The democodetools and democodelisting DEPRECATED

Alceu Frigeri

October 2025
(September 2022)

Abstract

This has been DEPRECATED in favour of **codedescribe**. It isn't supported since late 2022. The following documentation is just for the historical record. Loading this will generate an error (package deprecated), IF you really want to use it, you can still use it as `\usepackage{democodetools-1.0.1}`.

Abstract

This is 'yet another doc/docx/doc3' package. It is designed to be 'as class independent as possible', meaning: it makes no assumption about page layout (besides 'having a marginpar') or underline macros. Furthermore, it's assumed that `\maketitle` and the *abstract* environment were modified by the underline class, so alternatives (based on the article class) are provided. The main idea is to be able to document a package/class loading it first and then this, so that it is possible not only to document the 'syntax' but also to show the end result 'as is' when using that other specific class/package.

Contents

1 Introduction

The packages/classes doc/docx/doc3 (and for that matter doctools) where designed to be used with dtx files, which is handy for package developers, as long as one is fine with the 'default article' format (which is true most of the time). This came to be from the willingness of having the 'new look and feel' used in doc3, but, instead of having to rely on a standard class, being able to use any class as the base one, which allows to 'demonstrate the documented commands with the final layout'.

democodelisting defines a few macros to display and demonstrate LaTeX code verbatim (using *listings* and *scontents*), whilst *democodetools* defines a series of macros to display/enumerate macros and environments (somewhat resembling the *doc3* style).

2 democodelisting Package

It requires two packages: *listings* and *scontents*
Defines an environment: *stcode* and
4 commands: `\DemoCode`, `\DisplayCode`, `\TabbedDisplayCode` and `\setdclisting`.

2.1 In Memory Code Storage

Thanks to *scontents* (*expl3* based) it's possible to store LaTeX code snippets in a *expl3* key.

stcode `\begin{stcode} [⟨keys⟩] \end{stcode}`

This environment is an alias to *scontents* environment (from *scontents* package), all *scontents* keys are valid, with an additional one: *st* which is an alias to the *store-env* key. The environment body is stored verbatim in the *st* named key.

2.2 Code Display/Demo

\DisplayCode
\DemoCode
\TabbedDemoCode

`\DisplayCode [⟨dclisting-keys⟩] {⟨st-name⟩}`
`\DemoCode [⟨dclisting-keys⟩] {⟨st-name⟩}`
`\TabbedDemoCode [⟨dclisting-keys⟩] {⟨st-name⟩}`

`\DisplayCode` just typesets ⟨st-name⟩ (the key-name created with *stcode*), in verbatim mode with syntax highlight.
`\DemoCode` first typesets ⟨st-name⟩, as above, then it *executes* said code. Finally `\TabbedDemoCode` does the same, but typesetting it, and executed code, side by side. N.B. Both typeset and executed code are placed inside a *minipage* so that, when *executing* the code, one can have, for instance, 'normal' paragraph indentation.
For Example:

LaTeX Code:

```
  \begin{stcode}[st=stmeta]
    Some \LaTeX~coding, for example: \ldots.
\end{stcode}
This will just typesets \Key{stmeta}:

 \DisplayCode{stmeta}

and this will demonstrate it, side by side with source code:

\TabbedDemoCode[numbers=left,codeprefix={inner code},resultprefix={inner result
      }]{stmeta}
```

LaTeX Result:

This will just typesets *stmeta*:

LaTeX Code:

```
Some \LaTeX~coding, for example: \ldots.
```

and this will demonstrate it, side by side with source code:

inner code | inner result

```
Some \LaTeX~coding, for example:
  \ldots.
```

Some LaTeX coding, for example: . . . .

`\setdclisting`

`\setdclisting{⟨dclisting-keys⟩}`

Instead of setting/defining ⟨dclisting-keys⟩ per `\Demo`/`\Display` call, one can set those *globally*, better said, *in the called context group* .
N.B.: All `\Display`/`\Demo` commands create a local group (`\begingroup`) in which the ⟨dclisting-keys⟩ are defined, and discarded once said local group is closed. `\setdclisting` defines those keys in the *current* context/group (`\def`, `\edef`)

2.2.1 ⟨dclisting-keys⟩

Using a *key*=*value* syntax, one can fine tune *listings* syntax highlight.

⟨dclisting-keys⟩

settexcs, *settexcs2*, *settexcs3*
texcs, *texcs2*, *texcs3*
texcsstyle, *texcs2style*, *texcs3style*
Those define sets of LaTeX commands (csnames), the *set* variants initialize/redefine those sets (an empty list, clears the set), while the others extend those sets. The *style* ones redefines the command display style (an empty ⟨par⟩ resets the style to it's default).

setkeywd, *setkeywd2*, *setkeywd3*
keywd, *keywd2*, *keywd3*
keywdstyle, *keywd2style*, *keywd3style*
Same for other *keywords* sets.

setemph, *setemph2*, *setemph3*
emph, *emph2*, *emph3*
emphstyle, *emph2style*, *emph3style*
for some extra emphasis sets.

numbers, *numberstyle*
numbers possible values are *none* (default) and *left* (to add tinny numbers to the left of the listing). With *numberstyle* one can redefine the numbering style.

stringstyle, *commentstyle*
to redefine *strings* and *comments* formatting style.

bckgndcolor
to change the listing background's color.

codeprefix, *resultprefix*

those set the *codeprefix* (default: LaTeX Code:) and *resultprefix* (default: LaTeX Result:)

3 democodetools Package

3.1 Environments

`Macros`
`Envs`

`\begin{Macros}{⟨macrolist⟩}[⟨space⟩]`
`\begin{Envs}{⟨envlist⟩}[⟨space⟩]`

Those are the two main environments to describe *Macros* and *Environments*. Both typeset ⟨macrolist⟩ (csv list) or ⟨envlist⟩ (csv list) in the margin. The opcional ⟨space⟩ is “added” to the left marging length. N.B. Each element of the list gets `\detokenize`

`Syntax`

`\begin{Syntax}`

The *Syntax* environment sets the fontsize and activates `\obeylines`, so one can list macros/cmds/keys use, one per line.

LaTeX Code:

```
\begin{Envs}{Macros,Envs}
\begin{Syntax}%
\Macro{\begin{Macros}}{macrolist}
\Macro{\begin{Envs}}{envlist}
\end{Syntax}
Those are the two main ...
\end{Envs}
```

`Args`
`Keys`
`Values`
`Options`

`\begin{Args}`
`\begin{Args+}`
`\begin{Keys}`
`\begin{Keys+}`
`\begin{Values}`
`\begin{Values+}`
`\begin{Options}`
`\begin{Options+}`

Those environments are all the same, starting a dedicated *description list*. Together with the many `\Description...` commands, one can list all *Options*, *Args*, *Keys*, *Values* as needed. The + form are meant to be used with the `\Description...+` forms, for *in text* lists. The non + form are meant to have the many *'descriptors'* in the *margin par*.

3.2 Describe Commands

`\DescribeMacro`

`\DescribeMacro*!+{⟨csname⟩}[⟨oarglist⟩]{⟨marglist⟩}`

*****	typesets the macro name in bold face.
!	⟨marglist⟩ is treated as an expandable code, 'as is'.
+	the macro name is typeseted in text.
⟨***csname***⟩	macro name (`\detokenize`)
⟨***oarglist***⟩	csv list of optional args.
⟨***marglist***⟩	csv list of mandatory args.

`\DescribeArg`
`\DescribeKey`
`\DescribeValue`
`\DescribeOption`
`\DescribePackage`

`\DescribeArg*+[⟨type⟩]{⟨arg⟩}`
`\DescribeKey*+[⟨type⟩]{⟨arg⟩}`

```
\DescribeValue*+ [⟨type⟩] {⟨arg⟩}
\DescribeOption*+ [⟨type⟩] {⟨arg⟩}
\DescribePackage*+ [⟨type⟩] {⟨arg⟩}
```

`*`	typesets it in bold face.
`+`	typesets in text (not in marginpar)
⟨*type*⟩	key/arg/... format
⟨*arg*⟩	key/arg/... name.

3.3 Macros Typeset

`\Macro`

```
\Macro {⟨csname⟩} <⟨embl⟩> [⟨olist⟩] {⟨mlist⟩}
\Macro! {⟨csname⟩} <⟨embl⟩> {⟨par.desc.⟩}
```

When describing a macro ⟨`csname`⟩ (Command Sequence, csname) the ⟨`olist`⟩ and ⟨`mlist`⟩ are comma separated lists (csv) of optional and mandatory arguments. ⟨`embl`⟩ are optional, single char, 'embellishment' tokens, like * ! +. Finally, in the **!** form, the ⟨`par.desc.`⟩ is any text representing the macro parameter list (for non regular, non usual, cases).

LaTeX Code:

```
  \Macro
  {\Macro}<*!>[opt1,opt2]{arg3}
  \Macro!
  {\Macro}<!>{\xarg{embl}\marg{par.
    desc.}}
```

LaTeX Result:

```
\Macro*! [⟨opt1⟩] [⟨opt2⟩] {⟨arg3⟩}
\Macro! <⟨embl⟩> {⟨par.desc.⟩}
```

3.4 Args Typeset

`\oarg`
`\marg`
`\parg`
`\xarg`
`\Arg`
`\Meta`

```
\oarg [⟨type⟩] {⟨arg⟩}
\marg [⟨type⟩] {⟨arg⟩}
\parg [⟨type⟩] {⟨arg⟩}
\xarg [⟨type⟩] {⟨arg⟩}
\Arg [⟨type⟩] {⟨arg⟩}
\Meta {⟨arg⟩}
```

Those are meant to typeset the diverse kinds of 'command's arguments' (mandatory, optional, parenthesis . . .). `\Meta{arg}` typesets *arg* as ⟨arg⟩.

⟨*type*⟩	defaults to *`Meta`* (it's the csname of any valid formatting command, like Meta, textbf, etc.)
⟨*arg*⟩	the argument name itself.

LaTeX Code:

```
  \oarg{fam}
  \parg{xtra}
  \marg[textbf]{text}
  \xarg{x-text}
```

LaTeX Result:

[⟨fam⟩] (⟨xtra⟩) **{text}** <⟨x-text⟩>

3.5 Keys Typeset

`\Key`
`\Keylst`
`\KeyUse`

`\Key [⟨pre⟩] {⟨key⟩}`
`\Keylst [⟨default⟩] {⟨keylst⟩}`
`\KeyUse {⟨key⟩}`value

To typeset a ⟨Key⟩ or ⟨keylst⟩ (csv list). ⟨pre⟩ is just prepended to ⟨key⟩ whilst ⟨default⟩ is the default key value. `\KeyUse` is just a short-cut for a, possible, common construction.

LaTeX Code:

```
\Key{Akey}

\Keylst[Bkey]{Akey,Bkey}

\KeyUse{keyA}{arg}
```

LaTeX Result:

Akey
Akey , *Bkey* Default: ***Bkey***
keyA = ⟨arg⟩

`\Env`
`\Pack`
`\Value`

`\Env [⟨pre⟩] {⟨key⟩}`
`\Pack [⟨pre⟩] {⟨key⟩}`
`\Value [⟨pre⟩] {⟨key⟩}`

Similar to `\Key` above, they will typeset a ⟨Key⟩. ⟨pre⟩ is just prepended to ⟨key⟩ whilst ⟨default⟩ is the default key value.

3.6 Others

`\MetaFmt`

`\MetaFmt* [⟨type⟩]`

It sets the font size, series, face as defined by ⟨type⟩, ⟨type⟩ being one of *Oarg* , *Marg* , *Parg* , *Xarg* , *Macro* , *Code* , *Key* , *KeyVal* , *Option* , *Value* , *Default*. The star version uses bold.

`\MarginNote`

`\MarginNote {⟨text⟩}`

As the name implies, to add small margin notes.

`\dcAuthor`
`\dcDate`
`\dcTitle`
`\dcMakeTitle`

`\dcAuthor {⟨name⟩}`
`\dcDate {⟨date⟩}`
`\dcTitle {⟨title⟩}`
`\dcMakeTitle`

Those allow one to define (as in standard article, book, report classes) the document *author*, *date* and *date* `\dcMakeTitle` will write a typical title+author heading (as in the article class).

`dcAbstract`

`\begin{dcAbstract} \end{dcAbstract}`

Same as above, for the abstract.