟩}⟨*arguments*⟩

This macro describes an environment, in the same way \DescribeMacro does for macros. The ⟨*body content*⟩, which is optional, may be used to indicate what kind of content the environment is designed to contain. The \MacroArgs macro is automatically inserted before ⟨*body content*⟩.

2.3.4 Other entities

The document class also provides macros to describe BibTeX entries and generic themes. The BibTeX entries are described using the \BibEntry and \WithFields macros, while themes are described using the \Theme macro.

\BibEntry {⟨*entry name*⟩}\WithFields[⟨*optional fields*⟩]{⟨*mandatory fields*⟩}

These two macros describe a BibTeX entry named ⟨*entry name*⟩ (*i.e.*, @⟨*entry name*⟩) along with its optional and mandatory fields.

\Theme {⟨*theme name*⟩}

This macro describes a theme named ⟨*theme name*⟩. These could be used to describe any kind of theme, such as color themes of a document class.

\DescribeFile {⟨*filename*⟩}

This macro describes a special file named ⟨*filename*⟩. This could be a configuration file or similar that is either part of the package or something the package reads if available.

2.4 Describing the implementation

In true TEX (and literal programming) fashion the document class also provides ways to describe, in detail, parts of the implementation. The most essential of the implementation environments, without which skdoc doesn't generate any files, is the `MacroCode` environment. Other than that, the implementation environments should be compatible with or analogous to the standard ltxdoc document class.

2.4.1 Implementation environments

The environments described in this section indicate the implementation of different concepts including macros, environments and options. They each have a starred variant which doesn't print the concept name (only indexes it), and a non-starred variant which does (***Note:*** *inside these environments,* `\changes` *will refer to the relevant entity instead of logging 'general' changes*).

Some of the following environment can typeset descriptions of the internal arguments (`#1`, `#2` *etc.*) to improve readability of the implementation code.

`\begin{macro}` `{`⟨`\macro`⟩`}`⟨*# of args*⟩`{`⟨*arg 1 description*⟩`}...{`⟨*arg n description*⟩`}`
⟨*description*⟩
`\end{macro}`

With this environment, the implementation of a macro is described. Note that as with `\DescribeMacro`, LATEX3-style macros can not be used in ⟨`\macro`⟩ without the catcode hack mentioned earlier.

`\begin{environment}` `{`⟨*environment*⟩`}`⟨*# of args*⟩`{`⟨*arg 1 description*⟩`}...{`⟨*arg n description*⟩`}`
⟨*description*⟩
`\end{environment}`

This environment describes the implementation of an environment.

`\begin{option}` `{`⟨*option*⟩`}`
⟨*description*⟩
`\end{option}`

This environment describes the implementation of an option.

`\begin{bibentry}` `{`⟨*@entry*⟩`}`
⟨*description*⟩
`\end{bibentry}`

This environment describes the implementation of a BIBTEX entry type.

`\begin{theme}` `{`⟨*theme*⟩`}`
⟨*description*⟩
`\end{theme}`

This environment describes the implementation of a theme.

2.4.2 The `MacroCode` environment

The 'main event' of the skdoc document class is the `MacroCode` environment. It has roughly the same role the `macrocode` environment has in the docstrip system, except that it in addition to typesetting the implementation also saves it to the target files.

The workflow is simple; before using `MacroCode` to export code to a file the file must be declared using `\DeclareFile`, which also assigns a key to the file (the default is the filename). This key is passed to the `MacroCode` environment, which saves the code to the specified file.

`\DeclareFile` `[key=`⟨*key*⟩`,preamble=`⟨*preamble*⟩`]{`⟨*filename*⟩`}`

The `\DeclareFile` macro declares a file for future use with `MacroCode`. The optional argument is a comma separated list of key-value options, where the possible keys are `key` and `preamble`. Here ⟨*key*⟩ is a key that is used instead of the filename in `MacroCode`, and ⟨*preamble*⟩ is a token or command sequence expanding to a preamble which will be prepended to the file on output.

`\PreambleTo` `{`⟨`\`*token*⟩`}{`⟨*filename*⟩`}`

Reads the preamble from ⟨*filename*⟩. Lines from the file are appended to ⟨`\`*token*⟩ until a line which does not begin with `%%` is encountered.

`\SelfPreambleTo` `{`⟨`\`*token*⟩`}`

This reads the preamble from the curent file. It is equivalent to the sequence `\PreambleTo{`⟨`\`*token*⟩`}{\jobname.tex}`.

`\begin{MacroCode}` `{`⟨*key*⟩`}`
⟨*implementation*⟩
`\end{MacroCode}`

The `MacroCode` environment typesets and exports ⟨*implementation*⟩ verbatim to the file associated with ⟨*key*⟩. As such, it is the analogue of the `macrocode` environment from ltxdoc, but does not suffer from some of its drawbacks (the sensitivity to whitespace, for instance).

2.4.3 Hiding the implementation

For lagre packages it may be of interest to hide the implementation from the documentation. This is accomplished using the two marker macros `\Implementation` and `\Finale` (which should be present even if not hiding the implementation), and the switch macro `\OnlyDescription`.

`\Implementation`

This macro indicates the start of the implementation. Normally, this would directly precede the `\section` under which the implementation is organized.

`\Finale`

This macro indicates the end of the implementation. Usually the only things happening after this is the printing of indices, the change log, bibliographies and the end of the `document` environment.

`\OnlyDescription`

This macro, which should be issued in the preamble, indicates that the implementation should be hidden.

> **Warning:** this has the side effect that a page break is inserted where the implementation would normally reside.

2.5 Documenting changes

One type of useful information you should provide in your documentation is a list of changes. The skdoc document class provides a change list system based on the glossaries package. As such, including a change list in your documentation requires you to run `makeglossaries` between the first and second LaTeX run.

`\changes` `{`⟨*version*⟩`}{`⟨*description*⟩`}`

The `\changes` macro provides the main interface to the change list system, and adds changes to the change list. Each change is added with a *context*; if the macro is issued inside one of the macros described in section 2.4.1, the concept currenly being described will be the context. Outside these environment, the context is 'general'. For every context and ⟨*version*⟩, only one change may be recorded, otherwise glossaries will issue a warning.

`\PrintChanges`

This macro prints the list of changes. As explained earlier, this requires you to run `makeglossaries` between the two LaTeX runs.

2.6 Producing an index

The macros previously discussed in sections 2.2.5, 2.3 and 2.4.1 automatically index their aguments using glossaries. By running `makeglossaries` you can include an index of all macros, environments, packages and such that are discussed, documented or implemented in your package.

`\PrintIndex`

Much like the `\PrintChanges` macro, this prints the index. As with the list of changes, this requires that you run `makeglossaries` between the two LaTeX runs.

3 Changes

v1.0

General: Initial version.

4 Index

Numbers written in boldface refer to the page where the corresponding entry is described; numbers underlined refer to the page were the implementation of the corresponding entry is discussed. Numbers in roman refer to other mentions of the entry.

5 Bibliography

Lazarides, Yannis (2012). *Different approach to literate programming for LaTeX*. Url: http://tex.stackexchange.com/questions/47237/different-approach-to-literate-programming-for-latex.