The **skdoc** document class*†

Simon Sigurdhsson [sigurdhsson@gmail.com]

Version 0.9

Abstract The skdoc class provides macros to document the functionality and implementation of LaTeX packages and document classes. I 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/skdoc`.
†Development version available on `https://github.com/urdh/skdoc`.

1 Introduction

2 Documentation

2.1 Planned improvements

Planned improvements of this class include

- Proper documentation of all features, including those inherited from ydoc.
- Do things the LaTeX3 way, rename the variables properly and generally conform to expl3 guidelines when applicable.
- Better color scheme.
- `\DescribeEnvironment` has to be improved. The idea is to typeset the first line as with the macros, but the environment contents and `\end` command typeset below in a `\marginnote`.

- More organized font scheme. \cs, \env, \pkg, \opt, \thm and \bib must have sensible fonts and these should be reflected in the implementation details as well as the description.
- The uses of \cs and friens should be recorded in the index as well.
- Restructure the code.
- Starred variants of the implementation environments have to gobble the arguments that are normally printed!
- Themes and BibTeX entry types need macros in the spirit of \Option and friends.

3 Implementation

3.1 Require packages

We begin with loading the scrartcl KOMA-script class and a few packages we'll be needing.

```
\LoadClass[ DIV7,
            headings=big,
            numbers=noenddot,
            abstracton,
            bibliography=totoc,
            index=totoc
         ]{scrartcl}
```

These packages are basic low-level things that we use to declare commands, work with strings and so on.

```
\RequirePackage{etoolbox,xstring,xparse,atbegshi↙
   ,kvoptions}
```

Now, higher-level packages we use in our definitions.

```
\RequirePackage{verbatim,marginnote,calc,↙
   hyperref,multicol,hologo}
```

```
\RequirePackage[nomain,xindy,numberedsection,order=letter,
                sanitize={description=false,sort=false}]{glossaries}
```

We also include the ydoc packages we'll be extending.

```
\RequirePackage{ydoc-code,ydoc-desc}
```

The rest is basically just styling.

```
\RequirePackage[british]{babel}
\RequirePackage[babel]{microtype}
\RequirePackage{scrpage2,PTSerif,ascii}
\RequirePackage[defaultsans,osfigures,scale=0.95]{opensans}
```

3.2 Error messages

Set up some error message texts for later use.

```
\msg_new:nnn{skdoc}{key-exists}{File~key~"#1"~already~declared!}
\msg_new:nnn{skdoc}{key-nexists}{File~key~"#1"~hasn't~been~declared!}
\msg_new:nnn{skdoc}{wrote-file}{Writing~things~to~file~"#1".}
\msg_new:nnn{skdoc}{read-preamble}{Reading~preamble~from~file~"#1".}
```

3.3 Options

Define the (as of version 1.0) only option, and process it.

```
\SetupKeyvalOptions{
    family=skdoc,
    prefix=skdoc@
}
```

```
\DeclareStringOption{load}[\jobname]
\ProcessKeyvalOptions*
```

If the option was used, load the given package if it exists.

```
\IfStrEq{\skdoc@load}{}{}{%
    \IfFileExists{\skdoc@load.sty}{%
        \RequirePackage{\skdoc@load}
    }{}
}
```

3.4 The `MacroCode` environment

We need a token list and input/output.

```
\tl_new:N\skdoc@temptl
\ior_new:N\skdoc@input
\iow_new:N\skdoc@output
```

`\DeclareFile` `#1`: A list of key-value parameters
`#2`: Filename of the file being declared

This declares a file as part of the bundle, which means we will be writing things to it.

```
\DeclareDocumentCommand\DeclareFile{om}{
    \group_begin:
    \keys_define:nn{skdoc@declarefile}{%
        preamble .value_required:,
        preamble .code:n = \edef\skdoc@preamble↙
           {##1},
        key .value_required:,
        key .code:n = \def\skdoc@key{##1}
    }%
    \def\skdoc@preamble{}%
    \def\skdoc@key{#2}%
    \IfNoValueTF{#1}{}{\keys_set:nn{↙
       skdoc@declarefile}{#1}}
```

```
46      \tl_new:c{skdoc@output@\skdoc@key}
47      \int_if_exist:cTF{skdoc@output@\skdoc@key
           @line}{
48          \msg_critical:nnx{skdoc}{key-exists}{\
               skdoc@key}
49      }{
50          \int_zero_new:c{skdoc@output@\skdoc@key
               @line}
51      }
52      \IfStrEq{\skdoc@preamble}{}{}{
53          \tl_set:Nx\l_tmpa_tl{\skdoc@preamble}
54          \edef\skdoc@temp{\tl_head:N\l_tmpa_tl}
55          \def\skdoc@preamble@extra{
56              \skdoc@temp\skdoc@temp\space~This~is
                   ~file~`#2',~generated~from~`\c_
                   job_name_tl.tex'~(key~`\skdoc@key
                   ').
57          }
58      }
59      \expandafter\xdef\csname skdoc@write@#2\
           endcsname{%
60          \noexpand\msg_log:nnn{skdoc}{wrote-file
               }{#2}
61          \noexpand\iow_open:Nn\noexpand\
               skdoc@output{#2}
62          \noexpand\IfStrEq{\skdoc@preamble}{}{}{
63              \noexpand\iow_now:Nx\noexpand\
                   skdoc@output{\
                   skdoc@preamble@extra}
64              \noexpand\iow_now:Nx\noexpand\
                   skdoc@output{\skdoc@preamble}
65          }
66          \noexpand\iow_now:Nx\noexpand\
               skdoc@output{\noexpand\tl_to_str:c{
               skdoc@output@\skdoc@key}}
67          \noexpand\iow_close:N\noexpand\
```

```
            skdoc@output
    }
    \AfterEndDocument{\csname skdoc@write@#2\
       endcsname}
    \group_end:
}
```

skdoc@verbatim `#1`: The key of the file being described

This environment does all the actual work for `MacroCode`.

```
\DeclareDocumentEnvironment{skdoc@verbatim}{m}{%
    \iftoggle{skdoc@impl}{\@bsphack}{}
    \def\skdoc@key{#1}
    \int_if_exist:cTF{skdoc@output@\skdoc@key
       @line}{}{
        \msg_critical:nnx{skdoc}{key-nexists}{\
           skdoc@key}
    }%
    \marginnote{
        \leavevmode
        \llap{
            \scriptsize\color{gray}
            $\langle$\skdoc@key$\rangle$
            \makebox[2ex]{\strut}
        }
    }
    \def\verbatim@processline{%
        \tl_gput_right:cx{skdoc@output@\
           skdoc@key}{\the\verbatim@line\iow_
           newline:}%
        \int_gincr:c{skdoc@output@\skdoc@key
           @line}%
        \iftoggle{skdoc@impl}{
            \noindent\leavevmode%
            \hspace*{-5ex}
            \begin{minipage}[c][1ex]{\textwidth}
                \makebox[4ex]{%
```

```
                \leavevmode
                \tiny\color{lightgray}\hfill%
                \int_use:c{skdoc@output@\skdoc@key @line}%
            }%
            \hspace*{1ex}%
            {
                \verbatim@font\footnotesize
                \the\verbatim@line
            }
        \end{minipage}
        \vskip-.75ex\par
    }{}
}%
\let\do\@makeother\dospecials\catcode`\^^M\active%
\iftoggle{skdoc@impl}{
    \frenchspacing\@vobeyspaces
}{}
\verbatim@start%
}{%
\iftoggle{skdoc@impl}{\@esphack}{}%
}
```

MacroCode `#1`: The key of the file being described

```
\DeclareDocumentEnvironment{MacroCode}{m}{
\iftoggle{skdoc@impl}{
    \vspace{.2\baselineskip}
    \par\noindent
}{}
\skdoc@verbatim{#1}
}{
\endskdoc@verbatim
\iftoggle{skdoc@impl}{
    \vspace{.5\baselineskip}
```

```
    }{}
}
```

3.4.1 Reading a preamble

`\PreambleTo` #1: A token to which we will save the preamble
#2: File to read the preamble from
Read preamble from a document and store in variable.

```
\DeclareDocumentCommand\PreambleTo{mm}{%
    \group_begin:
    \msg_info:nnn{skdoc}{read-preamble}{#2}
    \ior_open:Nn\skdoc@input{#2}
    \bool_until_do:nn{\ior_if_eof_p:N\skdoc@input}{%
      \tl_if_empty:NTF\skdoc@temptl{}{%
          \tl_put_right:Nx\skdoc@temptl{\iow_newline:}
      }
      \tl_clear:N\l_tmpb_tl
      \ior_get_str:NN\skdoc@input\l_tmpa_tl
      \tl_put_right:Nx\l_tmpb_tl{\tl_head:N\l_tmpa_tl}
      \IfStrEq{\tl_to_str:N\l_tmpb_tl}{\@percentchar}{%
          \tl_set_eq:NN\l_tmpb_tl\skdoc@temptl
          \tl_concat:NNN\skdoc@temptl\l_tmpb_tl\l_tmpa_tl
      }{%
          \ior_close:N\skdoc@input
      }
    }
    \xdef#1{\tl_to_str:N\skdoc@temptl}
    \group_end:
}
```

`\SelfPreambleTo` `#1`: A token to which we will save the preamble
Shorthand to read preamble from current document.

```
\DeclareDocumentCommand\SelfPreambleTo{m}{%
    \PreambleTo{#1}{\c_job_name_tl}%
}
```

3.5 Styling

3.5.1 Colors

First, we redefine a couple of colors from ydoc as well as defining a couple for ourselves.

```
\providecolorset{RGB}{}{}{
    section,11,72,107;
    extlink,73,10,61;
    intlink,140,35,24;
    sharp,250,105,0;
    bright,198,229,217;
    macrodesc,73,10,61;
    keydesc,140,35,24;
    macroimpl,73,10,61;
    meta,11,72,107;
    scriptcolor,140,35,24;
    optioncolor,250,105,0;
    opt,250,205,0
}
```

3.5.2 Fonts

Then we redefine a couple of the KOMA-script font commands to use our newly defined colors, along with other fixes.

```
\RenewDocumentCommand\descfont{}{\sffamily\↙
   fontseries{sb}}
```

```
\RenewDocumentCommand\sectfont{}{\sffamily\↙
   fontseries{sb}}
\addtokomafont{minisec}{\bfseries}
\addtokomafont{paragraph}{\color{section}}
\addtokomafont{section}{\color{section}}
\addtokomafont{subsection}{\color{section}}
\addtokomafont{subsubsection}{\color{section}}
\addtokomafont{descriptionlabel}{\color{section↙
   }}
\addtokomafont{sectionentry}{\rmfamily\bfseries}
\addtokomafont{sectionentrypagenumber}{\rmfamily↙
   \bfseries}
```

3.5.3 Configuring hyperref

Finally, we set up hyperref to also use our colors.

```
\hypersetup{
    colorlinks=true,
    linkcolor=intlink,
    anchorcolor=intlink,
    citecolor=black,
    urlcolor=extlink
}
```

3.6 Documentation macros

We can now start defining the documentation macros.

3.6.1 Inline referencing

We introduce a couple of macros for referencing various constructs in running text, *i.e.* `\cs`-like macros.

`\cs` `#1`: The macro name to be typeset
The `\cs` macro typesets a macro.

```
%\DeclareDocumentCommand\cs{m}{\texttt{\
   textbackslash #1}}
```

`\env` `#1`: The environment name to be typeset
The `\env` macro typesets an environment.

```
%\DeclareDocumentCommand\env{m}{\texttt{#1}}
```

`\pkg` `#1`: The package name to be typeset
The `\pkg` macro typesets a package.

```
\DeclareDocumentCommand\pkg{m}{\textsf{#1}}
```

`\opt` `#1`: The option name to be typeset
The `\opt` macro typesets an option

```
\DeclareDocumentCommand\opt{m}{\texttt{#1}}
```

`\bib` `#1`: The BibTeX entry type name to be typeset
The `\bib` macro typesets a BibTeX entry type.

```
\DeclareDocumentCommand\bib{m}{\texttt{@#1}}
```

`\thm` `#1`: The theme name to be typeset
The `\thm` macro typesets a theme of some sort.

```
\DeclareDocumentCommand\thm{m}{\textrm{#1}}
```

3.6.2 Descriptional macros

A range of descriptional macros are made available by the ydoc package, but we need to redefine and extend them.
We begin with extending.

`\Describe@Macro` `#1`: The macro name, including leading backslash
The `\Describe@Macro` macro is changed to typeset its agument in a `\marginnote` instead of an `\fbox`.

```
\def\Describe@Macro#1{%
    \endgroup
    \edef\name{\expandafter\@gobble\string#1}%
    \global\@namedef{href@desc@\name}{}%
    \immediate\write\@mainaux{%
        \global\noexpand\@namedef{href@desc@\name}{}%
    }%
    \hbox\y@bgroup
    \@ifundefined{href@impl@\name}{}{\hyperlink{impl:\name}}%
    {%
        \hbox{
            \vbox to 0pt{
                \vss\hbox{
                    \raisebox{4ex}{\hypertarget{desc:\name}{}
                }
            }
        }%
        \marginnote{\llap{\PrintMacroName{#1}}}
        }%
    }%
    \ydoc@macrocatcodes
    \macroargsstyle
    \read@Macro@arg
}
```

`\descframe` `#1`: Contents to be framed

Similarly, `\descframe` is changed to produce an `\mbox` instead of an `\fbox`.

```
\def\descframe#1{%
    \mbox{\hspace*{1.5\descsep}#1\hspace*{2\descsep}}%
}
```

`\PrintMacroName` `#1`: Macro name to be printed

`\PrintMacroName` is hooked to also index the macro name while printing it.

```
\let\old@PrintMacroName\PrintMacroName
\DeclareDocumentCommand\PrintMacroName{m}{%
    \index@macro{#1}
    \old@PrintMacroName{#1}
}
```

`\PrintEnvName` `#1`: Either `\end` or `\begin`
`#2`: Name of the environment to be printed

Similarly to `\PrintMacroName`, the `\PrintEnvName` is hooked to index the environment when printing the `\begin` part of the environment.

```
\let\old@PrintEnvName\PrintEnvName
\def\PrintEnvName#1#2{
    \ifx#1\begin
        \edef\skdoc@temp{\noexpand\↙
           index@environment{#2}}
        \skdoc@temp
    \fi
    \old@PrintEnvName{#1}{#2}
}
```

Then we add a few of our own. For instance, we add macros to typeset descriptions of options. We also undefine the `\optpar` macro defined by ydoc, since we supersede it with `\Option`.

```
\let\optpar\relax
```

`\Options` `#1`: A comma-separated list of options

```
\DeclareDocumentCommand\Options{m}{
    \clist_set:Nn\l_tmpa_clist{#1}
    \marginnote{
        \clist_map_inline:Nn\l_tmpa_clist{
```

```
            \index@option{####1}
            \hfill
            \llap{\textcolor{opt}{\opt{####1}}}
            \mbox{}\\
        }
    }
    \nobreak
}
```

`\Option` `#1`: And option

```
\DeclareDocumentCommand\Option{m}{
    \Options{#1}
}

\def\skdoc@WithValues@peek{
    \ifx\@let@token\AndDefault\else\par\nobreak\↙
       fi
}
```

`\WithValues` `#1`: Values of a key-value option

The `\WithValues` macro peeks ahead to see if there's an `\AndDefault` macro further down. It typesets the values of a key-vaue option

```
\DeclareDocumentCommand\WithValues{m}{
    \noindent\makebox[\linewidth][l]{\texttt{\↙
       textcolor{gray}{#1}}}
    \futurelet\@let@token\skdoc@WithValues@peek
}
```

`\AndDefault` `#1`: The value of a key-value option

Typesets the default value of a key-value option. To the far right of the line.

```
\DeclareDocumentCommand\AndDefault{m}{
    \llap{\textcolor{gray}{\texttt{(#1)}}}\par\↙
       nobreak
```

```
}
```

Lastly, we define an envionment for showing examples. It's rediculously simple, really.

`example` (no arguments)

```
\DeclareDocumentEnvironment{example}{}{
    \minisec{Example:}
}{}
```

3.6.3 Implementation environment

We define environments that encase the implementation of macros, environments, options, BibTeX entry types and themes. Watch out—there's a lot of duplicate code here.

`macro` #1: True if this is the starred variant
#2: Name of the macro being implemented
#3: If given, the number of arguments that `\macro@impl@args` will read

```
\DeclareDocumentEnvironment{macro}{smo}{%
    \@bsphack
    \index@macro*{#2}
    \@macroname{#2}%
    \IfBooleanTF{#1}{}{
        \PrintMacroImpl{#2}
        \IfNoValueTF{#3}{
            \macro@impl@argline@noarg{(no~arguments)}
        }{\macro@impl@args[#3]}
    }%
}{
    \let\skdoc@macroname@key\@empty
    \@esphack
}
```

`environment` #1: True if this is the starred variant
#2: Name of the environment being implemented

#3: If given, the number of arguments that \macro@impl@args will read

```
\DeclareDocumentEnvironment{environment}{smo}{%
    \@bsphack
    \index@environment*{#2}
    \@environmentname{#2}%
    \IfBooleanTF{#1}{}{
        \PrintEnvImplName{#2}
        \IfNoValueTF{#3}{
            \macro@impl@argline@noarg{(no~arguments)}
        }{\macro@impl@args[#3]}
    }%
}{
    \let\skdoc@macroname@key\@empty
    \@esphack
}
```

option

#1: True if this is the starred variant
#2: Name of the option being implemented
#3: Values this key-value option can take

```
\DeclareDocumentEnvironment{option}{smg}{%
    \@bsphack
    \index@option*{#2}
    \@optionname{#2}%
    \IfBooleanTF{#1}{}{
        \PrintEnvImplName{#2}
        \IfNoValueTF{#3}{
            \macro@impl@argline@noarg{(option)}
        }{
            \macro@impl@argline@noarg{
                Option~with~values~\texttt{\textcolor{gray}{#3}}
            }
        }
    }%
}{
```

```
    \let\skdoc@macroname@key\@empty
    \@esphack
}
```

bibentry

#1: True if this is the starred variant
#2: Name of the BibTeX entry type being implemented

```
\DeclareDocumentEnvironment{bibentry}{sm}{%
    \@bsphack
    \index@bibentry*{#2}
    \@bibentryname{#2}%
    \IfBooleanTF{#1}{}{
        \PrintEnvImplName{#2}
        \macro@impl@argline@noarg{(\hologoRobust↙
           {BibTeX}~entry~type)}
    }%
}{
    \let\skdoc@macroname@key\@empty
    \@esphack
}
```

theme

#1: True if this is the starred variant
#2: Name of the theme being implemented

```
\DeclareDocumentEnvironment{theme}{sm}{%
    \@bsphack
    \index@theme*{#2}
    \@themename{#2}%
    \IfBooleanTF{#1}{}{
        \PrintEnvImplName{#2}
        \macro@impl@argline@noarg{(theme)}
    }%
}{
    \let\skdoc@macroname@key\@empty
    \@esphack
}
```

We also provide starred variants of the environments, which will add the implementation to the index but not print anything.

`macro*` `#1`: Name of the macro being implemented
`#2`: If given, the number of arguments that `\macro@impl@args` will read

```
\DeclareDocumentEnvironment{macro*}{mo}%
    {\begin{macro}*{#1}[#2]}{\end{macro}}
```

`environment*` `#1`: Name of the environment being implemented
`#2`: If given, the number of arguments that `\macro@impl@args` will read

```
\DeclareDocumentEnvironment{environment*}{mo}%
    {\begin{environment}*{#1}[#2]}{\end{↙
       environment}}
```

`option*` `#1`: Name of the option being implemented
`#2`: Values this key-value option can take

```
\DeclareDocumentEnvironment{option*}{mg}%
    {\begin{option}*{#1}{#2}}{\end{option}}
```

`bibentry*` `#1`: Name of the BibTeX entry type being implemented

```
\DeclareDocumentEnvironment{bibentry*}{m}%
    {\begin{bibentry}*{#1}}{\end{bibentry}}
```

`theme*` `#1`: Name of the theme being implemented

```
\DeclareDocumentEnvironment{theme*}{m}%
    {\begin{theme}*{#1}}{\end{theme}}
```

And finally, we redefine some of the underlying ydoc macros to behave the way we want them to. For instance, we redefine the commands that print environment and macro implementation names so that they typeset the name i a `\marginnote` rather than in an `\fbox`.

`\PrintEnvImplName` `#1`: The environment name to be printed

```
\def\PrintEnvImplName#1{%
    \par\mbox{}
    \marginnote{\llap{\implstyle{#1}}}
    \par
}
```

\PrintMacroImpl `#1`: The macro name to be printed

```
\def\PrintMacroImpl#1{%
    \par
    \hbox{%
        \edef\name{\expandafter\@gobble\string#1}%
        \global\@namedef{href@impl@\name}{}%
        \immediate\write\@mainaux{%
            \global\noexpand\@namedef{href@impl@\name}{}%
        }%
        \raisebox{4ex}[4ex]{\hypertarget{impl:\name}{}}%
        \@ifundefined{href@desc@\name}{}{\hyperlink{desc:\name}}%
        \marginnote{\llap{\PrintMacroImplName{#1}}}%
    }%
    \par
}
```

We also redefine the utility macros belonging to `\macro@impl@arg`. The argument number Description of the argument

```
\def\macro@impl@argline#1#2{%
    \par\noindent{\texttt{\##1}:~#2\strut}%
}
```

The number of arguments to read

```
\def\macro@impl@args[#1]{%
    \vspace*{-\baselineskip}
    \begingroup
    \let\macro@impl@argcnt\@tempcnta
    \let\macro@impl@curarg\@tempcntb
    \macro@impl@argcnt=#1\relax
    \macro@impl@curarg=0\relax
```

```
    \ifnum\macro@impl@curarg<\macro@impl@argcnt\relax
        \expandafter\macro@impl@arg
    \else
        \expandafter\macro@impl@endargs
    \fi
}

\def\macro@impl@endargs{
    \endgroup\par
}
```

The line to print instead of an argument line This last macro is a replacement used when there are no arguments or if the implementation is an option or something like that. It behaves pretty much like `\macro@impl@args`, but with only one argument to read.

```
\def\macro@impl@argline@noarg#1{%
    \vspace*{-\baselineskip}
    \par\noindent{#1\strut}\par\medskip%
}
```

3.7 The index

The index is based on glossaries, and as such the whole process of adding entries to the index is reduced to adding glossary entries. This is done through two wrapper macros around the `\newglossaryentry` macro.

`\@index@` `#1`: The key of the index entry

The text of the index entry What `\@index@` does is to decide wether we are hiding the implementation part of the documentation (discussed later), and wether we are in the actual implementation or not. If we are in the implementation and aren't printing it, we shouldn't add an index entry.

```
\DeclareDocumentCommand\@index@{mm}{
    \iftoggle{skdoc@impl}{
```

```
            \@index@@{#1}{#2}
        }{
            \iftoggle{skdoc@in@impl}{}{
                \@index@@{#1}{#2}
            }
        }
}
```

`\@index@@` #1: The key of the index entry
#2: The text of the index entry
This macro does the actual adding to the glossary.

```
\DeclareDocumentCommand\@index@@{mm}{
    \newglossaryentry{index-#1}{
        type=index,
        name={#2},
        description={\nopostdesc},
        sort={#1}
    }
}
```

3.7.1 Adding index entries

These macros add an index entry with different contents depending on the thing (macro, environment, etc.) that is being indexed. They all have starred variants which are used by the implementation environments, and non-starred variants used by the description macros (the star affects the style of the page number). Each environment first test wether the given entry key exists, and defines a new entry if it doesn't. Then, a usage of the entry is recorded.

`\index@macro` #1: True if this is the starred variant
#2: The name of the macro being indexed, including backslash

```
\DeclareDocumentCommand\index@macro{sm}{
    \def\skdoc@temp{\expandafter\@gobble\string↙
       #2}
```

```
    \ifglsentryexists{index-\skdoc@temp}{}{
        \@index@{\expandafter\@gobble\string#2}
                {\cs{\expandafter\@gobble\string#2}}
    }
    \IfBooleanTF{#1}{
        \glsadd[types=index]{index-\skdoc@temp}
    }{
        \glsadd[types=index,format=hyperit]{index-\skdoc@temp}
    }
}
```

`\index@environment` #1: True if this is the starred variant
#2: The name of the environment being indexed

```
\DeclareDocumentCommand\index@environment{sm}{
    \def\skdoc@temp{\string#2}
    \ifglsentryexists{index-\skdoc@temp}{}{
        \@index@{\string#2}
                {\env{\string#2}~(environment)}
    }
    \IfBooleanTF{#1}{
        \glsadd[types=index]{index-\skdoc@temp}
    }{
        \glsadd[types=index,format=hyperit]{index-\skdoc@temp}
    }
}
```

`\index@option` #1: True if this is the starred variant
#2: The name of the option being indexed

```
\DeclareDocumentCommand\index@option{sm}{
    \def\skdoc@temp{\string#2}
    \ifglsentryexists{index-\skdoc@temp}{}{
```

```
        \@index@{\string#2}
                {\opt{\string#2}~(option)}
    }
    \IfBooleanTF{#1}{
        \glsadd[types=index]{index-\skdoc@temp}
    }{
        \glsadd[types=index,format=hyperit]{index-\skdoc@temp}
    }
}
```

`\index@bibentry` `#1`: True if this is the starred variant
`#2`: The name of the BibTeX entry type being indexed, including initial `@` sign

```
\DeclareDocumentCommand\index@bibentry{sm}{
    \def\skdoc@temp{\expandafter\@gobble\string#2}
    \ifglsentryexists{index-\skdoc@temp}{}{
        \@index@{\expandafter\@gobble\string#2}
                {\bib{\expandafter\@gobble\string#2}~(\hologoRobust{BibTeX}~entry~type)}
    }
    \IfBooleanTF{#1}{
        \glsadd[types=index]{index-\skdoc@temp}
    }{
        \glsadd[types=index,format=hyperit]{index-\skdoc@temp}
    }
}
```

`\index@theme` `#1`: True if this is the starred variant
`#2`: The name of the theme being indexed

```
\DeclareDocumentCommand\index@theme{sm}{
```

```
    \def\skdoc@temp{\string#2}
    \ifglsentryexists{index-\skdoc@temp}{}{
        \@index@{\string#2}
                {\thm{\string#2}~(theme)}
    }
    \IfBooleanTF{#1}{
        \glsadd[types=index]{index-\skdoc@temp}
    }{
        \glsadd[types=index,format=hyperit]{index-\skdoc@temp}
    }
}
```

3.7.2 Displaying the index

Displaying the index is very simple. We begin by defining our own glossaries style.

```
\newglossarystyle{docindex}{
    \glossarystyle{indexgroup}
    \renewcommand*{\glspostdescription}{\unskip\leaders\hbox to 2.9mm{\hss.}\hfill\strut}
    \renewenvironment{theglossary}{
        \begin{multicols}{2}
        \setlength{\parindent}{0pt}
        \setlength{\parskip}{0pt plus 0.3pt}
        \let\item\@idxitem
    }{
        \end{multicols}
    }
    \renewcommand*{\glossaryentryfield}[5]{
        \item\glsentryitem{##1}\glstarget{##1}{##2}
            \ifx\relax##4\relax\else\space(##4)\fi
            ##3\glspostdescription\space ##5}
```

```
    \renewcommand*{\glsgroupheading}[1]{
        \IfStrEq{##1}{default}{
            \item{\descfont\glssymbolsgroupname}
        }{
            \item{\descfont\glsgetgrouptitle{##1}}
        }
    }
    \renewcommand*{\glsgroupskip}{
        \par\vspace{15\p@}\relax
    }
}
```

We follow that up by defining the actual glossay, and making sure to run `\makeglossaries` when the preamble is complete.

```
\newglossary{index}{ids}{ido}{Index}
\AtBeginDocument{\makeglossaries}
```

`\PrintIndex` (no arguments)

Finally, we define a command `\PrintIndex` that prints the index. Note the preamble that describes how the index page numbers work.

```
\providecommand*\PrintIndex{%
    \begingroup
    \renewcommand*{\glossarypreamble}{
        Numbers~written~in~italic~refer~to~the~page~where~the~
        corresponding~entry~is~described;~numbers~in~roman~
        refer~to~the~code~line~of~the~definition.\par
    }
    \printglossary[type=index,style=docindex]
    \endgroup
}
```

3.8 The changelog

The changelog is implemented as a glossary using the glossaries package. We begin by defining a name for general changes, and commands that save the name of the current macro, environment or similar for use by the \changes macro.

\generalname (no arguments)

```
\DeclareDocumentCommand\generalname{}{General}
```

Name of the macro being described

```
\DeclareDocumentCommand\@macroname{m}{
    \def\skdoc@macroname@stylized{\cs{\expandafter\@gobble\string#1}}
    \def\skdoc@macroname@key{macro-\expandafter\@gobble\string#1}
}
```

Name of the environment being described

```
\DeclareDocumentCommand\@environmentname{m}{
    \def\skdoc@macroname@stylized{\env{\string#1}}
    \def\skdoc@macroname@key{env-#1}
}
```

Name of the option being described

```
\DeclareDocumentCommand\@optionname{m}{
    \def\skdoc@macroname@stylized{\opt{\string#1}}
    \def\skdoc@macroname@key{opt-#1}
}
```

Name of the BibTeX entry being described

```
\DeclareDocumentCommand\@bibentryname{m}{
    \def\skdoc@macroname@stylized{\bib{\expandafter\@gobble\string#1}}
```

```
    \def\skdoc@macroname@key{bibentry-\
      expandafter\@gobble\string#1}
}
```

Name of the theme being described

```
\DeclareDocumentCommand\@themename{m}{
    \def\skdoc@macroname@stylized{\thm{\string
      #1}}
    \def\skdoc@macroname@key{thm-#1}
}
```

Along with these we also define the variables they affect as empty.

```
\def\skdoc@macroname@stylized{}
\let\skdoc@macroname@key\@empty
```

3.8.1 Adding changes

Since the changelog is based on glossaries, adding changes amounts to simply adding a glossary entry.

`\changes` `#1`: The version in which the changes were made
`#2`: A short description of the changes

```
\DeclareDocumentCommand\changes{mm}{%
    \@bsphack
    \ifglsentryexists{#1}{}{
        \newglossaryentry{#1}{
            type=changes,
            name={v#1},
            description={\nopostdesc},
            nonumberlist=true
        }
    }
    \ifx\skdoc@macroname@key\@empty
        \newglossaryentry{#1-general}{
            type=changes,
```

```
                description={\generalname{}:~#2},
                parent={#1},
                sort={0},
                nonumberlist=true
            }
            \glsadd[types=changes]{#1-general}
        \else
            \newglossaryentry{#1-\↙
               skdoc@macroname@key}{
                type=changes,
                description={\↙
                   skdoc@macroname@stylized{}:~#2},
                parent={#1},
                sort={\skdoc@macroname@key},
                nonumberlist=true
            }
            \glsadd[types=changes]{#1-\↙
               skdoc@macroname@key}
        \fi
        \@esphack
}
```

3.8.2 Displaying the changelog

Displaying the changelog is equally simple. We begin by defining our own glossaries style.

```
\newglossarystyle{changelog}{
    \glossarystyle{altlist}
    \renewenvironment{theglossary}{
        \begin{multicols}{2}\begin{description}
    }{
        \end{description}\end{multicols}
    }
    \renewcommand*{\glossaryentryfield}[5]{
        \par\vspace{5\p@}\relax
```

```
        \item[\glsentryitem{##1}\glstarget↙
           {##1}{##2}]
                    \mbox{}\par\nobreak\↙
                       @afterheading
    }
    \renewcommand{\glossarysubentryfield}[6]{%
        \par\hspace*{\itemindent}
        \glssubentryitem{##2}%
        \glstarget{##2}{\strut}##4\↙
           glspostdescription\space ##6
    }
}
```

We follow that up by defining the actual glossay, and making sure to run `\makeglossaries` when the preamble is complete.

```
\newglossary{changes}{gls}{glo}{Changes}
\AtBeginDocument{\makeglossaries}
```

`\PrintChanges` (no arguments)

Finally, we define a command `\PrintChanges` that prints the list of changes.

```
\DeclareDocumentCommand\PrintChanges{}{%
    \begingroup
    \printglossary[type=changes,style=changelog]
    \endgroup
}
```

3.9 Hiding the implementation

We define commands to hide the implementation from the documentation. Here, the "implementation" is understood to be everything between the `\Implementation` and `\Finale` macros. What we do is disable and/or reset page and section counters for the duration of the implementation, and set a shipout hook that simply discards the pages while we are in

the implementation. A consquence of this is that we must force a page break between what's before the implementation and what's after, which might look horrible.

Anyway, we define two toggles that keep track of things. One keeps track of wether to (not) hide the implementation, and one keeps track of wether we are in the implementation or not. These are provided with sensible defaults (*i.e.* true and false, respectively). We also define a counter in which we save the page number we had when the implementation started.

```
\newtoggle{skdoc@impl}
\newtoggle{skdoc@in@impl}
\toggletrue{skdoc@impl}
\togglefalse{skdoc@in@impl}
\newcounter{skdoc@impl@page}
```

Then we define the shipout hook. Fairly straight-forward.

```
\AtBeginShipout{
    \iftoggle{skdoc@impl}{}{
        \iftoggle{skdoc@in@impl}{\↙
           AtBeginShipoutDiscard}{}
    }
}
```

`\OnlyDescription` (no arguments)

The `\OnlyDescription` macro simply toggles the appropriate toggle.

```
\DeclareDocumentCommand\OnlyDescription{}{\↙
  togglefalse{skdoc@impl}}
```

`\Implementation` (no arguments)

The `\Implementation` macro defines all the sectioning commands to be empty (saving the old ones), clears the page, saves the page number and toggles the appropriate toggle.

```
\DeclareDocumentCommand\Implementation{}{
    \iftoggle{skdoc@impl}{}{
        \clearpage
        \toggletrue{skdoc@in@impl}
        \let\skdoc@old@part\part
        \DeclareDocumentCommand\part{som}{}
        \let\skdoc@old@section\section
        \DeclareDocumentCommand\section{som}{}
        \let\skdoc@old@subsection\subsection
        \DeclareDocumentCommand\subsection{som}{}
        \let\skdoc@old@subsubsection\subsubsection
        \DeclareDocumentCommand\subsubsection{som}{}
        \let\skdoc@old@paragraph\paragraph
        \DeclareDocumentCommand\paragraph{som}{}
        \let\skdoc@old@subparagraph\subparagraph
        \DeclareDocumentCommand\subparagraph{som}{}
        \setcounter{skdoc@impl@page}{\value{page}}
    }
}
```

`\Finale` (no arguments)

The `\Finale` macro basically just undoes what the `\Implementation` macro did.

```
\DeclareDocumentCommand\Finale{}{
    \iftoggle{skdoc@impl}{}{
        \clearpage
        \togglefalse{skdoc@in@impl}
        \let\part\skdoc@old@part
        \let\section\skdoc@old@section
        \let\subsection\skdoc@old@subsection
```

```
610             \let\subsubsection\↙
                   skdoc@old@subsubsection
611             \let\paragraph\skdoc@old@paragraph
612             \let\subparagraph\skdoc@old@subparagraph
613             \setcounter{page}{\value{skdoc@impl@page↙
                   }}
614         }
615 }
```

3.10 Document metadata

3.10.1 Setting metadata

We override a bunch of the general titlepage macros and add a few of our own. First, we initialize the underlying variables.

```
616 \let\@ctan\@empty%
617 \let\@repository\@empty%
618 \let\@plainemail\@empty%
619 \let\@email\@empty%
620 \let\@version\@empty%
```

Then, we define the actual macros.

`\package` `#1`: A list of key-value options
`#2`: The package name

The `\package` macro sets the package name of the documentation. The key-value options are `vcs` and `ctan`.

```
621 \DeclareDocumentCommand\package{om}{%
622     \keys_define:nn{skdoc@package}{%
623         vcs .value_required:,%
624         vcs .code:n = \repository{##1},%
625         ctan .code:n = \ctan{##1},%
626         ctan .default:n = #2%
627     }%
628     \IfNoValueTF{#1}{}{\keys_set:nn{↙
           skdoc@package}{#1}}%
```

```
    \def\@package{#2}%
    \title{The~\textsf{\textbf{\@package}}~package}%
}
```

`\ctan` `#1`: The name of a package or bundle on CTAN

```
\DeclareDocumentCommand\ctan{m}{%
    \def\@ctan{\url{http://www.ctan.org/pkg/#1}}%
}
```

`\repository` `#1`: The URI of an online repository

```
\DeclareDocumentCommand\repository{m}{%
    \def\@repository{\url{#1}}%
}
```

`\email` `#1`: The email address of the author

```
\DeclareDocumentCommand\email{m}{%
    \def\@plainemail{#1}%
    \def\@email{\href{mailto:\@plainemail}{\@plainemail}}%
}
```

`\version` `#1`: The version of the package, with no leading “v”

```
\DeclareDocumentCommand\version{m}{%
    \def\@version{#1}%
}
```

Finally, we set the default package name to `\jobname`.

```
\package{\jobname}
```

3.10.2 Using metadata

We define two macros that read useful metadata; `\theversion` and `\thepackage`. These are used internally by `\maketitle`.

`\theversion` (no arguments)

```
\DeclareDocumentCommand\theversion{}{v\@version}
```

`\thepackage` (no arguments)

```
\DeclareDocumentCommand\thepackage{}{\pkg{\@package}}
```

Additionally we define `\skdocpdfsettings`, which is also used by `\maketitle`, to include PDF metadata if the documentation is being compiled using pdfLaTeX.

```
\ifpdf
    \def\skdocpdfsettings{%
        \hypersetup{%
            pdfauthor   = {\@author\space<\@plainemail>},
            pdftitle    = {\@title},
            pdfsubject  = {Documentation~of~LaTeX~package~\@package},
            pdfkeywords = {\@package,~LaTeX,~TeX}
        }%
    }%
\else
    \let\skdocpdfsettings\empty%
\fi
```

3.11 General document commands

Most of the general document commands are defined by the scrartcl document class we base ourselves on, but a few of them have to be redefined.

3.11.1 The title page

The title page consists of the `\maketitle` and the `abstract`. We redefine both, inspired slightly by the PracTEX journal and the skrapport document class.

`\@maketitle` (no arguments)

```
\def\@maketitle{%
    \newpage
    \null
    \begin{flushleft}%
    {%
        \Huge\sectfont\@title%
        \ifx\@ctan\@empty\else%
            \footnote{Available~on~\@ctan.}%
        \fi
        \ifx\@repository\@empty\else%
            \footnote{Development~version~↙
               available~on~\@repository.}%
        \fi%
        \par%
    }%
    \vskip 1em
    {%
        \Large\@author
        \ifx\@email\@empty\else%
            \space
            \newlength\skdoc@minipage@ew%
            \settowidth{\skdoc@minipage@ew}{%
```

```
                \normalsize{$\lceil${\@email}$\rfloor$}}
            \begin{minipage}[b]{\skdoc@minipage@ew}
                \normalsize{$\lceil${\@email}$\rfloor$}
            \end{minipage}\par%
        \fi%
    }%
    \ifx\@version\@empty\else
        \vskip .5em
        {%
            \large Version~\@version\par%
        }%
    \fi
    \end{flushleft}%
    \par\bigskip%
}
```

`\maketitle` (no arguments)

```
\def\maketitle{%
    \skdocpdfsettings
    \renewcommand\thefootnote{\@fnsymbol\c@footnote}
    \@maketitle
    \skdocpdfsettings
}
```

`abstract` (no arguments)

```
\DeclareDocumentEnvironment{abstract}{}{
    \newlength\skdoc@abstract@tw%
    \newlength\skdoc@abstract@aw%
    \settowidth{\skdoc@abstract@tw}{\descfont\abstractname}%
```

```
    \setlength{\skdoc@abstract@aw}{\the\textwidth-\the\skdoc@abstract@tw-2em}%
    \begin{minipage}{\textwidth}
        \begin{minipage}[t]{\skdoc@abstract@tw}%
            \begin{flushright}%
                \leavevmode\descfont\abstractname%
            \end{flushright}%
        \end{minipage}%
        \hspace{1em}%
        \begin{minipage}[t]{\skdoc@abstract@aw}%
}{
        \end{minipage}
    \end{minipage}
}
```

3.11.2 Table of contents

The table of contents are redefined to imitate the excellent table of contents of the microtype manual.

```
\let\l@section@\l@section
\def\l@section{\vskip -.75ex\l@section@}

\def\l@subsection{\vskip.35ex\penalty\@secpenalty\@dottedtocline{2}{1.5em}{2.7em}}

\def\l@subsubsection#1#2{
    \leftskip 4.2em
    \parindent 0pt
    {\let\numberline\@gobble{\small #1~[#2]}}
}

\def\l@table{\@dottedtocline{1}{0pt}{1.5em}}

\def\l@figure{\@dottedtocline{1}{0pt}{1.5em}}
```

```
\def\@pnumwidth{1.7em}
\AtEndDocument{\addtocontents{toc}{\par}}
```

`\tableofcontents` (no arguments)

```
\let\old@tableofcontents\tableofcontents
\DeclareDocumentCommand\tableofcontents{}{
    \microtypesetup{protrusion=false}
    \old@tableofcontents
    \microtypesetup{protrusion=true}
}
```

3.12 Cosmetic changes

We perform a couple of cosmetic changes to existing features as well. First, we set a new header/footer style using the KOMA-script `\deftripstyle` macro.

```
\deftripstyle{skdoc}%
    {}{}{}%
    {\small The~\textsf{\textbf{\@package}}~↙
       package,~v\@version}{}{\small\pagemark}
\AfterBeginDocument{\pagestyle{skdoc}}
```

We also redefine the section level format to set the section numbers in the margin, much like the microtype manual.

```
\RenewDocumentCommand{\othersectionlevelsformat↙
   }{m}{%
    \makebox[0pt][r]{%
    \fontfamily{fos}\mdseries\selectfont
    \csname the#1\endcsname\enskip}%
}
```

Finally, we actually use microtype in the document class, and make sure to disable it in the verbatim environments. Set up microtype properly

```
\g@addto@macro\@verbatim{\microtypesetup{
   activate=false}}
\microtypesetup{expansion,kerning,spacing,
   tracking}
\DisableLigatures{family = tt*}
```

Oh, and we want \marginpars on the left, not on the right.

```
\AtBeginDocument{\reversemarginpar}
```

That's it, we're done!

```
\endinput
```

4 Changes

v0.9

General: Pre-release version with most of the general functionality implemented.

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers in roman refer to the code line of the definition.