styledcmd

Paolo De Donato

23 August 2021

styledcmd is a LaTeX package that allows you to create and manage different versions of your macro in order to be able to choose the better style for every occasion and avoid rewriting code each time.

1 How can you include it in your project?

You need only to have the file `styledcmd.sty` in your current working directory. Otherwise you can manually install it inside your preferred LaTeX compiler (for example `TeXLive` or `MiKTeX`) in order to make it available for all your projects. Instructions for manually install a package can be found on Internet.

Then once you've added it you can include in your project with this command:

```
\usepackage{styledcmd}
```

2 How do you use it?

You can create a formatted macro via the following command

`\newstyledcmd` `\renewstyledcmd` `\providestyledcmd`

```
\newstyledcmd {⟨macro name⟩} {⟨format name⟩} [⟨number of arguments⟩] {⟨code⟩}
```

You can repeat that command for the same macro in order to create different styles, for example these commands

```
\newstyledcmd{\saluto}{informal}[1]{Hi #1}
\newstyledcmd{\saluto}{formal}[1]{Good morning #1}
```

define the two formats `informal` and `formal` for macro `\saluto`. You can directly use the command `\saluto` and use the default format (the first declared one) or using a specific style by passing it as an optional argument. So commands

```
\saluto{uncle}
\saluto[informal]{uncle}
\saluto[formal]{uncle}
```

will be expanded respectively as Hi uncle, Hi uncle, Good morning uncle. With the same syntax you can use `\renewstyledcmd` and `\providestyledcmd` with the same meaning of `\renewcommand` and `\providecommand` respectively.

3 How do you change the default style?

In order to change the default style (the one used when you don't choose explicitily a format) you need to execute the following command

`\setGlobalStyle` `\setGlobalStyle {⟨command name⟩} {⟨new default format name⟩}`

For example in order to change the default style of command `\saluto` from `informal` to `formal` you need to execute command `\setGlobalStyle{\saluto}{formal}`. With this command the output of preceding commands will instead be Good morning uncle, Hi uncle, Good morning uncle.

4 Customize parameters with xparse

styledcmd loads automatically the xparse package for internal reasons. You can also define new styled commands with the same syntax used by `\NewDocumentCommand` with the following command

`\NewDocStyledCMD` `\NewDocStyledCMD {⟨command name⟩} {⟨format name⟩} {⟨arguments format⟩} {⟨code⟩}`
`\RenewDocStyledCMD`
`\ProvideDocStyledCMD`

For example we can create the following two styles

```
\NewDocStyledCMD{\prova}{stylea}{r<>}{Stile 1 #1}
\NewDocStyledCMD{\prova}{styleb}{r<>}{Stile 2 #1}
```

in order to execute

```
\prova<Hello>
\prova[stylea]<Hello>
\prova[styleb]<Hello>
```

which are expanded respectively as Stile 1 Hello; Stile 1 Hello; Stile 2 Hello. Notice that the first optional argument passed to a command defined via `\NewDocStyleCMD` will always be interpreted as a style argument, so you should use another syntax for optional arguments or use a mandatory argument for the first place.

For example this declaration `\NewDocStyledCMD{\bad}{style}{o m}{Bad declaration}` should be avoided since for example `\bad[arg1]{arg2}` will interpret `arg1` as a style name and not as the first optional argument for `\bad`.

5 Commands group

You can define also a commands group in which defined commands will share the same styles, that allow you to change the style for all commands in that group.

To start a group you have to use the following macro

`\styBeginGroup` `\styBeginGroup {⟨group name⟩}`

any group should be terminated via the following macro

\styEndGroup `\styEndGroup`

In each group commands are grouped toghether according to the used style. So inside a group you can define a style with the command

\styBeginStyle `\styBeginStyle {⟨format name⟩}`

and once you've finished to define command with that style you have to close it with the command

\styEndStyle `\styEndStyle`

In order to define a command inside a `\styBeginStyle ... \styEndStyle` you can use

\newGstyledcmd `\newGstyledcmd {⟨command name⟩} [⟨arguments number⟩] {⟨code⟩}`

or

\NewGDocStyledCMD `\NewGDocStyledCMD {⟨command name⟩} {⟨arguments format⟩} {⟨code⟩}`

if you want to use the `\NewDocumentCommand` syntax.

Notice that in this case you mustn't define the style name in `\newGstyledcmd` or in `\NewGDocStyledCMD` since they use the format defined in the enclosing `\styBeginStyle ... \styEndStyle`. The defined style for a group is the first defined one, in order to change the default format you should use the following macro

\setGroupStyle `\setGroupStyle {⟨group name⟩} {⟨new default style⟩}`

Consider now the following example

```
\styBeginGroup{saluti}
  \styBeginStyle{formal}
    \newGstyledcmd{\ciao}[1]{Good morning #1}
    \newGstyledcmd{\via}{Goodbye}
  \styEndStyle

  \styBeginStyle{informal}
    \newGstyledcmd{\ciao}[1]{Hi #1}
    \newGstyledcmd{\via}{Catch you later}
  \styEndStyle
\styEndGroup
```

We've defined the group `saluti` with two styles: `formal` and `informal`. In that group are also defined the two commands `\ciao` and `\via` for both the styles. Since in `saluti` group the first defined style is `formal` it'll be also the default style for this group, so commands `\ciao{Enrico}` and `\via` will be expanded as Good morning Enrico and Goodbye.

If now we execute the command `\setGroupStyle{saluti}{informal}` we change the default style to `informal` so the preceding commands this time will be expanded as Hi Enrico and Catch you later.

You can also pass an optional argument for the style name you want to use, in particular `\ciao[formal]{Enrico}` will be always expanded as Good morning Enrico.

6 Generate more styled commands

Suppose now that you've created your own version of `\newcommand` or `\NewDocumentCommand` in order to generate some classes of commands. Suppose that your function name is `\mynewcommand` and it accepts, like `\newcommand`, the new macro as its first argument. Then you can create styled commands with your own macro by using this function

`\stycmd_generate:NN`
`\stycmd_generate_renew:NN`

`\stycmd_generate:NN` ⟨*new generator name*⟩ ⟨*generator name*⟩

So for example executing

```
\stycmd_generate:NN\mynewstyledcommand\mynewcommand
```

you'll create the new macro `\mynewstyledcommand` that accepts the command you want to define as first argument, the style you want to use as second argument and the remaining arguments are the same needed by `\mynewcommand`.

Notice that `\newstyledcmd` is defined internally as

```
\stycmd_generate:NN \newstyledcmd \newcommand
```

Notice that `\mynewcommand` first argument has to be the macro you want to define, otherwise `\stycmd_generate:NN` will have undefined behaviour. The `\stycmd_generate_renew:NN` function should be used only for "renew" type commands that requires a preceding declaration, like `\renewcommand` and `\RenewDocumentCommand`.

Actually there isn't any analogue proceeding for defining customized versions for `\newGstyledcmd` and `\NewGDocStyledCMD`.

7 Implementation

```
⟨*package⟩
⟨@@=stycmd⟩
```

`\c__stycmd_defname_str` the default style

```
\str_const:Nn \c__stycmd_defname_str{ default }
```

(*End definition for* `\c__stycmd_defname_str`.)

`\l__stycmd_cexp_str` Temporary variables to store expanded names.
`\l__stycmd_fexp_str`

```
\str_new:N \l__stycmd_cexp_str
\str_new:N \l__stycmd_fexp_str
```

(*End definition for* `\l__stycmd_cexp_str` *and* `\l__stycmd_fexp_str`.)

`\__stycmd_cmd:nn` Name of a command bounded to some style.
`\__stycmd_cmd:nV`
`\__stycmd_cmd:VV`
`\__stycmd_cmd:`
`\__stycmd_cmddefname:n`

```
\cs_new:Npn \__stycmd_cmd:nn #1#2{ __stycmd_command_#1_#2 }

\cs_generate_variant:Nn \__stycmd_cmd:nn { nV }
\cs_generate_variant:Nn \__stycmd_cmd:nn { VV }

\cs_new:Nn \__stycmd_cmd:
  {
    \__stycmd_cmd:VV\l__stycmd_cexp_str\l__stycmd_fexp_str
```

```
  }

\cs_new:Npn \__stycmd_cmddefname:n #1
  {
    \__stycmd_cmd:nV { #1 } \c__stycmd_defname_str
  }
```

(*End definition for* `\__stycmd_cmd:nn`, `\__stycmd_cmd:`, *and* `\__stycmd_cmddefname:n`.)

`\__stycmd_erroring:nn` `\__stycmd_erroring:nV` `\__stycmd_erroring:VV`

Error messages for undefined styles.

```
\msg_new:nnn { stycmd } { noformat }
  {
    Style~#2~not~defined~for~command~#1
  }

\cs_new_protected:Npn\__stycmd_erroring:nn #1#2
  {
    \cs_if_free:cT { \__stycmd_cmd:nn {#1} {#2} }
      { \msg_error:nnnn { stycmd } { noformat } {#1} {#2} }
  }

\cs_generate_variant:Nn \__stycmd_erroring:nn { nV, VV }
```

(*End definition for* `\__stycmd_erroring:nn`.)

`\__stycmd_deferr:`

Error message when try to create the `default` style

```
\msg_new:nnn { stycmd } { deferror }
  {
    Cannot~define~a~style~with~name~default
  }

\cs_new_protected:Nn\__stycmd_deferr:
  {
    \str_if_eq:NNT \l__stycmd_fexp_str \c__stycmd_defname_str
      { \msg_error:nn { stycmd } { deferror }  }
  }
```

(*End definition for* `\__stycmd_deferr:`.)

`\__stycmd_macro_default:nn` `\__stycmd_macro_default:VV`

Sets the default style to use.

```
\cs_new_protected:Npn \__stycmd_macro_default:nn #1#2
  {
    \exp_args:Nc \ProvideDocumentCommand { \__stycmd_cmddefname:n { #1 } }
      {}
      {
        \use:c { \__stycmd_cmd:nn { #1 } { #2 } }
      }
  }

\cs_generate_variant:Nn \__stycmd_macro_default:nn { VV }
```

(*End definition for* `\__stycmd_macro_default:nn`.)

__stycmd_macro_overwrite:nn
__stycmd_macro_overwrite:VV

Overwrite the default style, not the command associated to that style.

```
\cs_new_protected:Npn \__stycmd_macro_overwrite:nn #1#2
  {
    \__stycmd_erroring:nn { #1 } { #2 }

    \exp_args:Nc \RenewDocumentCommand { \__stycmd_cmddefname:n { #1 } }
      {}
      {
        \use:c { \__stycmd_cmd:nn { #1 } { #2 } }
      }
  }

\cs_generate_variant:Nn \__stycmd_macro_overwrite:nn { VV }
```

(*End definition for* `\__stycmd_macro_overwrite:nn`.)

__stycmd_macro_declaration:n
__stycmd_macro_declaration:V

Declare the effective macro. It'll overwrite existing `\l__stycmd_fexp_str`.

```
\cs_new_protected:Npn \__stycmd_macro_declaration:n #1
  {
    \exp_args:Nc \ProvideDocumentCommand { #1 } {o}
      {
        \IfNoValueTF {##1}
          {
            \use:c { \__stycmd_cmddefname:n { #1 } }
          }
          {
            \str_set:Nx \l__stycmd_fexp_str {##1}
            \__stycmd_erroring:nV { #1 } \l__stycmd_fexp_str
            \use:c{ \__stycmd_cmd:nV { #1 } \l__stycmd_fexp_str }
          }
      }
  }

\cs_generate_variant:Nn \__stycmd_macro_declaration:n { V }
```

(*End definition for* `\__stycmd_macro_declaration:n`.)

__stycmd_init_vars:Nn

Initializes system variables. First argument is the command name, second argument should be `x` expandable.

```
\cs_new_protected:Npn \__stycmd_init_vars:Nn #1#2
  {
    \str_set:Nx \l__stycmd_cexp_str { \cs_to_str:N #1 }
    \str_set:Nx \l__stycmd_fexp_str {#2}
    \__stycmd_deferr:
  }
```

(*End definition for* `\__stycmd_init_vars:Nn`.)

\stycmd_generate:NN
\stycmd_generate_renew:NN

Declare the styled version `#1` of the macro generator command `#2`. the `_renew` variant requires a preceding declaration

```
\cs_new_protected:Npn \stycmd_generate:NN #1#2
  {
    \NewDocumentCommand #1 {m m}
      {
```

```
        \__stycmd_init_vars:Nn { ##1 } { ##2 }
        \__stycmd_macro_default:VV \l__stycmd_cexp_str \l__stycmd_fexp_str
        \__stycmd_macro_declaration:V \l__stycmd_cexp_str
        \exp_args:Nc #2 { \__stycmd_cmd: }
      }
  }

\cs_new_protected:Npn \stycmd_generate_renew:NN #1#2
  {
    \NewDocumentCommand #1 {m m}
      {
        \__stycmd_init_vars:Nn { ##1 } { ##2 }
        \__stycmd_erroring:VV \l__stycmd_cexp_str \l__stycmd_fexp_str
        \exp_args:Nc #2 { \__stycmd_cmd: }
      }
  }
```

(*End definition for* `\stycmd_generate:NN` *and* `\stycmd_generate_renew:NN`*. These functions are documented on page 4.*)

`\newstyledcmd` `\renewstyledcmd` `\providestyledcmd`

Declare a new macro with the specified style name.

```
\stycmd_generate:NN \newstyledcmd \newcommand
\stycmd_generate_renew:NN \renewstyledcmd \renewcommand
\stycmd_generate:NN \providestyledcmd \providecommand
```

(*End definition for* `\newstyledcmd`*,* `\renewstyledcmd`*, and* `\providestyledcmd`*. These functions are documented on page 1.*)

`\NewDocStyledCMD` `\RenewDocStyledCMD` `\ProvideDocStyledCMD`

Declare a new styled macro with the `\NewDocumentCommand` syntax.

```
\stycmd_generate:NN \NewDocStyledCMD \NewDocumentCommand
\stycmd_generate_renew:NN \RenewDocStyledCMD \RenewDocumentCommand
\stycmd_generate:NN \ProvideDocStyledCMD \ProvideDocumentCommand
```

(*End definition for* `\NewDocStyledCMD`*,* `\RenewDocStyledCMD`*, and* `\ProvideDocStyledCMD`*. These functions are documented on page 2.*)

`\setGlobalStyle`

Change the default style.

```
\NewDocumentCommand \setGlobalStyle{m m}{
    \__stycmd_init_vars:Nn { #1 } { #2 }

    \__stycmd_macro_overwrite:VV \l__stycmd_cexp_str \l__stycmd_fexp_str
}
```

(*End definition for* `\setGlobalStyle`*. This function is documented on page 2.*)

`\l__stycmd_group_str` `\l__stycmd_format_str`

Variables that stores group and format names.

```
\str_new:N \l__stycmd_group_str
\str_new:N \l__stycmd_format_str
```

(*End definition for* `\l__stycmd_group_str` *and* `\l__stycmd_format_str`*.*)

`\__stycmd_gvar:n` `\__stycmd_gvar:V` `\__stycmd_gvar:`

Group command list.

```
\cs_new:Npn \__stycmd_gvar:n #1 { g__stycmd_glist_#1_seq }

\cs_generate_variant:Nn \__stycmd_gvar:n { V }

\cs_new:Nn \__stycmd_gvar: { \__stycmd_gvar:V \l__stycmd_group_str }
```

(*End definition for* `\__stycmd_gvar:n` *and* `\__stycmd_gvar:`.)

`\__stycmd_push_g:Nn` `\__stycmd_push_g:NV` Pushes command name inside group command sequence. First argument is command, second is group name

```
\cs_new_protected:Npn \__stycmd_push_g:Nn #1#2
  {
    \seq_gput_left:cn { \__stycmd_gvar:n { #2 } } { #1 }
  }

\cs_generate_variant:Nn \__stycmd_push_g:Nn { NV }
```

(*End definition for* `\__stycmd_push_g:Nn`.)

`\styBeginGroup` Starts a group.

```
\msg_new:nnn { stycmd } { gnotclosed } { Group~not~closed }

\NewDocumentCommand \styBeginGroup { m }
  {
    \str_if_empty:NF \l__stycmd_group_str
      {
        \msg_error:nn { stycmd } { gnotclosed }
      }

    \str_set:Nx \l__stycmd_group_str {#1}

    \seq_if_exist:cF { \__stycmd_gvar: }
      { \seq_new:c { \__stycmd_gvar: } }
  }
```

(*End definition for* `\styBeginGroup`. *This function is documented on page 2.*)

`\styEndGroup` Closes a group.

```
\NewDocumentCommand \styEndGroup {}
  {
    \str_clear:N \l__stycmd_group_str
  }
```

(*End definition for* `\styEndGroup`. *This function is documented on page 3.*)

`\styBeginStyle` Starts a style subgroup.

```
\msg_new:nnn { stycmd } { fnotclosed } { Format~not~closed }

\NewDocumentCommand \styBeginStyle { m }
  {
    \str_if_empty:NF \l__stycmd_format_str
      {
        \msg_error:nn { stycmd } { fnotclosed }
      }
    \str_set:Nx \l__stycmd_format_str {#1}
  }
```

(*End definition for* `\styBeginStyle`. *This function is documented on page 3.*)

\styEndStyle Closes a style subgroup.

```
\NewDocumentCommand \styEndStyle {}
  {
    \str_clear:N \l__stycmd_format_str
  }
```

(*End definition for* `\styEndStyle`. *This function is documented on page* 3.)

\newGstyledcmd Declare a command inside a styled group.

```
\NewDocumentCommand \newGstyledcmd { m o +m }
  {
    \IfNoValueTF {#2}
      {
        \newstyledcmd {#1} { \l__stycmd_format_str } {#3}
      }
      {
        \newstyledcmd {#1} { \l__stycmd_format_str } [#2] {#3}
      }
    \__stycmd_push_g:NV {#1} \l__stycmd_group_str
  }
```

(*End definition for* `\newGstyledcmd`. *This function is documented on page* 3.)

\NewGDocStyledCMD Equivalent of `\NewDocStyledCMD` for command defined in a group.

```
\NewDocumentCommand \NewGDocStyledCMD { m m +m }
  {
    \NewDocStyledCMD {#1} { \l__stycmd_format_str } {#2} {#3}

    \__stycmd_push_g:NV {#1} \l__stycmd_group_str
  }
```

(*End definition for* `\NewGDocStyledCMD`. *This function is documented on page* 3.)

\setGroupStyle Change the default style for a group.

```
\NewDocumentCommand \setGroupStyle { m m }
  {
    \str_set:Nx \l__stycmd_group_str {#1}
    \str_set:Nx \l__stycmd_format_str {#2}
    \seq_map_inline:cn { \__stycmd_gvar: }
      {
        \setGlobalStyle {##1} { \l__stycmd_format_str }
      }
    \str_clear:N \l__stycmd_format_str
    \str_clear:N \l__stycmd_group_str
  }
```

(*End definition for* `\setGroupStyle`. *This function is documented on page* 3.)

⟨/package⟩