# The `mathcommand` package for LaTeX

[version v1.02–2019/07/03]

Thomas Colcombet

July 3, 2019

**Abstract**

The `mathcommand` package provides functionalities for defining macros

1. that have different behaviors depending on whether in math or text mode,
2. that absorb Primes, Indices and Exponents (PIE) as extra parameters usable in the code, and
3. offers some iteration facilities for defining macros with similar code.

The primary objective of this package is to be used together with the `knowledge` package for a proper handling of mathematical notations.

## 1 History of the package

**2019-05-12** First version of the package. V1.01 on CTAN,
**2019-05-14** New macros `\IfEmptyTF`, `\GetExponent`, and `\GetIndex`.
**2019-07-03** Corrects bug of functions declaring PIE's issuing an error when already existing. Version 1.02 on CTAN.

## 2 Defining text and math commands

The principle is that the package will maintain, for a macro `\macro`, two concurrent version of the code: a *math variant* (technically it is stored in a macro `\Math macro`) and a *text variant* (technically stored in a macro `\Text macro`[1]). The macro `\macro` itself will execute one or the other depending on whether it is executed in math or text mode. Note that all the macros are non-expandable for avoiding problems with mathematics that would be sent, for instance, to the table of contents. The list of commands is described at the end of the section.

For instance after executing:

```
\newmathcommand\macro[1]{\mathit{math}^\mathrm{code}_{(#1)}}
\newtextcommand\macro[1]{text code (#1)}
```

when executing `\macro` in math mode, the math code will be executed, and in text mode similarly:

[1] These cannot be used accidentally by the user since these control sequences contain a space.

`\macro{a}` yields 'text code (a)' while `$\macro{a}$` yields '$math^{\text{code}}_{(a)}$'.

If the macro `\macro` already exists, it is stored under the name `\LaTeXmacro`, and then everything's happen as if it had already been defined both in math and text mode.

This is interesting for redefining known macros. For instance `\c` is a convenient way to producing cedillas in LaTeX, as in `\c a` which yields 'a̧'. However, one may want `\c` to reprensent a variable $c$ in math mode. This is done using, e.g.:

```
\renewmathcommand\c{c}
```

Then, the macro `\c` still works in text mode, and using `\c` in math mode does display simply '$c$'.

The name of the macros offered by the `mathcommand` package are mere adaptations of the standard macros of LaTeX and of the package `xparse`[2]. Their syntax is the same (in particular in terms of parameter definitions):

`\newmathcommand` is similar to `\newcommand` and creates a math variant,
`\newtextcommand` is similar to `\newcommand` and creates a text variant,
`\renewmathcommand` is similar to `\renewcommand` and creates a math variant,
`\renewtextcommand` is similar to `\renewcommand` and creates a text variant,
`\declaremathcommand` is similar to `\newcommand` but defines the macro even if it exists before; it creates a math variant,
`\declaretextcommand` is similar to `\newcommand` but defines the macro even if it exists before; it creates a text variant,
`\NewDocumentMathCommand` is like `\NewDocumentCommand` of the `xparse` package, but creates a math variant,
`\NewDocumentTextCommand` is like `\NewDocumentCommand` of the `xparse` package, but creates a text variant,
`\RenewDocumentMathCommand` is like `\RenewDocumentCommand` of the `xparse` package, but creates a math variant,
`\RenewDocumentTextCommand` is like `\RenewDocumentCommand` of the `xparse` package, but creates a text variant,
`\DeclareDocumentMathCommand` is like `\DeclareDocumentCommand` of the `xparse` package, but creates a math variant,
`\DeclareDocumentTextCommand` is like `\DeclareDocumentCommand` of the `xparse` package, but creates a text variant,
`\ProvideDocumentMathCommand` is like `\ProvideDocumentCommand` of the `xparse` package, but creates a math variant,
`\ProvideDocumentTextCommand` is like `\ProvideDocumentCommand` of the `xparse` package, but creates a text variant.

The package offers also the following commands:

`\declarecommand` which is similar to `\newcommand` but defines the macro even if it exists before,
`\storecommand[optional-prefix]\macro` which copies the content of the macro `\macro` to `\optional-prefixmacro`. By default, the optional prefix is `LaTeX`. (Hence, it

[2]The package `xparse` offers a very convenient way to define macros with complicated parameter signatures.

does what is automatically made by commands such as `\declarecommand`).

# 3 Defining Prime/Indices/Exponents absorbing commands (PIE commands)

Another feature offered by the `mathcommand` package is to permit the definitions of macros that would absorb the primes, subscript and superscript that follow them. The three pieces of information are abbreviated as *PIE* (for "Primes-Indices-Exponents"). This terminology serves as a help for remembering the order prime-index-exponent. A *PIE command* is similar to a normal macro/command, but for the fact that the PIEs that follow are absorbed and can be used in the macro as three extra parameters. The list of macros usable for for defining PIE commands can be found at the end of this section.

This is best explained through an example. After writing:

```
\newcommandPIE\macro[1]{([#1]#3)#2#4}
```

one obtains that

`$\macro{A}_2'$` yields $([A]_2)'$ .

Indeed, in the body of the definition of `macro`, `#1` represents the normal parameter of the command, while the three following parameters (`#2`,`#3`,`#4` in this case) contain respectively the primes (either empty or a sequence of $'$ symbols), the index (either empty if there is no subscript or of the form `_{index}` if there is an index), and the exponent (either empty if there is no superscript or of the form `^{exponent}` if there is one). In the case of the above definition, the index (parameter `#3`) is written inside the parenthesis, while primes and exponents are put outside.

The are furthermore some helper functions:

`\IfEmptyTF` takes a string and two codes, and expands to the first one if the string is empty, and the second otherwise,

`\GetIndex` takes a string that is an index as in PIE commands, and expands to its content: it maps the empty string to the empty string, and strings of the form `_{sthg}` to `sthg`,

`\GetExponent` takes a string that is an exponent as in PIE commands, and expands to its content: it maps the empty string to the empty string, and strings of the form `^{sthg}` to `sthg`.

For instance:

```
\newmathcommandPIE\F{#2F#1\IfEmptyTF{#3}{}{^{(\GetExponent{#3})}}}
```

displays `$\F_2^3$` as '${}_2F^{(3)}$': the index is placed before, and the exponent is surrounded by parentheses.

**List of macros of defining PIE commands.** Once more, appart from the specificity of PIE commands, the syntax is as the original corresponding commands these are based on.

`\newcommandPIE` is similar to `\newcommand` (but defines a non-expandable macro)
`\renewcommandPIE` is similar to `\renewcommand` (but defines a non-expandable macro)
`\declarecommandPIE` is similar to `\newcommand` and works even if the macro already exists (and defines a non-expandable macro)
`\NewDocumentCommandPIE` is similar to `\NewDocumentCommand` of the `xparse` package,
`\RenewDocumentCommandPIE` is similar to `\RenewDocumentCommand` of the `xparse` package,
`\DeclareDocumentCommandPIE` is similar to `\DeclareDocumentCommand` of the `xparse` package,
`\ProvideDocumentCommandPIE` is similar to `\ProvideDocumentCommand` of the `xparse` package.

Finally, a bunch of macros are used to define math variants that are PIE commands:

`\newmathcommandPIE` is like `\newcommandPIE` and creates a math variant,
`\renewmathcommandPIE` is like `\renewcommandPIE` and creates a math variant,
`\declaremathcommandPIE` is like `\declarecommandPIE` and creates a math variant,
`\NewDocumentMathCommandPIE` is like `\NewDocumentCommandPIE`, but creates a math variant,
`\RenewDocumentMathCommandPIE` is like `\RenewDocumentCommandPIE`, but creates a math variant,
`\DeclareDocumentMathCommandPIE` is like `\DeclareDocumentCommandPIE`, but creates a math variant,
`\ProvideDocumentMathCommandPIE` is like `\ProvideDocumentCommandPIE`, but creates a math variant,

# 4 Looping for defining commands

The `mathcommand` package offer also some capabilities for automatically defining multiple similar macros. This is done using only one command:

`\LoopCommands`{list on which to iterate}[name 1][name 2]...[name 7]{code}

The *list on which to iterate* is a list of letters or braced sequences of letters. the *name 1*, *name 2* up to *name 7* optional parameters are expandable pieces of code that are to be evaluated and then converted into control sequences; they may use the extra parameter `#1`. Finally, *code* is the code to be executed that can use the parameters `#1`, `#2`, up to `#8`.

The result of executing this macro is that each of the letters or sequences of letters in the list on which to iterate will be taken one after the other. For each of them, the code is executed, taking as value of the parameter `#1` the element in the list, and as parameters `#2` to `#8` control sequences constructed from the evaluation of name 1 up to name 7 (using as parameters `#1` the element of the sequence).

For instance, imagine one easily wants to denote vectors simply as '`\vx`' instead of '`\vec` x' or '`\vec`{x}', it is sufficient to write:

```
\LoopCommands{abcdefghijklmnopqrstuvwxyz}[v#1]
    {\newcommand#2{\vec #1}}
```

It will result in the successive execution of `\newcommand\va{\vec a}` and so on up to `\newcommand\vz{\vec z}`.

Note also that the list on which to iterate is automatically expanded, and if a non-expandable control sequence is met, then it is replaced by its the text defining the control sequence. Hence using `{\alpha\beta}` is equivalent to `{{alpha}{beta}}`.

Some extra remarks may be helpful:

- As usual in TEX/LATEX, the code may have to use its own internal parameters, for instance for defining macros: such parameters should use double #'s, i.e., `##1`, `##2` up to `##9`.

  For instance:

  ```
  \LoopCommands{abcdefghijklmnopqrstuvwxyz}[o#1]
      {\declarecommandPIE#2{\overline{#1##2}##1##3}
  ```

  will result in `\ou` to be declared as the PIE command defined with as main body `\overline{u#2}#1#3` (note the translation of parameters, which is the standard way to proceed for TEX). In our case `\ou_1^2` yields '$\overline{u_1}^2$' (the subscript gets to be inside the bar, and the superscript and primes outside), and so on...

- When defining multiple commands, some may already exist. To avoid conflicts, one should use the 'declare' version of the defining commands. These will work independently of the context. Is it also good to define only the math variants using the appropriate commands of the package.

- The following strings are predefined for the user to loop on:

  `\lettersUppercase` stands for `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

  `\lettersLowercase` stands for `abcdefghijklmnopqrstuvwxyz`

  `\lettersAll` stands for `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`.

  `\lettersGreekLowercase` stands for $\alpha\beta\gamma\delta\epsilon\varepsilon\zeta\eta\theta\vartheta\iota\kappa\lambda\mu\nu\xi\pi\varpi\rho\varrho\sigma\varsigma\tau\upsilon\phi\varphi\chi\psi\omega$.

  `\lettersGreekUppercase` stands for $\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega$.

  `\lettersGreekAll` stands for $\alpha\beta\gamma\delta\epsilon\varepsilon\zeta\eta\theta\vartheta\iota\kappa\lambda\mu\nu\xi\pi\varpi\rho\varrho\sigma\varsigma\tau\upsilon\phi\varphi\chi\psi\omega\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega$.

  Hence, for instance:

```
\LoopCommands\lettersUppercase[bb#1]
      {\newmathcommand{\mathbb#1}}
\LoopCommands\lettersGreekLowercase[#1][LaTeX#1]
      {\renewmathcommand#2{\textcolor{blue}{#3}}
```

configures the macros `\bbA`, ..., `\bbZ` to display the letters in blackboard bold alphabet (e.g. with the `amsfonts` package), and the lowercase greek letters `\alpha`,... to be displayed in blue (with the `xcolor` package loaded). Note in the last

case the use of the third parameter used for accessing the macros `\LaTeXalpha`,... that are automatically generated by the `\renewmathcommand` macro.

# 5 Implementation

## 5.1 README.md

<*readme>

```
This directory contains the package

  name: mathcommand
  license: LaTeX Project Public License version 1.2 or above
  version: v1.02
  date: 2019/07/03
  author: Thomas Colcombet
  mail: thomas.colcombet@irif.fr
  web: -

Purpose:
 The mathcommand package provides functionalities for defining macros:
  - that have different behaviors depending on whether in math or text mode,
  - absorb Primes, Indices, Exponents (PIE) following LaTeX notations and
    have them as extra parameters usable in the code.
 The primary objective of this code is to be used together with the knowledge
 package for a proper handling of mathematical notations.

Install:
 It is sufficient to have the file mathcommand.sty accessible by LaTeX.
 It can be produced by 'make mathcommand.sty' if necessary.
 The documentation is in the file mathcommand.pdf.

Content of the file mathcommand-ctan.zip:
 - README.md: this file generated while compiling mathcommand.ins,
 - mathcommand.sty: the package file (generated using knowledge.ins)
 - mathcommand.pdf: the user documentation (generated by compiling
   mathcommand.dtx)
 - makefile: the makefile. Use 'make all' to generate mathcommand.sty
   and knowledge.pdf. It can also: clean the directory, make zip
   version of the sources, or ready for CTAN.
 - mathcommand.ins: is the file generating mathcommand.sty and
   README.md from mathcommand.dtx (using docstrip).
 - mathcommand.dtx: code and documentation.

```

</readme>

## 5.2 Code preparation

<*package>

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\RequirePackage{expl3}
\RequirePackage{etoolbox}
\RequirePackage{xparse}
```

```
\ExplSyntaxOn
\bool_if_exist:NTF\mathcommand_package_loaded_bool
 \endinput
 {\bool_new:N\mathcommand_package_loaded_bool
  \bool_set_true:N\mathcommand_package_loaded_bool}
```

## 5.3 Absorbing primes, indices and exponents (PIE)

### 5.3.1 Parsing pies

We start by defining the code used to absorb PIEs from the input stream. The main function defined in this context is

`\__mathcommand_absorb_PIE:nw`

which takes some code as first parameter, then absorbs primes, indices and exponents, and finally reinserts the code in the input stream, followed with three braces containing respectively the primes, the index, and the exponent.

It works by storing the code to be executed in `\__mathcommand_absorb_finished_tl`, preparing `\__mathcommand_primes_tl`, `\__mathcommand_index_t`, and `\__mathcommand_exponent_tl` to contain the PIEs. Then the core of the parsing is performed by `\__mathcommand_absorb:w`.

```
\cs_new:Npn\__mathcommand_absorb_PIE:nw#1{
  \tl_set:Nn\__mathcommand_absorb_finished_tl{#1}
  \tl_set:Nn\__mathcommand_primes_tl{}
  \tl_set:Nn\__mathcommand_index_tl{}
  \tl_set:Nn\__mathcommand_exponent_tl{}
  \__mathcommand_absorb:w
}
```

When the parsing is finished, `\__mathcommand_absorb_finished:` is executed, which inserts the original code stored in `\__mathcommand_absorb_finished_tl` followed by the PIEs in the input stream.

```
\cs_new:Nn\__mathcommand_absorb_finished:{
  \exp_args:NV\__mathcommand_absorb_finished_:\__mathcommand_exponent_tl
}
\cs_new:Nn\__mathcommand_absorb_finished_:{
  \exp_args:NV\__mathcommand_absorb_finished__:\__mathcommand_index_tl
}
\cs_new:Nn\__mathcommand_absorb_finished__:{
  \exp_args:NV\__mathcommand_absorb_finished_tl\__mathcommand_primes_tl
}
```

```
\cs_new:Npn\peek_subscript_remove:TFw
    {\peek_charcode_remove:NTF _}
\cs_new:Npn\peek_superscript_remove:TFw
    {\peek_charcode_remove:NTF ^}
\cs_new:Npn\peek_prime_remove:TFw
    {\peek_charcode_remove:NTF '}
```

```
\cs_new:Nn\__mathcommand_absorb_add_prime:{
  \tl_put_right:Nn\__mathcommand_primes_tl{'}
}
\ExplSyntaxOff
\expandafter\def\csname g_tmpa_tl\endcsname{_}
\ExplSyntaxOn
\cs_new:Nx\__mathcommand_absorb_add_index_after:Nn{
  \exp_not:N\tl_set:Nn\exp_not:N\__mathcommand_index_tl
     {\g_tmpa_tl{#2}}
  #1
}
\cs_new:Nn\__mathcommand_absorb_add_exponent_after:Nn{
  \tl_set:Nn\__mathcommand_exponent_tl{^{#2}}
  #1
}
\cs_new:Npn\__mathcommand_absorb:w{
   \peek_prime_remove:TFw
      {\__mathcommand_absorb_add_prime:
       \__mathcommand_absorb_p:w}
      \__mathcommand_absorb_:w}
\cs_new:Npn\__mathcommand_absorb_:w{
   \peek_subscript_remove:TFw
      {\__mathcommand_absorb_add_index_after:Nn
       \__mathcommand_absorb_i:w}
      \__mathcommand_absorb__:w}
\cs_new:Npn\__mathcommand_absorb__:w{
   \peek_superscript_remove:TFw
      {\__mathcommand_absorb_add_exponent_after:Nn
       \__mathcommand_absorb_e:w}
      \__mathcommand_absorb_finished:}
\cs_new:Npn\__mathcommand_absorb_p:w{
   \peek_prime_remove:TFw
      {\__mathcommand_absorb_add_prime:
       \__mathcommand_absorb_p:w}
      \__mathcommand_absorb_p_:w}
\cs_new:Npn\__mathcommand_absorb_p_:w{
   \peek_subscript_remove:TFw
      {\__mathcommand_absorb_add_index_after:Nn
       \__mathcommand_absorb_pi:w}
      \__mathcommand_absorb_finished:}
\cs_new:Npn\__mathcommand_absorb_pi:w{
   \peek_prime_remove:TFw
      {\__mathcommand_absorb_add_prime:
       \__mathcommand_absorb_pi:w}
      \__mathcommand_absorb_finished:}
\cs_new:Npn\__mathcommand_absorb_e:w{
   \peek_subscript_remove:TFw
```

```
      {\__mathcommand_absorb_add_index_after:Nn
       \__mathcommand_absorb_finished:}
      \__mathcommand_absorb_finished:}
\cs_new:Npn\__mathcommand_absorb_i:w{
   \peek_prime_remove:TFw
      {\__mathcommand_absorb_add_prime:
       \__mathcommand_absorb_pi:w}
      \__mathcommand_absorb_i_:w}
\cs_new:Npn\__mathcommand_absorb_i_:w{
   \peek_superscript_remove:TFw
      {\__mathcommand_absorb_add_exponent_after:Nn
       \__mathcommand_absorb_finished:}
      \__mathcommand_absorb_finished:}
```

### 5.3.2 Definition of high level commands

```
\NewDocumentCommand\newcommandPIE{ m o  o m }{
   \__xparse_check_definable:nNT {#1} \newcommandPIE
      {
        \cs_if_exist:NTF #1
          {
            \__kernel_msg_error:nnxx { mathcommand } { command-already-defined }
              { \use:nnn \token_to_str:N #1 { } }
              { \token_to_str:N \newcommandPIE }
          }
          { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
      }
}
\NewDocumentCommand\renewcommandPIE{ m o  o m }{
   \__xparse_check_definable:nNT {#1} \renewcommandPIE
      {
        \cs_if_exist:NTF #1
          { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
          {
            \__kernel_msg_error:nnxx { mathcommand } { command-not-yet-defined }
              { \use:nnn \token_to_str:N #1 { } }
              { \token_to_str:N \renewcommandPIE }
          }
      }
}
\NewDocumentCommand\declarecommandPIE{ m o  o m }{
   \__xparse_check_definable:nNT {#1} \declarecommandPIE
        { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
}
\cs_new:Nn\__mathcommand_declarecommandPIE:Nnnn{
   \use:x{
   \exp_not:N\__mathcommand_declarePIE_generic:Nnnn
      \exp_not:N#1
      {\IfNoValueTF{#2}{0}{#2}}
```

```
      {\cs_if_exist:NTF#1
            {\exp_not:N\renewrobustcmd}
            {\exp_not:N\newrobustcmd}
         \exp_not:N#1
         \IfNoValueTF{#2}{}{[#2]}
         \IfNoValueTF{#3}{}{[{\exp_not:n{#3}}]}}
      {\exp_not:n{#4}}
}}
\cs_new_protected:Npn\NewDocumentCommandPIE#1#2#3{
   \__xparse_check_definable:nNT {#1} \NewDocumentCommandPIE
      {
        \cs_if_exist:NTF #1
          {
            \__kernel_msg_error:nnxx { mathcommand } { command-already-defined }
              { \use:nnn \token_to_str:N #1 { } }
              { \token_to_str:N \NewDocumentCommandPIE }
          }
          { \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3} }
      }
}
\cs_new_protected:Npn\RenewDocumentCommandPIE#1#2#3{
   \__xparse_check_definable:nNT {#1} \RenewDocumentCommandPIE
      {
        \cs_if_exist:NTF #1
          { \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3} }
          {
             \__kernel_msg_error:nnxx { xparse } { command-not-yet-defined }
              { \use:nnn \token_to_str:N #1 { } }
              { \token_to_str:N \RenewDocumentCommandPIE }
          }
      }
}
\cs_new_protected:Npn\DeclareDocumentCommandPIE#1#2#3{
   \__xparse_check_definable:nNT {#1} \DeclareDocumentCommandPIE
      {
          \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3}
      }
}
\cs_new_protected:Npn\ProvideDocumentCommandPIE#1#2#3{
   \__xparse_check_definable:nNT {#1} \ProvideDocumentCommandPIE
      {
        \cs_if_exist:NTF #1{}
          {
            \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3}
          }
      }
}
\cs_set:Nn\__mathcommand_DeclareDocumentCommandPIE:Nnn{
```

```
  \group_begin:
    \DeclareDocumentCommand#1{#2}{}
    \int_gset_eq:NN\g_tmpa_int\l__xparse_current_arg_int
  \group_end:
  \__mathcommand_declarePIE_generic:Nnnn
    #1
    {\g_tmpa_int}
    {\DeclareDocumentCommand#1{#2}}
    {#3}
}
```

Control token, number parameters, defining command, code

```
\cs_new:Nn\__mathcommand_declarePIE_generic:Nnnn{
  \int_compare:nNnTF{#2}>{6}
   {\PackageError{mathcommand}
         {At~most~6~parameters~in~PIE~commands~when~defining~`\token_to_str:N#1'}
         {PIE~commands~(mathcommand~package)~do~not~accept~more~than~six~parameters.}}
   {\int_case:nn{#2}
    {{0} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3}
     {1} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4}
     {2} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5}
     {3} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6}
     {4} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7}
     {5} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7##8}
     {6} {\cs_set:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7##8##9}}
   {#4}
  \use:x{
    \exp_not:n{#3}
    {\exp_not:N\__mathcommand_absorb_PIE:nw
        {\exp_not:c{\cs_to_str:N#1~PIE~code}
        \int_case:nn{#2}
        {{0}{}
         {1}{\exp_not:n{{##1}}}
         {2}{\exp_not:n{{##1}{##2}}}
         {3}{\exp_not:n{{##1}{##2}{##3}}}
         {4}{\exp_not:n{{##1}{##2}{##3}{##4}}}
         {5}{\exp_not:n{{##1}{##2}{##3}{##4}{##5}}}
         {6}{\exp_not:n{{##1}{##2}{##3}{##4}{##5}{##6}}}}}}}
  }
}
```

### 5.3.3 Auxiliary functions

```
\def\lettersUppercase{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
\def\lettersLowercase{abcdefghijklmnopqrstuvwxyz}
\xdef\lettersAll{\lettersLowercase\lettersUppercase}
\def\lettersGreekLowercase{\alpha\beta\gamma\delta\epsilon\varepsilon\zeta\eta\theta\vartheta\io
\def\lettersGreekUppercase{\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega}
\xdef\lettersGreekAll{\lettersGreekLowercase\lettersGreekUppercase}

\cs_set_eq:NN\IfEmptyTF\tl_if_empty:nTF
```

```
\cs_new:Npn\EmptyContent#1{
    \tl_if_empty:nTF{#1}{{}}{
        \__mathcommand_EmptyContent:w #1*\end_marker:
    }
}
\cs_new:Npn\__mathcommand_EmptyContent:w #1#2\end_marker:{
    #1*
}
\cs_new:Npn\GetExponent#1{
    \tl_if_empty:nTF{#1}{}{
        \__mathcommand_GetIndexOrExponent:w #1\__end_marker__
    }
}
\cs_new:Npn\GetIndex#1{
    \tl_if_empty:nTF{#1}{}{
        \__mathcommand_GetIndexOrExponent:w #1\__end_marker__
    }
}
\cs_new:Npn\__mathcommand_GetIndexOrExponent:w #1#2#3\__end_marker__{
    #2
}
```

## 5.4 Separating math and text macros

```
\tl_const:Nn\__mathcommand_prefix_math_tl{Math~}
\tl_const:Nn\__mathcommand_prefix_text_tl{Text~}
\tl_const:Nn\__mathcommand_prefix_store_tl{LaTeX}
\cs_new:Nn\__mathcommand_to_mathtl:N{\__mathcommand_prefix_math_tl\cs_to_str:N#1}
\cs_new:Nn\__mathcommand_to_texttl:N{\__mathcommand_prefix_text_tl\cs_to_str:N#1}
\cs_new:Nn\__mathcommand_to_storetl:N{\__mathcommand_prefix_store_tl\cs_to_str:N#1}
\cs_new:Nn\__mathcommand_coretl:N
    {\expandafter\__command_coretl:w\string#1\end_mark}
\cs_new:Npn\__command_coretl:w#1~#2\end_mark{#2}
\cs_new:Npn\__mathcommand_if_exist:NTF
  {\cs_if_exist:NTF}
\cs_new:Npn\__mathcommand_if_exist_math:NTF#1
  {\cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}}
\cs_new:Npn\__mathcommand_if_exist_text:NTF#1
  {\cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}}
\cs_new:Npn\__mathcommand_if_exist_text_or_math:NTF#1
  {\__mathcommand_if_exist_math:NTF#1
      \use_i:nn{\__mathcommand_if_exist_text:NTF#1}}
\cs_new:Nn\__mathcommand_error_if_exist_math:NF{
   \cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}
   {\exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:N#1'~already~exists~in~math~mode}
       {}
   }{#2}
}
```

```
\cs_new:Nn\__mathcommand_error_if_exist_text:NF{
   \cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}
   {\exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:N#1'~already~exists~in~text~mode}
       {}
   }{#2}
}
\cs_new:Nn\__mathcommand_error_if_not_exist_math:NF{
   \cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}
     {#2}
     {\exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:N#1'~does~not~exist~in~math~mode}
       {}
   }
}
\cs_new:Nn\__mathcommand_error_if_not_exist_text:NF{
   \cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}
     {#2}
     {\exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:N#1'~does~not~exist~in~text~mode}
       {}
   }
}
\cs_new:Nn\__mathcommand_error_unknownmath:N{
   \exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:c{\__mathcommand_coretl:N #1}'~does~not~exist~in~math~mode}
       {}
}
\cs_new:Nn\__mathcommand_error_unknowntext:N{
   \exp_args:Nnx\PackageError{}
       {Command~`\token_to_str:c{\__mathcommand_coretl:N #1}'~does~not~exist~in~text~mode}
       {}
}
\cs_new:Nn\__mathcommand_try_math:N{
   \cs_if_exist:NTF#1#1{\__mathcommand_error_unknownmath:N#1}
}
\cs_new:Nn\__mathcommand_try_text:N{
   \cs_if_exist:NTF#1#1{\__mathcommand_error_unknowntext:N#1}
}
```

The macro `\__mathcommand_create_fork:N` takes a control sequence, and creates the forking code that executes either the math branch of the text branch. If this forking code is already present, the command does nothing. However, if some macro was already associated with this control sequence, then it is copied to the math variant, the text variant as well as the stored.

```
\cs_set:Nn\__mathcommand_create_fork:N{
   \let\__mathcommand_tmp_cs\undefined
   \exp_args:NNx
```

```
      \cs_new_protected:Npn\__mathcommand_tmp_cs{
         \exp_not:N\mode_if_math:TF
          {\exp_not:N\__mathcommand_try_math:N\exp_not:c{\__mathcommand_to_mathtl:N#1}}
          {\exp_not:N\__mathcommand_try_text:N\exp_not:c{\__mathcommand_to_texttl:N#1}}
      }
   \cs_if_exist:NTF#1{
     \cs_if_eq:NNTF#1\__mathcommand_tmp_cs
       { }
       { \cs_set_eq:cN{\__mathcommand_to_storetl:N#1}#1
         \cs_set_eq:cN{\__mathcommand_to_mathtl:N#1}#1
         \cs_set_eq:cN{\__mathcommand_to_texttl:N#1}#1
         \cs_set_eq:NN#1\__mathcommand_tmp_cs
       }
     }{ \cs_set_eq:NN#1\__mathcommand_tmp_cs }
}
```

## 5.5 Definition of the high level commands

```
\NewDocumentCommand\declarecommand{m}{
   \__xparse_check_definable:nNT {#1} \declarecommand
   {
     \cs_if_exist:NTF#1
        {\renewcommand#1}
        {\newcommand#1}
   }
}
\newrobustcmd\storecommand[2][\__mathcommand_prefix_store_tl]{
   \__xparse_check_definable:nNT {#2} \storecommand
   {
       \cs_if_exist:NTF#2{
           \cs_set_eq:cN{\__mathcommand_prefix_store_tl\cs_to_str:N#2}#2
       }
       {
           \PackageError{mathcommand}
          {The~command~`\token_to_str:N#2'~does~not~exist~(in~\token_to_str:N\storecommand)}
          {}
       }
   }
}

\NewDocumentCommand\NewDocumentMathCommand{m}{
   \__xparse_check_definable:nNT {#1} \NewDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_math:NF#1
      {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\NewDocumentTextCommand{m}{
   \__xparse_check_definable:nNT {#1} \NewDocumentTextCommand
   {
```

```
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_text:NF#1
      {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_texttl:N#1}}
   }
}
\NewDocumentCommand\newmathcommand{m}{
   \__xparse_check_definable:nNT {#1} \newmathcommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_math:NF#1
      {\exp_args:Nc\newcommand{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\newtextcommand{m}{
   \__xparse_check_definable:nNT {#1} \newtextcommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_text:NF#1
      {\exp_args:Nc\newcommand{\__mathcommand_to_texttl:N#1}}
   }
}
\NewDocumentCommand\RenewDocumentMathCommand{m}{
   \__xparse_check_definable:nNT {#1} \RenewDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_math:NF#1
      {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\RenewDocumentTextCommand{m}{
   \__xparse_check_definable:nNT {#1} \RenewDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_text:NF#1
      {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_texttl:N#1}}
   }
}
\NewDocumentCommand\renewmathcommand{m}{
   \__xparse_check_definable:nNT {#1} \renewmathcommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_math:NF#1
      {\exp_args:Nc\renewcommand{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\renewtextcommand{m}{
   \__xparse_check_definable:nNT {#1} \renewtextcommand
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_text:NF#1
```

```
      {\exp_args:Nc\renewcommand{\__mathcommand_to_texttl:N#1}}
   }
}
\NewDocumentCommand\declaremathcommand{m}{
   \__xparse_check_definable:nNT {#1} \renewmathcommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\declarecommand{\__mathcommand_to_mathtl:N#1}
   }
}
\NewDocumentCommand\declaretextcommand{m}{
   \__xparse_check_definable:nNT {#1} \renewtextcommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\declarecommand{\__mathcommand_to_texttl:N#1}
   }
}

\NewDocumentCommand\DeclareDocumentMathCommand{m}{
   \__xparse_check_definable:nNT {#1} \DeclareDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}
   }
}
\NewDocumentCommand\DeclareDocumentTextCommand{m}{
   \__xparse_check_definable:nNT {#1} \DeclareDocumentTextCommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_texttl:N#1}
   }
}
\NewDocumentCommand\ProvideDocumentMathCommand{mmm}{
   \__xparse_check_definable:nNT {#1} \ProvideDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\ProvideDocumentCommand{\__mathcommand_to_mathtl:N#1}{#2}{#3}
   }
}
\NewDocumentCommand\ProvideDocumentTextCommand{m}{
   \__xparse_check_definable:nNT {#1} \ProvideDocumentTextCommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\ProvideDocumentCommand{\__mathcommand_to_texttl:N#1}
   }
}
```

## 5.6 Definition of the high level combined commands

```
\NewDocumentCommand\NewDocumentMathCommandPIE{m}{
```

```
   \__xparse_check_definable:nNT {#1} \NewDocumentMathCommandPIE
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_math:NF#1
      {\exp_args:Nc\DeclareDocumentCommandPIE{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\newmathcommandPIE{m}{
   \__xparse_check_definable:nNT {#1} \newmathcommandPiE
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_exist_math:NF#1
      {\exp_args:Nc\newcommandPIE{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\RenewDocumentMathCommandPIE{m}{
   \__xparse_check_definable:nNT {#1} \RenewDocumentMathCommandPIE
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_math:NF#1
      {\exp_args:Nc\DeclareDocumentCommandPIE{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\renewmathcommandPIE{m}{
   \__xparse_check_definable:nNT {#1} \renewmathcommandPIE
   {
      \__mathcommand_create_fork:N#1
      \__mathcommand_error_if_not_exist_math:NF#1
      {\exp_args:Nc\renewcommandPIE{\__mathcommand_to_mathtl:N#1}}
   }
}
\NewDocumentCommand\DeclareDocumentMathCommandPIE{m}{
   \__xparse_check_definable:nNT {#1} \DeclareDocumentMathCommand
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}
   }
}
\NewDocumentCommand\declaremathcommandPIE{m}{
   \__xparse_check_definable:nNT {#1} \declaremathcommandPIE
   {
      \__mathcommand_create_fork:N#1
      \exp_args:Nc\declarecommandPIE{\__mathcommand_to_mathtl:N#1}
   }
}

\NewDocumentCommand\ProvideDocumentMathCommandPIE{mmm}{
   \__xparse_check_definable:nNT {#1} \ProvideDocumentMathCommandPIE
   {
      \__mathcommand_create_fork:N#1
```

```
    \exp_args:Nc\ProvideDocumentCommandPIE{\__mathcommand_to_mathtl:N#1}{#2}{#3}
  }
}
```

## 5.7 Looping for command definitions

```
\NewDocumentCommand\LoopCommands{ m ooooooo m }{
  \IfNoValueTF{#2}
    {\cs_set:Nn\__tmp_two:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_two:n{\exp_not:c{#2}}}
  \IfNoValueTF{#3}
    {\cs_set:Nn\__tmp_three:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_three:n{\exp_not:c{#3}}}
  \IfNoValueTF{#4}
    {\cs_set:Nn\__tmp_four:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_four:n{\exp_not:c{#4}}}
  \IfNoValueTF{#5}
    {\cs_set:Nn\__tmp_five:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_five:n{\exp_not:c{#5}}}
  \IfNoValueTF{#6}
    {\cs_set:Nn\__tmp_six:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_six:n{\exp_not:c{#6}}}
  \IfNoValueTF{#7}
    {\cs_set:Nn\__tmp_seven:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_seven:n{\exp_not:c{#7}}}
  \IfNoValueTF{#8}
    {\cs_set:Nn\__tmp_eight:n{\exp_not:c{##1}}}
    {\cs_set:Nn\__tmp_eight:n{\exp_not:c{#8}}}
  %
  \cs_gset:Nn\g_tmpb_cs:nnnnnnnn{#9}
  %
  \cs_gset:Nn\g_tmpa_cs:n{
    \tl_set:Nn\l_tmpa_tl{##1}
    \use:x{
       \exp_not:N\g_tmpb_cs:nnnnnnnn
          {\l_tmpa_tl}
          \__tmp_two:n{\l_tmpa_tl}
          \__tmp_three:n{\l_tmpa_tl}
          \__tmp_four:n{\l_tmpa_tl}
          \__tmp_five:n{\l_tmpa_tl}
          \__tmp_six:n{\l_tmpa_tl}
          \__tmp_seven:n{\l_tmpa_tl}
          \__tmp_eight:n{\l_tmpa_tl}}
    }
 \exp_args:Nx\tl_map_inline:nn{#1}
    {\tl_if_blank:nTF{##1}
      {}
      {\g_tmpa_cs:n{\__mathcommand_getbasename:n{##1}}}
    }
}
```

```
\cs_new:Nn\__mathcommand_getbasename:n{
      \tl_if_single:nTF{#1}{
         \token_if_cs:NTF#1
            {\cs_to_str:N#1}
            {#1}
      }{#1}
}
\ExplSyntaxOff
```