The ydoc Class and Packages

Martin Scharrer
martin@scharrer-online.de
http://latex.scharrer-online.de/ydoc/
CTAN: http://tug.ctan.org/pkg/ydoc

Version 0.1alpha
2010/04/10

Abstract

This package bundle is currently under development. All functionality, settings and macro as well as file names can change in later versions and may be incomplete! It is not ready yet to be used for other packages.

The ydoc class and packages provide macros to document the functionality and implementation of LaTeX classes and packages. It is similar to the ltxdoc class with the doc package, but uses more modern features/packages by default (e.g. xcolor, hyperref, listings). However, some of the features like code indexing is not yet included.

1 Introduction

The ydoc packages allow the documentation of LaTeX packages and classes. The name stands for "*Y*et another *Doc*umentation Package" and is a pun on the fact that there are several documentation packages written by package developers to document their own packages. All these packages didn't suited the author and therefore he, take a guess, wrote his own documentation package. It (will) support(s) all macros and environments (but not necessary with full/identical features) provided by the doc package to allow the fast adaption of existing .dtx files.

This documentation uses the ydoc packages itself and therefore also acts as a live example.

1.1 ydoc Files

The ydoc bundle consists (at the moment, subject to change) of the ydoc class and the packages ydoc, ydoc-code, ydoc-desc, ydoc-expl and ydoc-doc. The ydoc class and package allow the user the freedom to use the functionality with other classes if wanted. The class will load the package. The ydoc package

loads the packages `ydoc-code`, `ydoc-desc`, `ydoc-expl` and `ydoc-doc`, which provide the functionality to document LaTeX code implementation, describe the user-level macro, include live code examples and provide replacements for the macros of the `doc` package, respectively. This packages can be loaded on their own in other kind of LaTeX documents if required.

1.2 Similar Packages

Other documentation related classes and packages are `ltxdoc`, `doc`, `dox`, `xdoc`, `gmdoc`, `pauldoc`, `hypdoc`, `codedoc`, `nicetext` and `tkz-doc`.

2 Usage

(section incomplete)

2.1 Code Documentation Environments

```
\begin{macro}{⟨macro⟩}[⟨number of arguments⟩]{⟨arg 1 description⟩}...{⟨arg n description⟩}
  ⟨macro documentation⟩
  \begin{macrocode}
    ⟨macro code⟩
  \end{macrocode}
  ...
\end{macro}
```

The implementation of macros can be documented using this environment. The actual ⟨*macro code*⟩ must be placed in a `macrocode` environment. Longer macro definition can be split using multiple `macrocode` environments with interleaved documentation texts.

The `ydoc` definition of the `macro` environment has an additional feature compare to `doc`. The arguments of the macro (`#1`, `#2`, ...) can be documented in a vertical list. The environment has an optional argument to declare the ⟨*number of arguments*⟩ the macro implementation has. The descriptions of this macro arguments are read from the next arguments of the environment. If the ⟨*number of arguments*⟩ is not given or zero (or less) no further arguments are read by the `macro` environment.

```
\begin{macrocode}
  ⟨macro code⟩
\end{macrocode}
```

This environment wraps around any TeX code and types it verbatim. The environment end is read verbatim as well and must be written on a separate line beginning with a percent followed by exactly four spaces: '`%␣␣␣␣\end{macrocode}`'.

```
\begin{environment}{⟨environment name⟩}[⟨number of arguments⟩]{⟨arg 1 description⟩}...{⟨arg n descri
  ⟨environment documentation⟩
  \begin{macrocode}
    ⟨macro code⟩
  \end{macrocode}
  ...
\end{environment}
```

This environment provides the same functionality as the `macro` environment above, but for environments instead.

2.2 Description Macros and Environments

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

The `\DescribeMacro` is used to describe macros included their arguments. It takes the to be described ⟨*\macro*⟩ as first argument (can also be enclosed in `{ }`). The macro name can include '`@`'. Any number of ⟨*macro arguments*⟩ (in a broad sense, see Table 1) following it are formatted as arguments of this macro. Any following non-argument token (normal text, macro, etc.) will make `\DescribeMacro` stop collecting arguments. For example, if a TEX group should be started using `{ }` direct after `\DescribeMacro` a `\relax` (or a similar macro) should be inserted between them, otherwise the group will be taken as mandatory argument of the described macro.

Multiple `\DescribeMacro` in a row will automatically stacked inside one framed box. If this is not wanted simply separate them with `\relax` or any other macro or token. See also the `DescribeMacros` environment below.

Examples:

`\DescribeMacro\mymacro*[<optional>]{<meta text>}` will result in `\mymacro`*`[`⟨*optional*⟩`]{`⟨*meta text*⟩`}` (inside a framed box).

The above syntax description of `\DescribeMacro` itself was typeset with `\DescribeMacro\DescribeMacro<\textbackslash macro><macro arguments>`.

Special macros with have a partner macro as end marker can be typeset like this:
`\DescribeMacro\csname<text>\AlsoMacro\endcsname`, which will result in `\csname`⟨*text*⟩`\endcsname`.

`\Macro`⟨*\macro*⟩⟨*macro arguments*⟩

This macro is like an in-text version of `\DescribeMacro`. The macro description stays as part of the surrounding text and is not placed inside a framed

box. The description can be broken between lines. This can be avoided by placing it inside a `\mbox{}`. `\Macro` is equivalent to `\MacroArgs\AlsoMacro`.

`\MacroArgs`⟨*macro arguments*⟩

This macro formats the ⟨*macro arguments*⟩ the same way as `\DescribeMacro` and `\Macro` but without a macro name. Like `\Macro` the description is placed in-text.

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

This macro can only be used inside the ⟨*macro arguments*⟩ of the above macros and typesets an additional macro as part of the syntax of the described macro. The additional macro is normally an end- or other marker of some kind. Further macro arguments may follow. Macros which are not part of the syntax but normal arguments should be written as `<\textbackslash name>` (yielding ⟨*\name*⟩) instead.

Example:

```
\Macro\@for<\textbackslash var> ':=' <list> \AlsoMacro\do {<code>}
\@for⟨\var⟩:=⟨list⟩\do{⟨code⟩}
```

```
\begin{DescribeMacros}
  \Macro⟨\name⟩⟨arguments⟩
  \Macro⟨\name⟩⟨arguments⟩
  ...
\end{DescribeMacros}
```

This environment can be used to place multiple macro description into the same framed box. The macros are described using `\Macro`, which has a slightly different definition than outside of this environment, to place the description into a `\hbox`. The environment stacks these `\hbox`es in a `\vbox`. The macros can also be placed freely using anything which produces a `\hbox`, e.g. `\hbox{\Macro\A ~~~ \Macro\B}` or using a `tabular` (see also `DescribeMacrosTab`).

```
\begin{DescribeMacrosTab}{⟨tabular column definition⟩}
  ⟨tabular content⟩
\end{DescribeMacrosTab}
```

This is a special version of the `DescribeMacros` environment which adds a tabular environment around the content. This is useful if a large set of small macros should be desribed at once. Placing them all below each other would result in a very bad page layout. The environment has one argument which is passed to `tabular` as the column definition. A '`@{}`' is added before and after

to remove any margins.

```
\begin{DescribeEnv}{⟨name⟩}⟨arguments⟩
  ⟨body content⟩ \\
  ⟨more body content⟩
\end{DescribeEnv}
```

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

The `DescribeEnv` can be used to describe environments in the same way the `\DescribeMacro` macro describes macros. Supported ⟨*arguments*⟩ are shown in Table 1. Potential ⟨*body content*⟩ can be placed between the begin and end of the environment description to explain the user what kind of material should be placed inside it. The environment also exists in macro form as `\DescribeEnv`, which allows to provide small ⟨*body content*⟩ as an optional argument. Please note that for this optional argument a `\MacroArgs` is automatically inserted, but not for the `\DescribeEnv` environment content.

The body content is placed into a indented `\hbox{}` stacked inside a `\vbox{}` also holding the environment begin and end line. The `\\` macro is redefined to create a new indented `\hbox` acting as new code line. Therefore this environment is similar to a one-column `tabular`: all macros placed into a line are only valid up to the next line end.

```
\DescribeLength⟨\name⟩{⟨default value⟩}
```

This macro can be used to describe LaTeX lengths also known as dimensions. Multiple `\DescribeLength` macros in a row will automatically be grouped.

2.3 Format Macros

```
\cs{⟨macro name⟩}        \env{⟨environment name⟩}
\pkg{⟨package name⟩}     \cls{⟨class name⟩}
```

This macros can be used to format names of macros, environments, packages and classes, respectively. At the moment they simply use `\texttt`.

```
\bslash   \percent   \braceleft   \braceright
```

This macros define expandable backslash ($\backslash_{12}$), percent char ($\%_{12}$), and left ($\{_{12}$) and right ($\}_{12}$) braces with catcode 12 (other), respectively. They should only be used with text-typer font when used in text, because other fonts might not have the correct characters. The macros must be protected when used in a moving argument.

Table 1: Supported 'arguments' for \DescribeMacro/\DescribeEnv/\MacroArgs.

Description	Syntax	Result	Macro[a]
Meta text	`<text>`	⟨*text*⟩	`\meta`
Mandatory Argument	`{<text>}`	{⟨*text*⟩}	`\marg`
—, without meta text	`{text}`	{text}	
Optional Argument	`[<text>]`	[⟨*text*⟩]	`\oarg`
—, without meta text	`[text]`	[text]	
Picture Argument	`(<text>)`	(⟨*text*⟩)	`\parg`
—, without meta text	`(text)`	(text)	
Beamer Overlay Argument	`<<text>>`	<⟨*text*⟩>	`\aarg`
—, without meta text	`'<'text'>'`	<text'¿'	
Star	`*`	*	
Verbatim content	`'$&^%_#$\'`	`$&^%_#$\`	
—, produce ' char	`''`	'	
Insert any TEX code	`!\fbox{T}!`	T	
Unbreakable Space	`~`		
Space (explicit macro)	`\space`		
Second macro (e.g. endmarker)	`\AlsoMacro\macro`	`\macro`	

[a]) As alternative to be used inside normal text.

`\meta{`⟨*meta text*⟩`}`	`\marg{`⟨*argument text*⟩`}`
`\oarg{`⟨*argument text*⟩`}`	`\parg{`⟨*argument text*⟩`}`
`\aarg{`⟨*argument text*⟩`}`	`\sarg`

This macros allow to typeset meta text and mandatory, optional, picture and beamer overlay arguments as well as a star symbol. They are used internally by \MacroArgs and friends. See Table 1 for examples.

`\metastyle`	`\margstyle`
`\oargstyle`	`\pargstyle`
`\aargstyle`	`\sargstyle`

This macros are used to define the style in which the corresponding macros above are being formatted. They are used like `{`⟨*\stylemacro*⟩`{`⟨*material*⟩`}}`to allow the styles to use macros like `\ttfamily` or `\texttt{`⟨*material*⟩`}`. By default the optional argument and the also optional star are printed in the color '`optional`' which is a 65% gray.

2.4 Settings

The following macro and dimensions can be redefined by the user to adjust the layout of the package documentation.

`\descindent`	(Default: `-20pt`)
`\beforedescskip`	(Default: `12pt plus 4pt minus 4pt`)
`\afterdescskip`	(Default: `6pt plus 2pt minus 2pt`)

These length define the indention and vertical distances before and after a `\Describe...` macro or environment, respectively.

`\descsep`	(Default: `1em in tt font = 10.5pt`)

This macro defines the space on the left and right side between the description text and the framed box.

2.5 Macros and Environments to include LaTeX Code Examples

```
\begin{example}
\end{example}
```

```
\begin{examplecode}(to be written)
\end{examplecode}
```

3 Implementation

3.1 Class File

```
\LoadClassWithOptions{article}
%%\RequirePackage{doc}
\RequirePackage{ydoc}
```

3.2 Package File

```
\RequirePackage{ydoc-code}
\RequirePackage{ydoc-expl}
\RequirePackage{ydoc-desc}
\RequirePackage{ydoc-doc}

\RequirePackage{newverbs}
\MakeSpecialShortVerb{\qverb}{\"}
\AtBeginDocument{\catcode`\^^A=14\relax}
```

3.3 Macros and Environments to document Implementations

3.3.1 Color and style definitions

```
\RequirePackage{xcolor}
\definecolor{macroimpl}{rgb}{0.0,0.0,0.4}
```

3.3.2 General Macros

`\ydocwrite`

```
\@ifundefined{ydocwrite}{%
  \newwrite\ydocwrite
}{}
```

`\ydocfname`

```
\@ifundefined{ydocfname}{%
  \def\ydocfname{\jobname.cod}%
}{}
```

`\ydoc@catcodes`

```
\def\ydoc@catcodes{%
  \let\do\@makeother
  \dospecials
  \catcode`\\=\active
  \catcode`\^^M=\active
  \catcode`\ =\active
}
```

3.3.3 Handling Macrocode

`macrocode`

```
\def\macrocode{%
  \par\noindent
  \begingroup
  \ydoc@catcodes
  \macro@code
}
\def\endmacrocode{}
```

`\macro@code`

#1: verbatim macro code

```
\begingroup
\endlinechar\m@ne
\@firstofone{%
\catcode`\|=0\relax
\catcode`\(=1\relax
\catcode`\)=2\relax
\catcode`\*=14\relax
\catcode`\{=12\relax
\catcode`\}=12\relax
\catcode`\ =12\relax
\catcode`\%=12\relax
\catcode`\\=\active
\catcode`\^^M=\active
\catcode`\ =\active
}*
|gdef|macro@code#1^^M%     \end{macrocode}(*
|endgroup|expandafter|macro@@code|expandafter(|↙
   ydoc@removeline#1|noexpand|lastlinemacro^^M)*
)*
```

```
|gdef|ydoc@removeline#1^^M(|noexpand|firstlinemacro)*
|gdef|ydoc@defspecialmacros(*
|def^^M(|noexpand|newlinemacro)*
|def (|noexpand|spacemacro)*
|def\(|noexpand|bslashmacro)*
)*
|gdef|ydoc@defrevspecialmacros(*
|def|newlinemacro(|noexpand^^M)*
|def|spacemacro(|noexpand )*
|def|bslashmacro(|noexpand\)*
)*
|endgroup
```

`\macro@@code`

#1: verbatim macro code

```
\def\macro@@code#1{%
  {\ydoc@defspecialmacros
  \xdef\themacrocode{#1}}%
  \PrintMacroCode
  \end{macrocode}%
}
```

`\linenumberbox`

```
\def\newlinemacro{\\\null}
\def\spacemacro{\ }
\def\bslashmacro{\char92}
\def\lastlinemacro#1{}
\def\firstlinemacro{\linenumberbox}
\def\newlinemacro{\\\linenumberbox}
\newcounter{linenumber}
\def\linenumberbox{%
  \hbox to 1.25em{}%
  \llap{%
    \stepcounter{linenumber}%
    {\footnotesize\color{gray}\thelinenumber~}%
  }%
}
```

`\PrintMacroCode`

```
\def\PrintMacroCode{%
  \begingroup
  \ttfamily
  \noindent\themacrocode
  \endgroup
}
```

\PrintMacroCode

```
\RequirePackage{listings}

\def\PrintMacroCode{%
  \begingroup
  \let\firstlinemacro\empty
  \let\lastlinemacro\empty
  \def\newlinemacro{^^J}%
  \let\bslashmacro\bslash
  \let\spacemacro\space
  \immediate\openout\ydocwrite=\ydocfname\relax
  \immediate\write\ydocwrite{\themacrocode}%
  \immediate\closeout\ydocwrite
  \@nameuse{ydoc@countbslashes}%
  \ydoclistingssettings
  \let\input\@input
  \lstinputlisting{\ydocfname}%
  \endgroup
}
```

\ydoclistingssettings

```
\def\ydoclistingssettings{%
  \lstset{%
    language=[latex]tex,basicstyle=\ttfamily,
    numbers=left,numberstyle=\tiny\color{gray},↙
       firstnumber=last,
    breaklines,prebreak={\mbox{\tiny$\swarrow$}}%
  }%
}
```

\macro@impl@args

#1: number of macro arguments

```
\def\macro@impl@args[#1]{%
  \begingroup
  \parindent=10pt\relax
  \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
}
```

`\macro@impl@endargs`

```
\def\macro@impl@endargs{%
  \endgroup
  \unskip\par\noindent\ignorespaces
}
```

`\macro@impl@argline`

#1: argument number
#2: argument description

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

`\macro@impl@arg`

#1: argument description

```
\def\macro@impl@arg#1{%
  \advance\macro@impl@curarg by\@ne\relax
  \macro@impl@argline{\the\macro@impl@curarg}{#1}%
  \ifnum\macro@impl@curarg<\macro@impl@argcnt\relax
    \expandafter\macro@impl@arg
  \else
    \expandafter\macro@impl@endargs
  \fi
}
```

`macro`

#1: implemented macro

```
\def\macro#1{%
  \PrintMacroImplName{#1}%
  \@ifnextchar[%]
    {\macro@impl@args}%
    {}%
}
\def\endmacro{}
```

environment

#1: environment name

```
\def\environment#1{%
  \PrintEnvImplName{#1}%
  \@ifnextchar[%]
    {\macro@impl@args}%
    {}%
}
\def\endenvironment{}
```

\PrintMacroImplName

#1: macro (token)

```
\def\PrintMacroImplName#1{%
  \par\bigskip\noindent
  \hbox{\hspace*{\descindent}\fbox{{\implstyle{\string#1}}}}%
  \par\medskip\noindent
}
```

\PrintEnvImplName

#1: environment name

test

```
\def\PrintEnvImplName#1{%
  \par\bigskip\noindent
  \hbox{\hspace*{\descindent}\fbox{{\implstyle{#1}}}}%
  %
  \par\medskip
}
```

\implstyle

```
\def\implstyle{\ttfamily\bfseries\color{macroimpl}}
```

`\bslash`

Defines an expandable backslash with catcode 12: '$\backslash_{12}$'. The `\@firstofone` trick is used to read the `\gdef\bslash` code before changing the catcode.

```
{%
\@firstofone{%
  \catcode`\\=12
  \gdef\bslash
}{\}
}%}
```

3.4 Provide doc macros

`\ydoc@countbslashes`

Reads the macro code into a temp box. The backslashes are defined to increase a counter.

```
\newcount\ydoc@bslashcnt
\def\ydoc@countbslashes{%
  \begingroup
    \let\firstlinemacro\empty
    \let\lastlinemacro\empty
    \let\newlinemacro\empty
    \let\spacemacro\empty
    \def\bslashmacro{\global\advance\ydoc@bslashcnt ↙
       by\@ne}%
    \setbox\@tempboxa\hbox{\themacrocode}%
  \endgroup
}
```

`\CheckSum`

```
\def\CheckSum#1{%
  \gdef\ydoc@checksum{#1}%
}
\let\ydoc@checksum\z@
```

`\AlsoImplementation`

`\OnlyDescription`

`\StopEventually`

\Finale

The first two macros modify the \StopEventually macro which either stores its argument in \Final or executes it itself.

```
\def\AlsoImplementation{%
  \gdef\StopEventually##1{%
    \@bsphack
    \gdef\Finale{##1\ydoc@checkchecksum}%
    \@esphack
  }%
}
\AlsoImplementation
\def\OnlyDescription{%
  \@bsphack
  \long\gdef\StopEventually##1{##1\endinput}%
  \@esphack
}
\let\Finale\relax
```

\MakePercentComment

\MakePercentIgnore

```
\def\MakePercentIgnore{\catcode`\%9\relax}
\def\MakePercentComment{\catcode`\%14\relax}
```

\DocInput

```
\def\DocInput#1{\MakePercentIgnore\input{#1}\
    MakePercentComment}
```

\CharacterTable

```
\providecommand*\CharacterTable[1]{%
  \PackageWarning{ydoc}{Ignoring Character Table -
     not implemented yet!}{}{}%
}
```

\DoNotIndex

```
\providecommand*\DoNotIndex[1]{%
  \PackageWarning{ydoc}{Ignoring DoNotIndex - not implemented yet!}{}{}%
}
```

\changes

```
\providecommand*\changes[3]{%
  \PackageWarning{ydoc}{Ignoring changes - not implemented yet!}{}{}%
}
```

\RecordChanges

```
\providecommand*\RecordChanges{%
  \PackageWarning{ydoc}{List of changes not implemented yet!}{}{}%
}
```

\PrintChanges

```
\providecommand*\PrintChanges{%
  \PackageWarning{ydoc}{List of changes not implemented yet!}{}{}%
}
```

\PrintIndex

```
\providecommand*\PrintIndex{%
  \PackageWarning{ydoc}{Code index not implemented yet!}{}{}%
}
```

\CodelineIndex

```
\providecommand*\CodelineIndex{%
  \PackageWarning{ydoc}{Code line index not implemented yet!}{}{}%
}
```

\EnableCrossrefs

```
\providecommand*\EnableCrossrefs{%
  \PackageWarning{ydoc}{Cross references not implemented yet!}{}{}%
}
```

`\GetFileInfo`

```
\providecommand*\GetFileInfo[1]{%
  \PackageWarning{ydoc}{Get File Info not implemented yet!}{}{}%
}
```

`\ydoc@checkchecksum`

```
\def\ydoc@checkchecksum{%
  \ifnum\ydoc@checksum=\m@ne
    \message{^^J}%
    \message{**********************^^J}%
    \message{* No Checksum found! *^^J}%
    \message{**********************^^J}%
  \else
  \ifnum\ydoc@checksum=\ydoc@bslashcnt
    \message{^^J}%
    \message{********************^^J}%
    \message{* Checksum passed *^^J}%
    \message{********************^^J}%
  \else
    \message{^^J}%
    \message{****************************^^J}%
    \message{* Checksum wrong (\ydoc@checksum<>\the\ydoc@bslashcnt) ^^J}%
    \message{****************************^^J}%
    \GenericError{Checksum wrong}{}{}{}%
  \fi
  \fi
}

\RequirePackage{shortvrb}
\MakeShortVerb{\|}
```

3.5 Description Macros and Environments

3.5.1 Color and style definitions

```
\RequirePackage{xcolor}
\definecolor{macrodesc}{rgb}{0.0,0.0,0.8}
\definecolor{macroimpl}{rgb}{0.0,0.0,0.4}
\definecolor{meta}{rgb}{0.0,0.4,0.4}
\definecolor{scriptcolor}{rgb}{0.2,0.6,0.2}
\definecolor{optioncolor}{rgb}{0.3.0.2,0}
\colorlet{optional}{black!65!white}
\colorlet{metaoptional}{optional!50!meta}
```

3.5.2 Text Formatting Macros

\meta

Prints ⟨*meta text*⟩.

```
\def\meta#1{%
  \ensuremath\langle
  {\metastyle{#1\/}}%
  \ensuremath\rangle
}
```

\@meta

Checks if `#1` is surrounded by angles `< >`. If so it calls `\is@meta` which removes the angles and calls `\meta`.

```
\def\@meta#1{%
  \@ifnextchar<%
    {\is@meta}%
    {}%
  #1%
}
```

\is@meta

Only removes the `< >` ands calls `\meta`.

```
\def\is@meta<#1>{%
  \meta{#1}%
}
```

\marg

Calls `\marg` with angles added to force meta format.

```
\def\marg#1{\@marg{<#1>}}
```

\oarg

Calls \oarg with angles added to force meta format.

```
\def\oarg#1{\@oarg{<#1>}}
```

\parg

Calls \parg with angles added to force meta format.

```
\def\parg#1{\@parg{<#1>}}
```

\aarg

Calls \aarg with angles added to force meta format.

```
\def\aarg#1{\@aarg{<#1>}}
```

\@marg

Sets style and adds braces. Text is formatted with \@meta which might add meta format.

```
\def\@marg#1{%
  {\margstyle{%
    {\ttfamily\braceleft}%
    \@meta{#1}%
    {\ttfamily\braceright}%
  }}%
}
```

\@oarg

Sets style and adds brackets. Text is formatted with \@meta which might add meta format.

```
\def\@oarg#1{%
  {\oargstyle{%
    {\ttfamily[}%
    \@meta{#1}%
    {\ttfamily]}%
  }}%
}
```

\@parg

Sets style and adds parentheses. Text is formatted with \@meta which might add meta format.

```
\def\@parg#1{%
  {\pargstyle{%
    {\ttfamily(}%
    \@meta{#1}%
    {\ttfamily)}%
  }}%
}
```

`\@aarg`

Sets style and adds angles. Text is formatted with `\@meta` which might add meta format.

```
\def\@aarg#1{%
  {\aargstyle{%
    {\ttfamily<}%
    \@meta{#1}%
    {\ttfamily>}%
  }}%
}
```

`\sarg`

Prints star with given style.

```
\def\sarg{{\sargstyle{*}}}
```

`\pkg`

`\cls`

`\env`

`\cs`

`\opt`

```
\def\pkg{\texttt}
\def\cls{\texttt}
\def\env{\texttt}
\def\cs#1{\texttt{\textbackslash #1}}
\def\opt{\textsf}
```

3.5.3 Text Formatting Styles

`\macrodescstyle`

Style of described macro names.

```
370 \def\macrodescstyle{\ttfamily\bfseries\color{↙
    macrodesc}}
```

`\macroargsstyle`

Default style for macro arguments (e.g. \MacroArgs).

```
372 \def\macroargsstyle{\ttfamily}
```

`\envcodestyle`

Default style for code body content in described environments.

```
374 \def\envcodestyle{\ttfamily}
```

`\verbstyle`

Style for verbatim text inside macro argument list.

```
376 \def\verbstyle{\ttfamily}
```

`\metastyle`

Meta text style. Because \macroargsstyle might be also active a \normalfont reset the font.

```
378 \def\metastyle{\normalfont\itshape\color{meta}}
```

`\margstyle`

Style for \marg.

```
380 \def\margstyle{}
```

`\oargstyle`

Style for \oarg. A special color is set to show the 'optional' status.

```
382 \def\oargstyle{\color{optional}\colorlet{meta}{↙
    metaoptional}}
```

\pargstyle

Style for \parg.

```
\def\pargstyle{}
```

\aargstyle

Style for \aarg.

```
\def\aargstyle{}
```

\sargstyle

Style for \sarg. A special color is set to show the 'optional' status.

```
\def\sargstyle{\ttfamily\color{optional}}
```

3.5.4 Dimension Registers

\descindent

```
\newdimen\descindent
\descindent=-\parindent
```

\beforedescskip

```
\newdimen\beforedescskip
\beforedescskip=\bigskipamount
```

\afterdescskip

```
\newdimen\afterdescskip
\afterdescskip=\medskipamount
```

\descsep

Set to 1em in tt font.

```
\newdimen\descsep
\begingroup
\ttfamily
\global\descsep=1em\relax
\endgroup
```

3.5.5 Macro Argument Reading Mechanism

`\read@Macro@arg`

Reads next token and calls second macro.

```
\def\read@Macro@arg{%
  \futurelet\@let@token\handle@Macro@arg
}
```

`\handle@Macro@arg`

Checks if next token is the begin of a valid macro argument and calls the appropriate read macro or the end macro otherwise.

```
\def\handle@Macro@arg{%
  \ifcase0%
   \ifx\@let@token\bgroup1\else
   \ifx\@let@token[\empty2\else
   \ifx\@let@token(\empty3\else
   \ifx\@let@token<\empty4\else
   \ifx\@let@token*\empty5\else
   \ifx\@let@token'\empty6\else
   \ifx\@let@token!\empty7\else
   \ifx\@let@token\@sptoken8\else
   \ifx\@let@token\space9\else
   \ifx\@let@token~9\else
   \ifx\@let@token\AlsoMacro10\else
   \ifx\@let@token\DescribeMacro11\fi
   \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
  \relax
   \unskip
   \expandafter\end@Macro@args%0
   \or\expandafter\read@Macro@marg%1
   \or\expandafter\read@Macro@oarg%2
   \or\expandafter\read@Macro@parg%3
   \or\expandafter\read@Macro@angle%4
   \or\expandafter\read@Macro@sarg%5
   \or\expandafter\read@Macro@verb%6
   \or\expandafter\read@Macro@cmds%7
   \or\expandafter\read@Macro@rmspace%8
   \or\expandafter\read@Macro@addtoken%9
  \else%10-
  \fi
}
```

`\end@Macro@args`

Closes box as calls hook. Might be locally redefined by some macros calling \read@Macro@args.

```
\def\end@Macro@args{%
  \y@egroup
  \after@Macro@args
}
```

`\after@Macro@args`

Hook to add additional commands in certain situations.

```
\def\after@Macro@args{%
}
```

Macro argument reading macros

This macros read the macro arguments and call the appropriate format macros.

`\read@Macro@marg`

```
\def\read@Macro@marg#1{%
  \@marg{#1}\read@Macro@arg
}
```

`\read@Macro@oarg`

```
\def\read@Macro@oarg[#1]{%
  \@oarg{#1}\read@Macro@arg
}
```

`\read@Macro@parg`

```
\def\read@Macro@parg(#1){%
  \@parg{#1}\read@Macro@arg
}
```

`\read@Macro@aarg`

```
\def\read@Macro@aarg<#1>>{%
  \aarg{#1}\read@Macro@arg
}
```

`\read@Macro@angle`

```
\def\read@Macro@angle<{%
  \futurelet\@let@token\read@Macro@angle@
}
```

`\read@Macro@angle@`

```
\def\read@Macro@angle@{%
  \ifx\@let@token<%
    \expandafter\read@Macro@aarg
  \else
    \expandafter\read@Macro@meta
  \fi
}
```

`\read@Macro@meta`

```
\def\read@Macro@meta#1>{%
  \meta{#1}\read@Macro@arg
}
```

`\read@Macro@sarg`

```
\def\read@Macro@sarg#1{%
  \sarg\read@Macro@arg
}
```

`\read@Macro@verb`

Sets up verbatim mode calls second macro.

```
\def\read@Macro@verb{%
  \begingroup
  \let\do\@makeother
  \dospecials
  \read@Macro@verb@
}
```

`\read@Macro@verb@`

Closes verbatim mode and formats text. If `#1` is empty (`''`) than a single `'` is printed.

```
\def\read@Macro@verb@'#1'{%
  \endgroup
  \ifx\relax#1\relax
    {\verbstyle{\string'}}%
  \else
    {\verbstyle{#1}}%
  \fi
  \read@Macro@arg
}
```

`\read@Macro@cmds`

Simply executes given code.

```
\def\read@Macro@cmds!#1!{%
  #1\relax
  \read@Macro@arg
}
```

`\read@Macro@rmspace`

Removes space. The `\@firstofone` is used to preserve the space in the macro definition.

```
\@firstofone{\def\read@Macro@rmspace} {%
  \read@Macro@arg
}
```

`\read@Macro@addtoken`

Takes token over from input to output 'stream'. This is used for `\space` and `~`.

```
\def\read@Macro@addtoken#1{%
  #1\read@Macro@arg
}
```

3.5.6 Description Macros

For Macros

`\DescribeMacro`

```
\@ifundefined{DescribeMacro}{}{%
  \PackageInfo{ydoc-desc}{Redefining \string\↙
     DescribeMacro}{}%
}
```

A \DescribeMacro places itself in a DescribeMacros environment. Multiple \DescribeMacro macros will stack themselves inside this environment. For this to work \DescribeMacros is locally defined to \y@egroup to close the \hbox from the previous \DescribeMacro.

```
\def\DescribeMacro{%
  \DescribeMacros
  \let\DescribeMacros\y@egroup
  \def\after@Macro@args{\endDescribeMacros}%
  \begingroup\makeatletter
  \Describe@Macro
}
```

\Describe@Macro

```
\def\Describe@Macro#1{%
  \endgroup
  \hbox\y@bgroup
  \PrintMacroName{#1}%
  \macroargsstyle
  \read@Macro@arg
}
```

\Macro

Simply uses the two macros below.

```
\newcommand*\Macro{\MacroArgs\AlsoMacro}
```

\@Macro

Alternative definition of \Macro inside DescribeMacros environments.

```
\def\@Macro{%
  \begingroup\makeatletter
  \Describe@Macro
}
```

\AlsoMacro

Reads argument while @ is a letter, prints the macro name and reads further arguments.

```
\newcommand*\AlsoMacro{%
  \begingroup\makeatletter
  \AlsoMacro@
}
```

```
\def\AlsoMacro@#1{%
  \endgroup
  \PrintMacroName{#1}%
  \read@Macro@arg
}
```

\MacroArgs

Uses the normal macro argument reading mechanism from \DescribeMacro. Instead of a box a simple group is added.

```
\newcommand*\MacroArgs{%
  \begingroup
  \let\end@Macro@args\endgroup
  \read@Macro@arg
}
```

\DescribeMacros

```
\def\DescribeMacros{%
  \begingroup
  \let\Macro\@Macro
  \parindent=0pt\relax
  \setbox\descbox\vbox\y@bgroup
}
```

\endDescribeMacros

```
\def\endDescribeMacros{%
  \y@egroup
  \PrintMacros
  \endgroup
}
```

\DescribeMacrosTabcolsep

```
\def\DescribeMacrosTabcolsep{\tabcolsep}
```

\DescribeMacrosTab

```
\def\DescribeMacrosTab{%
  \DescribeMacros
  \hbox\y@bgroup
  \tabcolsep=\DescribeMacrosTabcolsep\relax
  \DescribeMacrosTab@
}
\def\DescribeMacrosTab@#1{\tabular{@{}#1@{}}}
```

`\endDescribeMacrosTab`

```
\def\endDescribeMacrosTab{%
  \endtabular\y@egroup
  \endDescribeMacros
}
```

For Lengths

`\DescribeLength`

```
\newcommand*\DescribeLength{%
  \begingroup
  \let\DescribeLength\Describe@Length
  \setbox\descbox\hbox\y@bgroup
    \tabular{@{}l@{\hspace{2em}}l@{}}%
    \Describe@Length
}
```

`\Describe@Length`

```
\newcommand*\Describe@Length[2]{%
  \PrintLengthName{#1}&
  (Default: {\macroargsstyle#2\unskip})%
  \@ifnextchar\DescribeLength
    {\\}%
    {%
      \endtabular
      \y@egroup
      \PrintLength
      \endgroup
    }%
}
```

For Environments

`\DescribeEnv`

```
\@ifundefined{DescribeEnv}{}{%
  \PackageInfo{ydoc-desc}{Redefining \string\DescribeEnv}{}%
}
\let\DescribeEnv\relax

\newcommand*\DescribeEnv[2][]{%
  \begingroup
  \def\DescribeEnv@name{#2}%
  \let\\\DescribeEnv@newline
```

Sets after-macro-arguments hook. First checks if the environment or macro version was used. The environment starts a new line only if the next token isn't `\end`, which is taken as end of the environment.

```
  \ifx\@currenvir\DescribeEnv@string
    \def\after@Macro@args{%
      \let\after@Macro@args\empty
      \setbox\@tempboxa\hbox\y@bgroup
      \@ifnextchar\end{}%
        {\DescribeEnv@newline}%
      #1%
    }%
```

The macro version adds the optional argument as content line if given.

```
  \else
    \ifx\relax#1\relax
      \def\after@Macro@args{%
        \y@bgroup
        \endDescribeEnv
      }%
    \else
      \def\after@Macro@args{%
        \setbox\@tempboxa\hbox\y@bgroup
        \DescribeEnv@newline\MacroArgs#1%
        \endDescribeEnv
      }%
    \fi
  \fi
```

Start `\vbox` and adds first line.

```
  \setbox\descbox\vbox\y@bgroup
  \envcodestyle
  \let\PrintEnv\PrintSubEnv
  \hbox\y@bgroup
  \PrintEnvName{\begin}{\DescribeEnv@name}%
  \macroargsstyle
  \read@Macro@arg
}
```

\DescribeEnv@newline

Closes existing and starts a new horizontal box representing a indented line. The optional argument allows to add extra space between lines like the normal \\. Negative values are not supported.

```
\newcommand*\DescribeEnv@newline[1][0pt]{%
  \strut\y@egroup
  {\vskip#1}%
  \hbox\y@bgroup\strut
  \hspace*{\descsep}%
  \ignorespaces
}%
```

\DescribeEnv@string

Holds the environment name for comparison.

```
\def\DescribeEnv@string{DescribeEnv}
```

\descbox

Save box to store description content.

```
\newbox\descbox
```

\endDescribeEnv

```
\def\endDescribeEnv{%
  \y@egroup
  \begingroup
  \setbox\@tempboxa\lastbox
  \ifcase0%
    \ifdim\wd\@tempboxa>\descsep1\fi
    \ifdim\ht\@tempboxa>\ht\strutbox1\fi
    \ifdim\dp\@tempboxa>\dp\strutbox1\fi
  \else
```

```
      \box\@tempboxa
    \fi
    \endgroup
    \hbox\y@bgroup
      \PrintEnvName{\end}{\DescribeEnv@name}
    \y@egroup
    \y@egroup
    \PrintEnv
    \endgroup
}
```

3.5.7 Print Macros

`\PrintMacroName`

Formats macro name. The backslash is forced to `tt` font.

```
\def\PrintMacroName#1{%
  {\macrodescstyle{\strut
   \texttt{\char92}%
   \escapechar\m@ne
   \string#1}}%
}
```

`\PrintLengthName`

Formats length register name.

```
\let\PrintLengthName\PrintMacroName
```

`\PrintEnvName`

`#1` = '`\begin`' or '`\end`', `#2` = env name.

```
\def\PrintEnvName#1#2{%
  \strut
  \string#1\braceleft
  {\macrodescstyle#2}%
  \braceright
}
```

`\PrintMacros`

Prints macros described using `\DescribeMacros`. The actual content was stored inside `\descbox`. If it is wider than the line width it is centered.

```
\def\PrintMacros{%
  \par\vspace\beforedescskip
  \noindent\hspace*{\descindent}%
  \ifdim\wd\descbox>\linewidth
    \makebox[\linewidth][c]{\fbox{\hspace*{\descsep}\↙
       usebox{\descbox}\hspace*{\descsep}}}
  \else
    \fbox{\hspace*{\descsep}\usebox{\descbox}\hspace↙
       *{\descsep}}%
  \fi
  \par\vspace\afterdescskip
}
```

`\PrintLength`

Prints lengths registers described using one or multiple `\DescribeLength`.

```
\let\PrintLength\PrintMacros
```

`\PrintEnv`

Prints `DescribeEnv` environments. The actual content was stored inside `\descbox`.

```
\def\PrintEnv{%
  \par\vspace\beforedescskip
  \noindent\hspace*{\descindent}%
  \fbox{\hspace*{\descsep}\usebox{\descbox}\hspace*{\↙
     descsep}}%
  \par\vspace\afterdescskip
}
```

`\PrintSubEnv`

Prints sub environments, i.e. `DescribeEnv` environments inside the body of another `DescribeEnv`. The actual content was stored inside `\descbox`.

```
\def\PrintSubEnv{%
  \hbox{\hbox{\usebox{\descbox}}}%
}
```

3.5.8 Special Character Macros

`\bslash`

Defines an expandable backslash with catcode 12: '$\_{12}$'. The `\@firstofone` trick is used to read the `\gdef\bslash` code before changing the catcode.

```
{%
\@firstofone{%
  \catcode`\\=12
  \gdef\bslash
}{\}
}%}
```

`\percent`

Defines an expandable percent character with catcode 12: '$\%_{12}$'.

```
\begingroup
\catcode`\%=12
\gdef\percent{%}
\endgroup
```

`\braceleft`

`\braceright`

Defines expandable left and right braces with catcode 12: '$\{_{12}$' '$\}_{12}$'.

```
\begingroup
\catcode`\<=1
\catcode`\>=2
\catcode`\{=12
\catcode`\}=12
\gdef\braceleft <{>
\gdef\braceright<}>
\endgroup
```

3.5.9 Other Macros

`\y@bgroup`

`\y@egroup`

These macros are used to begin and end `\vbox`/`\hbox`-es.

```
\def\y@bgroup{\bgroup\color@setgroup}
\def\y@egroup{\color@endgroup\egroup}
```

3.6 Include Code Examples

```
\RequirePackage{listings}
\lst@RequireAspects{writefile}
\def\ydoc@exafile{\jobname.exa}
```

`\exampleprintsettings`

```
\def\exampleprintsettings{frame=lines}%

\newbox\examplecodebox
\newbox\exampleresultbox
```

`\BoxExample`

```
\def\BoxExample{%
  \setbox\examplecodebox\hbox{\color@setgroup
    \expandafter\expandafter\expandafter\
       lstinputlisting
    \expandafter\expandafter\expandafter[%
    \expandafter\exampleprintsettings\expandafter,\
       thisexampleprintsettings]%
    {\ydoc@exafile}%
  \color@endgroup}%
  \setbox\exampleresultbox\hbox{\color@setgroup
    \@@input\ydoc@exafile\relax
  \color@endgroup}%
}
```

`\PrintExample`

```
\RequirePackage{showexpl}
\def\PrintExample{%
  \begingroup
  \lstset{basicstyle=\ttfamily}%
  \MakePercentComment
  \LTXinputExample[varwidth]{\ydoc@exafile}%
  \endgroup
}

\def\examplecodesettings{gobble=4}
```

`examplecode`

```
\lstnewenvironment{examplecode}[1][]{%
  \def\thisexampleprintsettings{#1}%
  \expandafter\lstset\expandafter{\
     examplecodesettings,#1}%
  \setbox\@tempboxa\hbox\bgroup
  \lst@BeginWriteFile{\ydoc@exafile}%
}
{%
  \lst@EndWriteFile
  \egroup
  \PrintExample
}

\RequirePackage{float}
```

example

```
\floatstyle{plain}
\newfloat{example}{tbhp}{loe}
\floatname{example}{\examplename}
\def\examplename{Example}
```

exampletable

```
\newenvironment{exampletable}{%
  \floatstyle{plaintop}%
  \restylefloat{example}%
  \example
}{\endexample}
```