

The `mathcommand` package for L^AT_EX

[version v1.01–2019/05/12]

Thomas Colcombet

May 12, 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

This first version of the package was written in May 2019.

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.

For instance after executing:

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

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

`\macro{a}` yields ‘text code (a)’ while `$_macro{a}$` yields ‘ $math_{(a)}^{code}$ ’.

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

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 L^AT_EX, as in `\c a` which yields ‘*ǎ*’. However, one may want `\c` to represent 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 L^AT_EX 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 does what is automatically made by commands such as `\declarecommand`).

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

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.

This is best explained through an example. After writing:

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

one obtains that

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

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 commands provided for defining *PIE commands* are given in the following list. 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 to control sequences; they may use the extra parameter #1. *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 if by default 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 main text. Hence using `{\alpha\beta}` is equivalent to `{{\alpha}{\beta}}`.

Some extra remarks may be helpful:

- As usual in \TeX/L\TeX , 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 exists. To avoid conflict, one could 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 $\alpha\beta\gamma\delta\epsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\pi\varpi\rho\sigma\tau\upsilon\phi\chi\psi\omega$.
 - `\lettersGreekUppercase` stands for $\Gamma\Delta\Theta\Lambda\Xi\P\Sigma\Upsilon\Phi\Psi\Omega$.
 - `\lettersGreekAll` stands for $\alpha\beta\gamma\delta\epsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\pi\varpi\rho\sigma\tau\upsilon\phi\chi\psi\omega\Gamma\Delta\Theta\Lambda\Xi\P\Sigma\Upsilon\Phi\Psi\Omega$.

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 (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 an extra parameter that is used for accessing the macros `\LaTeXalpha`, ... that are automatically generated by the `\renewmathcommand` macro.

5 Implementation

5.1 README.md

```
<*readme>
1 This directory contains the package
2
3 name: mathcommand
4 license: LaTeX Project Public License version 1.2 or above
5 version: v1.01
6 date: 2019/05/12
7 author: Thomas Colcombet
8 mail: thomas.colcombet@irif.fr
9 web: -
10
11 Purpose:
12 The mathcommand package provides functionalities for defining macros:
13 - that have different behaviors depending on whether in math or text mode,
14 - absorb Primes, Indices, Exponents (PIE) following LaTeX notations and
15   have them as extra parameters usable in the code.
16 The primary objective of this code is to be used together with the knowledge
17 package for a proper handling of mathematical notations.
18
19 Install:
20 It is sufficient to have the file mathcommand.sty accessible by LaTeX.
21 It can be produced by 'make mathcommand.sty' if necessary.
22 The documentation is in the file mathcommand.pdf.
23
24 Content of the file mathcommand-ctan.zip:
25 - README.md: this file generated while compiling mathcommand.ins,
26 - mathcommand.sty: the package file (generated using knowledge.ins)
27 - mathcommand.pdf: the user documentation (generated by compiling
28   mathcommand.dtx)
29 - makefile: the makefile. Use 'make all' to generate mathcommand.sty
30   and knowledge.pdf. It can also: clean the directory, make zip
31   version of the sources, or ready for CTAN.
32 - mathcommand.ins: is the file generating mathcommand.sty and
33   README.md from mathcommand.dtx (using docstrip).
34 - mathcommand.dtx: code and documentation.
35
</readme>
```

5.2 Code preparation

```
<*package>
36 \RequirePackage{expl3}
37 \RequirePackage{etoolbox}
38 \RequirePackage{xparse}
39 \ExplSyntaxOn
```

```

40 \bool_if_exist:NTF\mathcommand_package_loaded_bool
41 \endinput
42 {\bool_new:N\mathcommand_package_loaded_bool
43 \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`.

```

44 \cs_new:Npn\__mathcommand_absorb_PIE:nw#1{
45 \tl_set:Nn\__mathcommand_absorb_finished_tl{#1}
46 \tl_set:Nn\__mathcommand_primes_tl{ }
47 \tl_set:Nn\__mathcommand_index_tl{ }
48 \tl_set:Nn\__mathcommand_exponent_tl{ }
49 \__mathcommand_absorb:w
50 }

```

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.

```

51 \cs_new:Nn\__mathcommand_absorb_finished:{
52 \exp_args:NV\__mathcommand_absorb_finished:\__mathcommand_exponent_tl
53 }
54 \cs_new:Nn\__mathcommand_absorb_finished_{
55 \exp_args:NV\__mathcommand_absorb_finished_\__mathcommand_index_tl
56 }
57 \cs_new:Nn\__mathcommand_absorb_finished_{
58 \exp_args:NV\__mathcommand_absorb_finished_tl\__mathcommand_primes_tl
59 }

60 \cs_new:Npn\peek_subscript_remove:TFw
61 {\peek_charcode_remove:NTF _}
62 \cs_new:Npn\peek_superscript_remove:TFw
63 {\peek_charcode_remove:NTF ^}
64 \cs_new:Npn\peek_prime_remove:TFw
65 {\peek_charcode_remove:NTF '}

66 \cs_new:Nn\__mathcommand_absorb_add_prime:{
67 \tl_put_right:Nn\__mathcommand_primes_tl{'}
68 }

```

```

69 \ExplSyntaxOff
70 \expandafter\def\csname g_tmpa_tl\endcsname{_{}}
71 \ExplSyntaxOn
72 \cs_new:Nx\__mathcommand_absorb_add_index_after:Nn{
73   \exp_not:N\tl_set:Nn\exp_not:N\__mathcommand_index_tl
74     {\g_tmpa_tl{#2}}
75   #1
76 }

77 \cs_new:Nn\__mathcommand_absorb_add_exponent_after:Nn{
78   \tl_set:Nn\__mathcommand_exponent_tl{^{#2}}
79   #1
80 }

81 \cs_new:Npn\__mathcommand_absorb:w{
82   \peek_prime_remove:TFw
83     {\__mathcommand_absorb_add_prime:
84       \__mathcommand_absorb_p:w}
85     \__mathcommand_absorb_:w}
86 \cs_new:Npn\__mathcommand_absorb_:w{
87   \peek_subscript_remove:TFw
88     {\__mathcommand_absorb_add_index_after:Nn
89       \__mathcommand_absorb_i:w}
90     \__mathcommand_absorb__:w}
91 \cs_new:Npn\__mathcommand_absorb__:w{
92   \peek_superscript_remove:TFw
93     {\__mathcommand_absorb_add_exponent_after:Nn
94       \__mathcommand_absorb_e:w}
95     \__mathcommand_absorb_finished:}

96 \cs_new:Npn\__mathcommand_absorb_p:w{
97   \peek_prime_remove:TFw
98     {\__mathcommand_absorb_add_prime:
99       \__mathcommand_absorb_p:w}
100   \__mathcommand_absorb_p_:w}
101 \cs_new:Npn\__mathcommand_absorb_p_:w{
102   \peek_subscript_remove:TFw
103     {\__mathcommand_absorb_add_index_after:Nn
104       \__mathcommand_absorb_pi:w}
105     \__mathcommand_absorb_finished:}

106 \cs_new:Npn\__mathcommand_absorb_pi:w{
107   \peek_prime_remove:TFw
108     {\__mathcommand_absorb_add_prime:
109       \__mathcommand_absorb_pi:w}
110     \__mathcommand_absorb_finished:}

111 \cs_new:Npn\__mathcommand_absorb_e:w{
112   \peek_subscript_remove:TFw
113     {\__mathcommand_absorb_add_index_after:Nn
114       \__mathcommand_absorb_finished:}
115     \__mathcommand_absorb_finished:}

```



```

116 \cs_new:Npn\__mathcommand_absorb_i:w{
117   \peek_prime_remove:TFw
118   {\__mathcommand_absorb_add_prime:
119    \__mathcommand_absorb_pi:w}
120   \__mathcommand_absorb_i:w}

121 \cs_new:Npn\__mathcommand_absorb_i_:w{
122   \peek_superscript_remove:TFw
123   {\__mathcommand_absorb_add_exponent_after:Nn
124    \__mathcommand_absorb_finished:}
125   \__mathcommand_absorb_finished:}

```

5.3.2 Definition of high level commands

```

126 \NewDocumentCommand\newcommandPIE{ m o o m }{
127   \__xparse_check_definable:nNT {#1} \newcommandPIE
128   {
129     \cs_if_exist:NTF #1
130     {
131       \__kernel_msg_error:nxxx { mathcommand } { command-already-defined }
132       { \use:nnn \token_to_str:N #1 { } }
133       { \token_to_str:N \newcommandPIE }
134     }
135     { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
136   }
137 }

138 \NewDocumentCommand\renewcommandPIE{ m o o m }{
139   \__xparse_check_definable:nNT {#1} \renewcommandPIE
140   {
141     \cs_if_exist:NTF #1
142     { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
143     {
144       \__kernel_msg_error:nxxx { mathcommand } { command-not-yet-defined }
145       { \use:nnn \token_to_str:N #1 { } }
146       { \token_to_str:N \renewcommandPIE }
147     }
148   }
149 }

150 \NewDocumentCommand\declarecommandPIE{ m o o m }{
151   \__xparse_check_definable:nNT {#1} \declarecommandPIE
152   { \__mathcommand_declarecommandPIE:Nnnn #1{#2}{#3}{#4} }
153 }

154 \cs_new:Nn\__mathcommand_declarecommandPIE:Nnnn{
155   \use:x{
156     \exp_not:N\__mathcommand_new_generic:Nnnn
157     \exp_not:N#1
158     {\IfNoValueTF{#2}{0}{#2}}
159     {\cs_if_exist:NTF#1
160      {\exp_not:N\renewrobustcmd}
161      {\exp_not:N\newrobustcmd}

```

```

162         \exp_not:N#1
163         \IfNoValueTF{#2}{#2}
164         \IfNoValueTF{#3}{#3}{\exp_not:n{#3}}
165     {\exp_not:n{#4}}
166 }}

167 \cs_new_protected:Npn\NewDocumentCommandPIE#1#2#3{
168     \__xparse_check_definable:nNT {#1} \NewDocumentCommandPIE
169     {
170         \cs_if_exist:NTF #1
171         {
172             \__kernel_msg_error:nxxx { mathcommand } { command-already-defined }
173             { \use:nnn \token_to_str:N #1 { } }
174             { \token_to_str:N \NewDocumentCommandPIE }
175         }
176         { \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3} }
177     }
178 }

179 \cs_new_protected:Npn\RenewDocumentCommandPIE#1#2#3{
180     \__xparse_check_definable:nNT {#1} \RenewDocumentCommandPIE
181     {
182         \cs_if_exist:NTF #1
183         { \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3} }
184         {
185             \__kernel_msg_error:nxxx { xparse } { command-not-yet-defined }
186             { \use:nnn \token_to_str:N #1 { } }
187             { \token_to_str:N \RenewDocumentCommandPIE }
188         }
189     }
190 }

191 \cs_new_protected:Npn\DeclareDocumentCommandPIE#1#2#3{
192     \__xparse_check_definable:nNT {#1} \DeclareDocumentCommandPIE
193     {
194         \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3}
195     }
196 }

197 \cs_new_protected:Npn\ProvideDocumentCommandPIE#1#2#3{
198     \__xparse_check_definable:nNT {#1} \ProvideDocumentCommandPIE
199     {
200         \cs_if_exist:NTF #1{
201             {
202                 \__mathcommand_DeclareDocumentCommandPIE:Nnn #1 {#2} {#3}
203             }
204         }
205     }

206 \cs_set:Nn\__mathcommand_DeclareDocumentCommandPIE:Nnn{
207     \group_begin:
208     \DeclareDocumentCommand#1{#2}{#3}
209     \int_gset_eq:NN\g_tmpa_int\l__xparse_current_arg_int

```

```

210 \group_end:
211 \__mathcommand_new_generic:Nnnn
212   #1
213   {\g_tmpa_int}
214   {\DeclareDocumentCommand#1{#2}}
215   {#3}
216 }

Control token, number parameters, defining command, code
217 \cs_new:Nn\__mathcommand_new_generic:Nnnn{
218   \int_compare:nNnTF{#2}>{6}{
219     {\PackageError{mathcommand}
220       {At~most~6~parameters~in~PIE~commands~when~defining~‘\token_to_str:N#1’}
221       {PIE~commands~(mathcommand~package)~do~not~accept~more~than~six~parameters.}}
222     {\int_case:nn{#2}
223       {0}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3}
224       {1}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4}
225       {2}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5}
226       {3}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6}
227       {4}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7}
228       {5}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7##8}
229       {6}{\cs_new:cpn{\cs_to_str:N#1~PIE~code}##1##2##3##4##5##6##7##8##9}}
230     {#4}
231   \use:x{
232     \exp_not:n{#3}
233     {\exp_not:N\__mathcommand_absorb_PIE:nw
234       {\exp_not:c{\cs_to_str:N#1~PIE~code}
235         \int_case:nn{#2}
236         {{0}}{
237           {1}{\exp_not:n{{##1}}}
238           {2}{\exp_not:n{{##1}{##2}}}
239           {3}{\exp_not:n{{##1}{##2}{##3}}}
240           {4}{\exp_not:n{{##1}{##2}{##3}{##4}}}
241           {5}{\exp_not:n{{##1}{##2}{##3}{##4}{##5}}}
242           {6}{\exp_not:n{{##1}{##2}{##3}{##4}{##5}{##6}}}}}}
243   }
244 }

```

5.3.3 Auxiliary functions

```

245 \def\lettersUppercase{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
246 \def\lettersLowercase{abcdefghijklmnopqrstuvwxyz}
247 \xdef\lettersAll{\lettersLowercase\lettersUppercase}
248 \def\lettersGreekLowercase{\alpha\beta\gamma\delta\epsilon\varepsilon\zeta\eta\theta\vartheta\iota}
249 \def\lettersGreekUppercase{\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega}
250 \xdef\lettersGreekAll{\lettersGreekLowercase\lettersGreekUppercase}

251 \cs_set_eq:NN\IfEmptyTF\tl_if_empty:nTF

252 \cs_new:Npn\EmptyContent#1{
253   \tl_if_empty:nTF{#1}{{}}{
254     \__mathcommand_EmptyContent:w #1*\end_marker:
255   }

```

```

256 }
257 \cs_new:Npn\__mathcommand_EmptyContent:w #1#2\end_marker:{
258     #1*
259 }

260 \cs_new:Npn\GetContent#1{
261     \tl_if_empty:nTF{#1}{}{
262         \__mathcommand_GetContent:w #1*\end_marker:
263     }
264 }
265 \cs_new:Npn\__mathcommand_GetContent:w #1#2\end_marker:{
266     #1*
267 }

```

5.4 Separating math and text macros

```

268 \tl_const:Nn\__mathcommand_prefix_math_tl{Math~}
269 \tl_const:Nn\__mathcommand_prefix_text_tl{Text~}
270 \tl_const:Nn\__mathcommand_prefix_store_tl{LaTeX}

271 \cs_new:Nn\__mathcommand_to_mathtl:N{\__mathcommand_prefix_math_tl\cs_to_str:N#1}
272 \cs_new:Nn\__mathcommand_to_texttl:N{\__mathcommand_prefix_text_tl\cs_to_str:N#1}
273 \cs_new:Nn\__mathcommand_to_storetl:N{\__mathcommand_prefix_store_tl\cs_to_str:N#1}
274 \cs_new:Nn\__mathcommand_coretl:N
275     {\expandafter\__command_coretl:w\string#1\end_mark}
276 \cs_new:Npn\__command_coretl:w#1~#2\end_mark{#2}

277 \cs_new:Npn\__mathcommand_if_exist:NTF
278     {\cs_if_exist:NTF}
279 \cs_new:Npn\__mathcommand_if_exist_math:NTF#1
280     {\cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}}
281 \cs_new:Npn\__mathcommand_if_exist_text:NTF#1
282     {\cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}}
283 \cs_new:Npn\__mathcommand_if_exist_text_or_math:NTF#1
284     {\__mathcommand_if_exist_math:NTF#1
285         \use_i:nn{\__mathcommand_if_exist_text:NTF#1}}

286 \cs_new:Nn\__mathcommand_error_if_exist_math:NF{
287     \cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}
288     {\exp_args:Nnx\PackageError{
289         {Command~'\token_to_str:N#1'~already~exists~in~math~mode}
290     }}
291     }{#2}
292 }

293 \cs_new:Nn\__mathcommand_error_if_exist_text:NF{
294     \cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}
295     {\exp_args:Nnx\PackageError{
296         {Command~'\token_to_str:N#1'~already~exists~in~text~mode}
297     }}
298     }{#2}
299 }

300 \cs_new:Nn\__mathcommand_error_if_not_exist_math:NF{

```

```

301   \cs_if_exist:cTF{\__mathcommand_to_mathtl:N#1}
302     {#2}
303     {\exp_args:Nnx\PackageError{}}
304     {Command~'\token_to_str:N#1'~does~not~exist~in~math~mode}
305     {}
306   }
307 }

308 \cs_new:Nn\__mathcommand_error_if_not_exist_text:NF{
309   \cs_if_exist:cTF{\__mathcommand_to_texttl:N#1}
310     {#2}
311     {\exp_args:Nnx\PackageError{}}
312     {Command~'\token_to_str:N#1'~does~not~exist~in~text~mode}
313     {}
314   }
315 }

316 \cs_new:Nn\__mathcommand_error_unknownmath:N{
317   \exp_args:Nnx\PackageError{}}
318   {Command~'\token_to_str:c{\__mathcommand_coretl:N #1}'~does~not~exist~in~math~mode}
319   {}
320 }

321 \cs_new:Nn\__mathcommand_error_unknowntext:N{
322   \exp_args:Nnx\PackageError{}}
323   {Command~'\token_to_str:c{\__mathcommand_coretl:N #1}'~does~not~exist~in~text~mode}
324   {}
325 }

326 \cs_new:Nn\__mathcommand_try_math:N{
327   \cs_if_exist:NTF#1#1{\__mathcommand_error_unknownmath:N#1}
328 }
329 \cs_new:Nn\__mathcommand_try_text:N{
330   \cs_if_exist:NTF#1#1{\__mathcommand_error_unknowntext:N#1}
331 }

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.

332 \cs_set:Nn\__mathcommand_create_fork:N{
333   \let\__mathcommand_tmp_cs\undefined
334   \exp_args:NNx
335     \cs_new_protected:Npn\__mathcommand_tmp_cs{
336       \exp_not:N\mode_if_math:TF
337         {\exp_not:N\__mathcommand_try_math:N\exp_not:c{\__mathcommand_to_mathtl:N#1}}
338         {\exp_not:N\__mathcommand_try_text:N\exp_not:c{\__mathcommand_to_texttl:N#1}}
339     }
340   \cs_if_exist:NTF#1{
341     \cs_if_eq:NNTF#1\__mathcommand_tmp_cs
342     { }

```

```

343     { \cs_set_eq:cN{\__mathcommand_to_storetl:N#1}#1
344       \cs_set_eq:cN{\__mathcommand_to_mathtl:N#1}#1
345       \cs_set_eq:cN{\__mathcommand_to_texttl:N#1}#1
346       \cs_set_eq:NN#1\__mathcommand_tmp_cs
347     }
348   }{ \cs_set_eq:NN#1\__mathcommand_tmp_cs }
349 }

```

5.5 Definition of the high level commands

```

350 \NewDocumentCommand\declarecommand{m}{
351   \__xparse_check_definable:nNT {#1} \declarecommand
352   {
353     \cs_if_exist:NTF#1
354       {\renewcommand#1}
355       {\newcommand#1}
356   }
357 }
358 \newrobustcmd\storecommand[2][\__mathcommand_prefix_store_tl]{
359   \__xparse_check_definable:nNT {#2} \storecommand
360   {
361     \cs_if_exist:NTF#2{
362       \cs_set_eq:cN{\__mathcommand_prefix_store_tl\cs_to_str:N#2}#2
363     }
364     {
365       \PackageError{mathcommand}
366       {The~command~'\token_to_str:N#2'~does~not~exist~(in~\token_to_str:N\storecommand)}
367       {}
368     }
369   }
370 }
371
372 \NewDocumentCommand\NewDocumentMathCommand{m}{
373   \__xparse_check_definable:nNT {#1} \NewDocumentMathCommand
374   {
375     \__mathcommand_create_fork:N#1
376     \__mathcommand_error_if_exist_math:NF#1
377     {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}}
378   }
379 }
380 \NewDocumentCommand\NewDocumentTextCommand{m}{
381   \__xparse_check_definable:nNT {#1} \NewDocumentTextCommand
382   {
383     \__mathcommand_create_fork:N#1
384     \__mathcommand_error_if_exist_text:NF#1
385     {\exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_texttl:N#1}}
386   }
387 }
388 \NewDocumentCommand\newmathcommand{m}{
389   \__xparse_check_definable:nNT {#1} \newmathcommand
390   {

```

```

391     \_mathcommand_create_fork:N#1
392     \_mathcommand_error_if_exist_math:N#1
393     {\exp_args:Nc\newcommand{\_mathcommand_to_mathtl:N#1}}
394   }
395 }
396 \NewDocumentCommand\newtextcommand{m}{
397   \_xparse_check_definable:nNT {#1} \newtextcommand
398   {
399     \_mathcommand_create_fork:N#1
400     \_mathcommand_error_if_exist_text:N#1
401     {\exp_args:Nc\newcommand{\_mathcommand_to_texttl:N#1}}
402   }
403 }
404 \NewDocumentCommand\RenewDocumentMathCommand{m}{
405   \_xparse_check_definable:nNT {#1} \RenewDocumentMathCommand
406   {
407     \_mathcommand_create_fork:N#1
408     \_mathcommand_error_if_not_exist_math:N#1
409     {\exp_args:Nc\DeclareDocumentCommand{\_mathcommand_to_mathtl:N#1}}
410   }
411 }
412 \NewDocumentCommand\RenewDocumentTextCommand{m}{
413   \_xparse_check_definable:nNT {#1} \RenewDocumentMathCommand
414   {
415     \_mathcommand_create_fork:N#1
416     \_mathcommand_error_if_not_exist_text:N#1
417     {\exp_args:Nc\DeclareDocumentCommand{\_mathcommand_to_texttl:N#1}}
418   }
419 }
420 \NewDocumentCommand\renewmathcommand{m}{
421   \_xparse_check_definable:nNT {#1} \renewmathcommand
422   {
423     \_mathcommand_create_fork:N#1
424     \_mathcommand_error_if_not_exist_math:N#1
425     {\exp_args:Nc\renewcommand{\_mathcommand_to_mathtl:N#1}}
426   }
427 }
428 \NewDocumentCommand\renewtextcommand{m}{
429   \_xparse_check_definable:nNT {#1} \renewtextcommand
430   {
431     \_mathcommand_create_fork:N#1
432     \_mathcommand_error_if_not_exist_text:N#1
433     {\exp_args:Nc\renewcommand{\_mathcommand_to_texttl:N#1}}
434   }
435 }
436 \NewDocumentCommand\declaremathcommand{m}{
437   \_xparse_check_definable:nNT {#1} \renewmathcommand
438   {
439     \_mathcommand_create_fork:N#1
440     \exp_args:Nc\declarecommand{\_mathcommand_to_mathtl:N#1}

```

```

441   }
442 }
443 \NewDocumentCommand\declaretextcommand{m}{
444   \__xparse_check_definable:nNT {#1} \renewtextcommand
445   {
446     \__mathcommand_create_fork:N#1
447     \exp_args:Nc\declarecommand{\__mathcommand_to_texttl:N#1}
448   }
449 }
450
451 \NewDocumentCommand\DeclareDocumentMathCommand{m}{
452   \__xparse_check_definable:nNT {#1} \DeclareDocumentMathCommand
453   {
454     \__mathcommand_create_fork:N#1
455     \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}
456   }
457 }
458 \NewDocumentCommand\DeclareDocumentTextCommand{m}{
459   \__xparse_check_definable:nNT {#1} \DeclareDocumentTextCommand
460   {
461     \__mathcommand_create_fork:N#1
462     \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_texttl:N#1}
463   }
464 }
465 \NewDocumentCommand\ProvideDocumentMathCommand{mmm}{
466   \__xparse_check_definable:nNT {#1} \ProvideDocumentMathCommand
467   {
468     \__mathcommand_create_fork:N#1
469     \exp_args:Nc\ProvideDocumentCommand{\__mathcommand_to_mathtl:N#1}{#2}{#3}
470   }
471 }
472 \NewDocumentCommand\ProvideDocumentTextCommand{m}{
473   \__xparse_check_definable:nNT {#1} \ProvideDocumentTextCommand
474   {
475     \__mathcommand_create_fork:N#1
476     \exp_args:Nc\ProvideDocumentCommand{\__mathcommand_to_texttl:N#1}
477   }
478 }

```

5.6 Definition of the high level combined commands

```

479 \NewDocumentCommand\NewDocumentMathCommandPIE{m}{
480   \__xparse_check_definable:nNT {#1} \NewDocumentMathCommandPIE
481   {
482     \__mathcommand_create_fork:N#1
483     \__mathcommand_error_if_exist_math:NF#1
484     {\exp_args:Nc\DeclareDocumentCommandPIE{\__mathcommand_to_mathtl:N#1}}
485   }
486 }
487 \NewDocumentCommand\newmathcommandPIE{m}{

```



```

488 \__xparse_check_definable:nNT {#1} \newmathcommandPiE
489 {
490   \__mathcommand_create_fork:N#1
491   \__mathcommand_error_if_exist_math:N#1
492   {\exp_args:Nc\newcommandPiE{\__mathcommand_to_mathtl:N#1}}
493 }
494 }
495 \NewDocumentCommand\RenewDocumentMathCommandPiE{m}{
496   \__xparse_check_definable:nNT {#1} \RenewDocumentMathCommandPiE
497   {
498     \__mathcommand_create_fork:N#1
499     \__mathcommand_error_if_not_exist_math:N#1
500     {\exp_args:Nc\DeclareDocumentCommandPiE{\__mathcommand_to_mathtl:N#1}}
501   }
502 }
503 \NewDocumentCommand\renewmathcommandPiE{m}{
504   \__xparse_check_definable:nNT {#1} \renewmathcommandPiE
505   {
506     \__mathcommand_create_fork:N#1
507     \__mathcommand_error_if_not_exist_math:N#1
508     {\exp_args:Nc\renewcommandPiE{\__mathcommand_to_mathtl:N#1}}
509   }
510 }
511 \NewDocumentCommand\DeclareDocumentMathCommandPiE{m}{
512   \__xparse_check_definable:nNT {#1} \DeclareDocumentMathCommand
513   {
514     \__mathcommand_create_fork:N#1
515     \exp_args:Nc\DeclareDocumentCommand{\__mathcommand_to_mathtl:N#1}
516   }
517 }
518 \NewDocumentCommand\declaremathcommandPiE{m}{
519   \__xparse_check_definable:nNT {#1} \declaremathcommandPiE
520   {
521     \__mathcommand_create_fork:N#1
522     \exp_args:Nc\declarecommandPiE{\__mathcommand_to_mathtl:N#1}
523   }
524 }
525
526 \NewDocumentCommand\ProvideDocumentMathCommandPiE{mmm}{
527   \__xparse_check_definable:nNT {#1} \ProvideDocumentMathCommandPiE
528   {
529     \__mathcommand_create_fork:N#1
530     \exp_args:Nc\ProvideDocumentCommandPiE{\__mathcommand_to_mathtl:N#1}{#2}{#3}
531   }
532 }

```

5.7 Looping for command definitions

```

533 \NewDocumentCommand\LoopCommands{ m oooooo m }{
534   \IfNoValueTF{#2}

```

```

535     {\cs_set:Nn\__tmp_two:n{\exp_not:c{##1}}}
536     {\cs_set:Nn\__tmp_two:n{\exp_not:c{#2}}}
537 \IfNoValueTF{#3}
538     {\cs_set:Nn\__tmp_three:n{\exp_not:c{##1}}}
539     {\cs_set:Nn\__tmp_three:n{\exp_not:c{#3}}}
540 \IfNoValueTF{#4}
541     {\cs_set:Nn\__tmp_four:n{\exp_not:c{##1}}}
542     {\cs_set:Nn\__tmp_four:n{\exp_not:c{#4}}}
543 \IfNoValueTF{#5}
544     {\cs_set:Nn\__tmp_five:n{\exp_not:c{##1}}}
545     {\cs_set:Nn\__tmp_five:n{\exp_not:c{#5}}}
546 \IfNoValueTF{#6}
547     {\cs_set:Nn\__tmp_six:n{\exp_not:c{##1}}}
548     {\cs_set:Nn\__tmp_six:n{\exp_not:c{#6}}}
549 \IfNoValueTF{#7}
550     {\cs_set:Nn\__tmp_seven:n{\exp_not:c{##1}}}
551     {\cs_set:Nn\__tmp_seven:n{\exp_not:c{#7}}}
552 \IfNoValueTF{#8}
553     {\cs_set:Nn\__tmp_eight:n{\exp_not:c{##1}}}
554     {\cs_set:Nn\__tmp_eight:n{\exp_not:c{#8}}}
555 %
556 \cs_gset:Nn\g_tmpb_cs:nnnnnnnn{#9}
557 %
558 \cs_gset:Nn\g_tmpa_cs:n{
559   \tl_set:Nn\l_tmpa_tl{##1}
560   \use:x{
561     \exp_not:N\g_tmpb_cs:nnnnnnnn
562     {\l_tmpa_tl}
563     \__tmp_two:n{\l_tmpa_tl}
564     \__tmp_three:n{\l_tmpa_tl}
565     \__tmp_four:n{\l_tmpa_tl}
566     \__tmp_five:n{\l_tmpa_tl}
567     \__tmp_six:n{\l_tmpa_tl}
568     \__tmp_seven:n{\l_tmpa_tl}
569     \__tmp_eight:n{\l_tmpa_tl}
570   }
571 \exp_args:Nx\tl_map_inline:nn{#1}
572   {\tl_if_blank:nTF{##1}
573     {}
574     {\g_tmpa_cs:n{\__mathcommand_getbasename:n{##1}}}}
575   }
576 }
577 \cs_new:Nn\__mathcommand_getbasename:n{
578   \tl_if_single:nTF{#1}{
579     \token_if_cs:NTF#1
580     {\cs_to_str:N#1}
581     {#1}
582   }{#1}
583 }

```

584 \ExplSyntaxOff