

# The `alphalph` package

Heiko Oberdiek\*

2019/12/09 v2.6

## Abstract

The package provides methods to represent numbers with a limited set of symbols. Both  $\text{\LaTeX}$  and plain  $\text{\TeX}$  are supported.

## Contents

<b>1</b>	<b>Documentation</b>	<b>2</b>
1.1	Introduction	2
1.2	Use cases	3
1.2.1	Number system based on symbols	3
1.2.2	Wrap symbols around	3
1.2.3	Multiple symbols	3
1.3	Glossary	4
1.4	Package usage	5
1.5	User commands	5
1.6	Programmer commands	6
1.7	Design principles	6
1.7.1	Number presentation commands	6
1.7.2	General usability	6
<b>2</b>	<b>Implementation</b>	<b>6</b>
2.1	Begin of package	6
2.2	Catcodes	8
2.3	Package loading	9
2.4	$\epsilon$ - $\text{\TeX}$ detection	9
2.5	Help macros	9
2.6	Symbol provider	10
2.6.1	Alphabet	10
2.7	Finding number of symbols	11
2.8	Methods	13
2.8.1	Common methods	13
2.8.2	Method ‘ <code>alph</code> ’	13
2.8.3	Method ‘ <code>wrap</code> ’	14
2.8.4	Method ‘ <code>mult</code> ’	15
2.9	User interface	16

---

\*Please report any issues at <https://github.com/ho-tex/alphalph/issues>

<b>3</b>	<b>Installation</b>	<b>17</b>
3.1	Download	17
3.2	Bundle installation	17
3.3	Package installation	17
3.4	Refresh file name databases	18
3.5	Some details for the interested	18
<b>4</b>	<b>History</b>	<b>18</b>
	[1999/03/19 v0.1]	18
	[1999/04/12 v1.0]	19
	[1999/04/13 v1.1]	19
	[1999/06/26 v1.2]	19
	[2006/02/20 v1.3]	19
	[2006/05/30 v1.4]	19
	[2007/04/11 v1.5]	19
	[2007/09/09 v2.0]	19
	[2008/08/11 v2.1]	19
	[2010/03/01 v2.2]	19
	[2010/04/18 v2.3]	19
	[2011/05/13 v2.4]	20
	[2016/05/16 v2.5]	20
	[2019/12/09 v2.6]	20
<b>5</b>	<b>Index</b>	<b>20</b>

# 1 Documentation

## 1.1 Introduction

L<sup>A</sup>T<sub>E</sub>X counters can be represented in different ways by using presentation commands:

```
\arabic, \roman, \Roman,
\alph, \Alph, \fnsymbol
```

The ranges of supported counter values are more or less restricted. Only `\arabic` can be used with any counter value T<sub>E</sub>X supports.

Presentation command	Supported domain	Ignored values	Error message "Counter too large"
<code>\arabic</code>	<code>-MAX..MAX</code>		
<code>\roman, \Roman</code>	<code>1..MAX</code>	<code>-MAX..0</code>	
<code>\alph, \Alph</code>	<code>1..26</code>	<code>0</code>	<code>-MAX..-1, 27..MAX</code>
<code>\fnsymbol</code>	<code>1..9</code>	<code>0</code>	<code>-MAX..-1, 10..MAX</code>

MAX = 2147483647

Ordinal numbers are often used in documents: numbering of chapters, sections, figures, footnotes and so on. The layouter chooses `\Alph` for chapter numbers and `\fnsymbol` for footnotes. But what can be done if there are more than 26 chapters or more than 10 footnotes? This package `alphalph` allows to define new presentation commands. They rely on a existing command and define presentations for values greater the limits. Three different methods are provided by the package. In the following use cases they are presented.

## 1.2 Use cases

### 1.2.1 Number system based on symbols

Assume you are writing a book and your lecturer demands that chapter numbers must be letters. But you have already 30 chapters and you have only 26 letters?

In the decimal system the situation would be clear. If you run out of digits, you are using more digits to represent a number. This method can also be used for letters. After chapter 26 with Z we use AA, AB, AC, and AD for the remaining chapters.

Happily this package already defines this presentation command:

```
\usepackage{alphalph}
\renewcommand*{\thechapter}{%
  \AlphAlph{\value{chapter}}%
}
```

`\AlphAlph` generates: A, B, C, ..., Z, AA, AB, ...

The other presentation command is `\alphalph` for lowercase letters.

### 1.2.2 Wrap symbols around

Nine footnote symbols are quite a few. Too soon the symbols are consumed and L<sup>A</sup>T<sub>E</sub>X complains with the error “Counter too large”. However, it could be acceptable to start again with the symbols from the beginning, especially if there are less than nine symbols on a page. This could be achieved by a counter reset. But finding the right place can be difficult or needs manual actions. Also a unique counter value can be desirable (e.g. for generating unique anchor/link names). Package `alphalph` allows you to define a macro that implements a “wrap around”, but letting the value of the counter untouched:

```
\usepackage{alphalph}
\makeatletter
\newalphalph{\fnsymbolwrap}[wrap]{\@fnsymbol}{}
\makeatother
\renewcommand*{\thefootnote}{%
  \fnsymbolwrap{\value{footnote}}%
}
```

`\fnsymbolwrap` generates: \* (1), † (2), ‡ (3), ..., †† (9), \* (10), † 11, ...

### 1.2.3 Multiple symbols

L<sup>A</sup>T<sub>E</sub>X’s standard set of footnote symbols contains doubled symbols at the higher positions. Could this principle be generalized? Yes, but first we need a clean footnote symbol list without doubled entries, example:

```
\usepackage{alphalph}
\makeatletter
\newcommand*{\fnsymbolsingle}[1]{%
  \ensuremath{%
    \ifcase#1%
    \or *%
    \or \dagger
    \or \ddagger
    \or \mathsection
    \or \mathparagraph
    \else
    \@ctrerr
  }%
}
```

```

    \fi
  }%
}
\makeatother
\newalphalph{\fnsymbolmult}[mult]{\fnsymbolsingle}{}
\renewcommand*{\thefootnote}{%
  \fnsymbolmult{\value{footnote}}%
}

```

The own definition of `\fnsymbolsingle` has the advantage that this list can easily be modified. Otherwise you can use `\@fnsymbol` directly, because it uses the same first five symbols.

```

\usepackage{alphalph}
\makeatletter
\newalphalph{\fnsymbolmult}[mult]{\@fnsymbol}{5}
\makeatother
\renewcommand*{\thefootnote}{%
  \fnsymbolmult{\value{footnote}}%
}

```

`\fnsymbolmult` generates: \* (1), † (2), ‡ (3), § (4), ¶ (5), \*\* (6), ..., \*\*\*\* 16, †††† 17, ...

The same method can also be used for the chapter problem in the first discussed use case:

```

\usepackage{alphalph}
\makeatletter
\newalphalph{\AlphMult}[mult]{\@Alph}{26}
\makeatother
\renewcommand*{\chapter}{%
  \AlphMult{\value{chapter}}%
}

```

`\AlphMult` then generates AA, BB, CC, and DD for chapters 27–30.

### 1.3 Glossary

**Counter presentation command** is a macro that expects a L<sup>A</sup>T<sub>E</sub>X counter name as argument. Numbers cannot be used. Examples: `\arabic`, `\alph`, `\fnsymbol`.

**Number presentation command** is a macro that expects a number as argument. A number is anything that T<sub>E</sub>X accepts as number including `\value`. Examples: `\alphalph`, `\AlphAlph`, `\alphalph@alph`

However, `\alph` or `\fnsymbol` are not number presentation commands because they expect a counter name as argument. Happily L<sup>A</sup>T<sub>E</sub>X counter presentation commands internally uses number presentation commands with the same name, but prefixed by ‘@’. Thus `\@alph`, `\@fnsymbol` are number presentation commands.

**Symbols provider** is a command that can be used to get a list of symbols. For example, `\@Alph` provides the 26 uppercase letters from ‘A’ to ‘Z’. Basically a symbol provider is a number presentation command, usually with a limited range.

**Number of symbols** is the number of the last symbol slot of a symbol provider. Thus `\@Alph` generates 26 symbols, `\@fnsymbol` provides 9 symbols.

## 1.4 Package usage

The package `alphalph` can be used with both plain `TEX` and `LATEX`:

**plain T<sub>E</sub>X:** `\input alphalph.sty`

**L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>:** `\usepackage{alphalph}`

There aren't any options.

## 1.5 User commands

<code>\AlphAlph {<i>number</i>}</code>
<code>\alphalph {<i>number</i>}</code>

Both macros are number presentation commands that expects a number as argument. `LATEX` counters are used with `\value`.

The macros represents a number by letters. First single letters A..Z are used, then two letters AA..ZZ, three letters AAA...ZZZ, ... follow.

Macro `\AlphAlph` uses uppercase letters, `\alphalph` generates the lowercase variant.

<i>number</i>	<code>\AlphAlph{<i>number</i>}</code>	<code>\alphalph{<i>number</i>}</code>
1	A	a
2	B	b
26	Z	z
27	AA	aa
30	AD	ad
2000	BXX	bxx
3752127	HELLO	hello
10786572	WORLD	world
2147483647	FXSHRXW	fxshrxw

<code>\newalphalph {<i>cmd</i>} [<i>method</i>] [<i>symbols provider</i>] [<i>number of symbols</i>]</code>
---

Macro `\newalphalph` defines `<cmd>` as new number presentation command. Like `\newcommand` an error is thrown, if macro `<cmd>` already exists.

The `<method>` is one of `alph`, `wrap`, or `mult`. The default is `alph`.

As symbol provider a number presentation command can be used, e.g. `\@fnsymbol`, `\@Alph`, or `\alphalph@alph`.

The last argument is the number of symbols. If the argument is empty, then `\newalphalph` tries to find this number itself. `LATEX`'s number presentation commands throw an error message, if the number is too large. This error message is put in a macro `\ctrerr`. Thus `\newalphalph` calls the symbol provider and tests a number by typesetting it in a temporary box. The error macro `\ctrerr` is caught, it proofs that the number is not supported. Also if the width of the result is zero the number is considered as unavailable.

The empty argument is useful for potentially variable lists. However if the end cannot be detected, then the number of symbols must be given. This is also a lot faster. Therefore don't let the argument empty without reason.

## 1.6 Programmer commands

<code>\alphalph@Alph {&lt;number&gt;}</code>
<code>\alphalph@alph {&lt;number&gt;}</code>

They are basically the same as `\@Alph` and `\@alph`. Some languages of package `babel` redefine L<sup>A</sup>T<sub>E</sub>X's macros to include some font setup that breaks expandibility. Therefore `\AlphAlph` and `\alphalph` are based on `\alphalph@Alph` and `\alphalph@alph` to get the letters. The behaviour of these symbol providers for numbers outside the range 1..26 is undefined.

## 1.7 Design principles

### 1.7.1 Number presentation commands

All number presentation commands that this package defines (including `\alphalph` and `\AlphAlph`) have the following properties:

- They are fully expandable. This means that they can safely
  - be written to a file,
  - used in moving arguments (L<sup>A</sup>T<sub>E</sub>X: they are *robust*),
  - used in a `\csname- $\endcsname$`  pair.
- If the argument is zero or negative, the commands expand to nothing like `\romannumeral`.
- The argument is a T<sub>E</sub>X number. Anything that would be accepted by `\number` is a valid argument:
  - explicite constants,
  - macros that expand to a number,
  - count registers, L<sup>A</sup>T<sub>E</sub>X counter can used via `\value`, e. g.:  
`\alphalph{\value{page}}`
  - ...
- $\varepsilon$ -T<sub>E</sub>X's numeric expressions are supported, if  $\varepsilon$ -T<sub>E</sub>X is available. Then `\numexpr` is applied to the argument. Package `\calc`'s expressions are not supported. That would violate the expandibility.

### 1.7.2 General usability

**T<sub>E</sub>X format:** The package does not depend on L<sup>A</sup>T<sub>E</sub>X, it can also be used by plain T<sub>E</sub>X, for example.

**$\varepsilon$ -T<sub>E</sub>X:**  $\varepsilon$ -T<sub>E</sub>X is supported, the macros are shorter and faster. But  $\varepsilon$ -T<sub>E</sub>X's extensions are not requirements. Without  $\varepsilon$ -T<sub>E</sub>X, just the implementation changes. The properties remain unchanged.

## 2 Implementation

### 2.1 Begin of package

```
1 <*package>
```

Reload check, especially if the package is not used with L<sup>A</sup>T<sub>E</sub>X.

```
2 \begingroup\catcode61\catcode48\catcode32=10\relax%
3 \catcode13=5 % ^M
4 \endlinechar=13 %
5 \catcode35=6 % #
6 \catcode39=12 % '
7 \catcode44=12 % ,
8 \catcode45=12 % -
9 \catcode46=12 % .
10 \catcode58=12 % :
11 \catcode64=11 % @
12 \catcode123=1 % {
13 \catcode125=2 % }
14 \expandafter\let\expandafter\x\csname ver@alphalph.sty\endcsname
15 \ifx\x\relax % plain-TeX, first loading
16 \else
17 \def\empty{}%
18 \ifx\x\empty % LaTeX, first loading,
19 % variable is initialized, but \ProvidesPackage not yet seen
20 \else
21 \expandafter\ifx\csname PackageInfo\endcsname\relax
22 \def\x#1#2{%
23 \immediate\write-1{Package #1 Info: #2.}%
24 }%
25 \else
26 \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
27 \fi
28 \x{alphalph}{The package is already loaded}%
29 \aftergroup\endinput
30 \fi
31 \fi
32 \endgroup%
```

Package identification:

```
33 \begingroup\catcode61\catcode48\catcode32=10\relax%
34 \catcode13=5 % ^M
35 \endlinechar=13 %
36 \catcode35=6 % #
37 \catcode39=12 % '
38 \catcode40=12 % (
39 \catcode41=12 % )
40 \catcode44=12 % ,
41 \catcode45=12 % -
42 \catcode46=12 % .
43 \catcode47=12 % /
44 \catcode58=12 % :
45 \catcode64=11 % @
46 \catcode91=12 % [
47 \catcode93=12 % ]
48 \catcode123=1 % {
49 \catcode125=2 % }
50 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
51 \def\x#1#2#3[#4]{\endgroup
52 \immediate\write-1{Package: #3 #4}%
53 \xdef#1{#4}%
54 }%
55 \else
56 \def\x#1#2[#3]{\endgroup
57 #2[#3]}%
```

```

58     \ifx#1\@undefined
59     \xdef#1{#3}%
60     \fi
61     \ifx#1\relax
62     \xdef#1{#3}%
63     \fi
64   }%
65 \fi
66 \expandafter\x\csname ver@alphalph.sty\endcsname
67 \ProvidesPackage{alphalph}%
68 [2019/12/09 v2.6 Convert numbers to letters (HO)]%

```

## 2.2 Catcodes

```

69 \begingroup\catcode61\catcode48\catcode32=10\relax%
70 \catcode13=5 % ^M
71 \endlinechar=13 %
72 \catcode123=1 % {
73 \catcode125=2 % }
74 \catcode64=11 % @
75 \def\x{\endgroup
76   \expandafter\edef\csname AlPh@AtEnd\endcsname{%
77     \endlinechar=\the\endlinechar\relax
78     \catcode13=\the\catcode13\relax
79     \catcode32=\the\catcode32\relax
80     \catcode35=\the\catcode35\relax
81     \catcode61=\the\catcode61\relax
82     \catcode64=\the\catcode64\relax
83     \catcode123=\the\catcode123\relax
84     \catcode125=\the\catcode125\relax
85   }%
86 }%
87 \x\catcode61\catcode48\catcode32=10\relax%
88 \catcode13=5 % ^M
89 \endlinechar=13 %
90 \catcode35=6 % #
91 \catcode64=11 % @
92 \catcode123=1 % {
93 \catcode125=2 % }
94 \def\TMP@EnsureCode#1#2{%
95   \edef\AlPh@AtEnd{%
96     \AlPh@AtEnd
97     \catcode#1=\the\catcode#1\relax
98   }%
99   \catcode#1=#2\relax
100 }
101 \TMP@EnsureCode{33}{12}% !
102 \TMP@EnsureCode{39}{12}% '
103 \TMP@EnsureCode{40}{12}% (
104 \TMP@EnsureCode{41}{12}% )
105 \TMP@EnsureCode{43}{12}% +
106 \TMP@EnsureCode{44}{12}% ,
107 \TMP@EnsureCode{46}{12}% .
108 \TMP@EnsureCode{47}{12}% /
109 \TMP@EnsureCode{59}{12}% ;
110 \TMP@EnsureCode{60}{12}% <
111 \TMP@EnsureCode{62}{12}% >
112 \TMP@EnsureCode{91}{12}% [
113 \TMP@EnsureCode{93}{12}% ]

```



```

114 \TMP@EnsureCode{96}{12}% ‘
115 \TMP@EnsureCode{124}{12}% |
116 \edef\AlPh@AtEnd{\AlPh@AtEnd\noexpand\endinput}

```

## 2.3 Package loading

```

117 \begingroup\expandafter\expandafter\expandafter\endgroup
118 \expandafter\ifx\cname RequirePackage\endcsname\relax
119 \input infwarerr.sty\relax
120 \input intcalc.sty\relax
121 \else
122 \RequirePackage{infwarerr}[2007/09/09]%
123 \RequirePackage{intcalc}[2007/09/09]%
124 \fi

```

## 2.4 $\epsilon$ -TeX detection

```

125 \begingroup\expandafter\expandafter\expandafter\endgroup
126 \expandafter\ifx\cname numexpr\endcsname\relax
127 \catcode124=9 % '!': ignore
128 \catcode43=14 % '+': comment
129 \else
130 \catcode124=14 % '!': comment
131 \catcode43=9 % '+': ignore
132 \fi

```

## 2.5 Help macros

`\AlPh@Error`

```

133 \def\AlPh@Error#1{%
134 \begingroup
135 \escapechar=92 % backslash
136 \@PackageError{alphalph}{#1}\@ehc
137 \endgroup
138 }

```

`\AlPh@ifDefinable`

```

139 \begingroup\expandafter\expandafter\expandafter\endgroup
140 \expandafter\ifx\cname @ifdefinable\endcsname\relax
141 \def\AlPh@ifDefinable#1#2{%
142 \ifcase\ifx#1\undefined\else\ifx#1\relax\else1\fi\fi0 %
143 #2%
144 \else
145 \AlPh@Error{%
146 Command \string#1 already defined%
147 }%
148 \fi
149 }%
150 \else

```

`\AlPh@ifDefinable`

```

151 \let\AlPh@ifDefinable\@ifdefinable
152 \fi

```

`\@ReturnAfterElseFi` `\@ReturnAfterFi` The following commands moves the ‘then’ and ‘else’ part respectively behind the `\if`-construct. This prevents a too deep `\if`-nesting and so a TeX capacity error because of a limited input stack size. I use this trick in several packages, so I

don't prefix these internal commands in order not to have the same macros with different names. (It saves memory.)

```
153 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
154 \long\def\@ReturnAfterFi#1\fi{\fi#1}
```

`\@gobblefour` L<sup>A</sup>T<sub>E</sub>X defines commands for eating arguments. Define `\@gobblefour` if it is not defined (plain T<sub>E</sub>X).

```
155 \expandafter\ifx\csname @gobblefour\endcsname\relax
156 \long\def\@gobblefour#1#2#3#4{}%
157 \fi
```

`AlPh@ifOptArg`

```
158 \begingroup\expandafter\expandafter\expandafter\endgroup
159 \expandafter\ifx\csname kernel@ifnextchar\endcsname\relax
160 \begingroup\expandafter\expandafter\expandafter\endgroup
161 \expandafter\ifx\csname @ifnextchar\endcsname\relax
162 \def\AlPh@ifOptArg#1#2{%
163 \def\AlPh@TempA{#1}%
164 \def\AlPh@TempB{#2}%
165 \futurelet\AlPh@Token\AlPh@ifOptArgNext
166 }%
167 \let\AlPh@BracketLeft=[%]
168 \def\AlPh@ifOptArgNext{%
169 \ifx\AlPh@Token\AlPh@BracketLeft
170 \expandafter\AlPh@TempA
171 \else
172 \expandafter\AlPh@TempB
173 \fi
174 }%
175 \else
176 \def\AlPh@ifOptArg{\@ifnextchar[}%]
177 \fi
178 \else
179 \def\AlPh@ifOptArg{\kernel@ifnextchar[}%]
180 \fi
```

## 2.6 Symbol provider

### 2.6.1 Alphabet

The output of `\alphalph` and `\AlphAlph` should be usable as part of command names (see `\@namedef`, `\csname`, ...). Unhappily some languages of package `babel` redefine L<sup>A</sup>T<sub>E</sub>X's `\@alph` and `\@Alph` in a manner that they cannot be used in expandable context any more. Therefore package `alphalph` provides its own commands.

`\alphalph@Alph` `\alphalph@alph` The two commands `\AlPh@Alph` and `\AlPh@alph` convert a number into a letter (uppercase and lowercase respectively). The character `@` is used as an error symbol, if the number isn't in the range of 1 until 26. Here we need no space after the number `#1`, because the error symbol `@` for the zero case stops scanning the number. This error symbol should not appear anywhere (except for bugs).

```
181 \def\alphalph@Alph#1{%
182 \ifcase#1%
183 @%
184 \or A\or B\or C\or D\or E\or F\or G\or H\or I\or J\or K\or L\or M%
185 \or N\or O\or P\or Q\or R\or S\or T\or U\or V\or W\or X\or Y\or Z%
186 \else
```

```

187   \AlPh@ctrerr
188   @%
189   \fi
190 }
191 \def\alphalph@alph#1{%
192   \ifcase#1%
193   @%
194   \or a\or b\or c\or d\or e\or f\or g\or h\or i\or j\or k\or l\or m%
195   \or n\or o\or p\or q\or r\or s\or t\or u\or v\or w\or x\or y\or z%
196   \else
197     \AlPh@ctrerr
198     @%
199     \fi
200 }

```

`\AlPh@ctrerr` Macro `\AlPh@ctrerr` is used as hook for the algorithm to get the available number of symbols.

```
201 \def\AlPh@ctrerr{}
```

## 2.7 Finding number of symbols

```

\AlPh@GetNumberOfSymbols #1: symbols provider
202 \def\AlPh@GetNumberOfSymbols#1{%
203   \AlPh@TestNumber1!{#1}%
204   \ifAlPh@Unavailable
205     \def\AlPh@Number{0}%
206     \AlPh@Error{No symbols found}%
207   \else
208     \def\AlPh@Number{1}%
209     \AlPh@ExpSearch2!{#1}%
210   \fi
211 }

\ifAlPh@Unavailable
212 \let\ifAlPh@Unavailable\iffalse
213 \def\AlPh@Unavailabletrue{%
214   \global\let\ifAlPh@Unavailable\iftrue
215 }
216 \def\AlPh@Unavailablefalse{%
217   \global\let\ifAlPh@Unavailable\iffalse
218 }

\AlPh@TestNumber #1: number to be tested
#2: symbols provider
219 \def\AlPh@TestNumber#1!#2{%
220   \AlPh@Unavailablefalse
221   \begingroup
222     \setbox0=\hbox{%
223       \begingroup % color
224         \let\@ctrerr\AlPh@Unavailabletrue
225         \let\AlPh@ctrerr\AlPh@Unavailabletrue
226         #2{#1}%
227       \endgroup
228     }%
229   \ifdim\wd0=0pt %
230     \AlPh@Unavailabletrue
231   \fi

```

```

232 \endgroup
233 }

\AlPh@ExpSearch #1: number to be tested
                #2: symbols provider
234 \def\AlPh@ExpSearch#1!#2{%
235 \let\AlPh@Next\relax
236 \AlPh@TestNumber#1!{#2}%
237 \ifAlPh@Unavailable
238 \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
239 \else
240 \def\AlPh@Number{#1}%
241 \ifnum#1>1073741823 %
242 \AlPh@TestNumber2147483647!{#2}%
243 \ifAlPh@Unavailable
244 \AlPh@BinSearch#1!2147483647!{#2}%
245 \else
246 \def\AlPh@Number{0}%
247 \AlPh@Error{%
248 Maximal symbol number not found%
249 }%
250 \fi
251 \else
252 \def\AlPh@Next{%
253 \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}!{#2}%
254 }%
255 \fi
256 \fi
257 \AlPh@Next
258 }

\AlPh@BinSearch #1: available number
                #2: unavailable number, #2 > #1
                #3: symbols provider
259 \def\AlPh@BinSearch#1!#2!#3{%
260 \expandafter\AlPh@ProcessBinSearch
261 \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
262 #1!#2!{#3}%
263 }

\AlPh@ProcessBinSearch #1: number to be tested, #2 ≤ #1 ≤ #3
                      #2: available number
                      #3: unavailable number
                      #4: symbols provider
264 \def\AlPh@ProcessBinSearch#1!#2!#3!#4{%
265 \let\AlPh@Next\relax
266 \ifnum#1>#2 %
267 \ifnum#1<#3 %
268 \AlPh@TestNumber#1!{#4}%
269 \ifAlPh@Unavailable
270 \def\AlPh@Next{%
271 \AlPh@BinSearch#2!#1!{#4}%
272 }%
273 \else
274 \def\AlPh@Next{%
275 \AlPh@BinSearch#1!#3!{#4}%
276 }%

```

```

277     \fi
278   \else
279     \def\AlPh@Number{#2}%
280   \fi
281 \else
282   \def\AlPh@Number{#2}%
283 \fi
284 \AlPh@Next
285 }

```

## 2.8 Methods

The names of method macros start with `\AlPh@Method`. These macros do the main job in converting a number to its representation. A method command is called with three arguments. The first argument is the number of symbols. The second argument is the basic macro for converting a number with limited number range. The last parameter is the number that needs converting.

### 2.8.1 Common methods

```

\AlPh@CheckPositive #1: number to be checked #2: continuation macro
#3: number of symbols (hidden here)
#4: symbol provider (hidden here)
286 \def\AlPh@CheckPositive#1!#2{%
287   \ifnum#1<1 %
288     \expandafter\@gobblefour
289   \fi
290   #2{#1}%
291 }

```

### 2.8.2 Method ‘alph’

```

\AlPh@Method@alph #1: number of symbols
#2: symbols provider
#3: number to be converted
292 \def\AlPh@Method@alph#1#2#3{%
293   \expandafter\AlPh@CheckPositive
294 |   \number#3!%
295 +   \the\numexpr#3!%
296   \AlPh@ProcessAlph
297   {#1}{#2}%
298 }

\AlPh@ProcessAlph #1: current number
#2: number of symbols
#3: symbols provider
299 \def\AlPh@ProcessAlph#1#2#3{%
300   \ifnum#1>#2 %
301     \@ReturnAfterElseFi{%
302       \expandafter\AlPh@StepAlph\number
303       \intcalcInc{%
304         \intcalcMod{\intcalcDec{#1}}{#2}%
305       }%
306     \expandafter!\number
307     \intcalcDiv{\intcalcDec{#1}}{#2}%
308     !{#2}{#3}%

```

```

309   }%
310   \else
311     \@ReturnAfterFi{%
312       #3{#1}%
313     }%
314   \fi
315 }

```

```

\AlPh@StepAlph #1: current last digit
               #2: new current number
               #3: number of symbols
               #4: symbols provider
316 \def\AlPh@StepAlph#1!#2!#3#4{%
317   \AlPh@ProcessAlph{#2}{#3}{#4}%
318   #4{#1}%
319 }

```

### 2.8.3 Method ‘wrap’

```

\AlPh@Method@wrap #1: number of symbols
                  #2: symbols provider
                  #3: number to be converted
320 \def\AlPh@Method@wrap#1#2#3{%
321   \expandafter\AlPh@CheckPositive
322 |   \number#3!%
323 +   \the\numexpr#3!%
324   \AlPh@ProcessWrap
325   {#1}{#2}%
326 }

```

```

\AlPh@ProcessWrap #1: number to be converted
                  #2: number of symbols
                  #3: symbols provider
327 \def\AlPh@ProcessWrap#1#2#3{%
328   \ifnum#1>#2 %
329     \@ReturnAfterElseFi{%
330       \expandafter\AlPh@StepWrap\number
331       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
332       !{#3}%
333     }%
334   \else
335     \@ReturnAfterFi{%
336       #3{#1}%
337     }%
338   \fi
339 }

```

```

\AlPh@StepWrap #1: final number
               #2: symbols provider
340 \def\AlPh@StepWrap#1!#2{%
341   #2{#1}%
342 }

```

## 2.8.4 Method ‘mult’

After the number of symbols is exhausted, repetitions of the symbol are used.

$$x := \text{number to be converted}$$

$$n := \text{number of symbols}$$

$$r := \text{repetition length}$$

$$s := \text{symbol slot}$$

$$r = ((x - 1) \div n) + 1$$

$$s = ((x - 1) \bmod n) + 1$$

```

\AlPh@Method@mult #1: number of symbols
#2: symbols provider
#3: number to be converted
343 \def\AlPh@Method@mult#1#2#3{%
344   \expandafter\AlPh@CheckPositive
345 |   \number#3!%
346 +   \the\numexpr#3!%
347   \AlPh@ProcessMult
348   {#1}{#2}%
349 }

\AlPh@ProcessMult #1: number to be converted
#2: number of symbols
#3: symbols provider
350 \def\AlPh@ProcessMult#1#2#3{%
351   \ifnum#1>#2 %
352     \@ReturnAfterElseFi{%
353       \expandafter\AlPh@StepMult\romannumeral
354       \intcalcInc{\intcalcDiv{\intcalcDec{#1}}{#2}}%
355       000%
356       \expandafter!\number
357       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
358       !{#3}%
359     }%
360   \else
361     \@ReturnAfterFi{%
362       #3{#1}%
363     }%
364   \fi
365 }

\AlPh@StepMult #1#2: repetitions coded as list of character ‘m’
#3: symbol slot
#4: symbols provider
366 \def\AlPh@StepMult#1#2!#3!#4{%
367   \ifx\\#2\\%
368   \else
369     \@ReturnAfterFi{%
370       \AlPh@StepMult#2!#3!{#4}%
371     }%
372   \fi
373   #4{#3}%
374 }

```

## 2.9 User interface

`\newalphalph` Macro `\newalphalph` had three arguments in versions below 2.0. For the new method argument we use an optional argument an first position.

```
#1: cmd
[#2]: method name: alph (default), wrap, mult
hash-ok #3: symbols provider
#4: number of symbols
375 \AlPh@ifDefinable\newalphalph{%
376   \def\newalphalph#1{%
377     \AlPh@ifOptArg{%
378       \AlPh@newalphalph{#1}%
379     }{%
380       \AlPh@newalphalph{#1}[alph]%
381     }%
382   }%
383 }
```

```
\AlPh@newalphalph #1: cmd #2: method name
#3: symbols provider
#4: number of symbols
384 \def\AlPh@newalphalph#1[#2]#3#4{%
385   \begingroup\expandafter\expandafter\expandafter\endgroup
386   \expandafter\ifx\csname AlPh@Method@#2\endcsname\relax
387     \AlPh@Error{%
388       Unknown method %
389 |       '#2'%
390 +       '\detokenize{#2}'%
391     }%
392   \else
393     \ifx\#4\%
394       \AlPh@GetNumberOfSymbols{#3}%
395       \ifcase\AlPh@Number
396       \else
397         \begingroup
398         \escapechar=92 % backslash
399         \@PackageInfo{alphalph}{%
400           Number of symbols for \string#1 is \AlPh@Number
401         }%
402       \endgroup
403       \expandafter\AlPh@NewAlphAlph
404       \csname AlPh@Method@#2\expandafter\endcsname
405       \AlPh@Number!{#1}{#3}%
406     \fi
407   \else
408     \expandafter\AlPh@NewAlphAlph
409     \csname AlPh@Method@#2\expandafter\endcsname
410 |     \number#4!%
411 +     \the\numexpr#4!%
412     {#1}{#3}%
413   \fi
414 \fi
415 }%
```

```
\AlPh@NewAlphAlph #1: method macro
#2: number of symbols
#3: cmd
#4: symbols provider
```



```

416 \def\AlPh@NewAlphAlph#1#2!#3#4{%
417   \AlPh@IfDefinable#3{%
418     \ifnum#2>0 %
419       \def#3{#1{#2}{#4}}%
420     \else
421       \AlPh@Error{%
422         Definition of \string#3 failed,\MessageBreak
423         because number of symbols (#2) is not positive%
424       }%
425     \fi
426   }%
427 }

```

`\AlphAlph`

```
428 \newalphalph\AlphAlph\alphalph@Alph{26}
```

`\alphalph`

```
429 \newalphalph\alphalph\alphalph@alph{26}
```

```
430 \AlPh@AtEnd%
```

```
431 \</package>
```

## 3 Installation

### 3.1 Download

**Package.** This package is available on CTAN<sup>1</sup>:

[CTAN:macros/latex/contrib/alphalph/alphalph.dtx](#) The source file.

[CTAN:macros/latex/contrib/alphalph/alphalph.pdf](#) Documentation.

**Bundle.** All the packages of the bundle ‘alphalph’ are also available in a TDS compliant ZIP archive. There the packages are already unpacked and the documentation files are generated. The files and directories obey the TDS standard.

[CTAN:install/macros/latex/contrib/alphalph.tds.zip](#)

*TDS* refers to the standard “A Directory Structure for T<sub>E</sub>X Files” ([CTAN:pkg/tds](#)). Directories with `texmf` in their name are usually organized this way.

### 3.2 Bundle installation

**Unpacking.** Unpack the `alphalph.tds.zip` in the TDS tree (also known as `texmf` tree) of your choice. Example (linux):

```
unzip alphalph.tds.zip -d ~/texmf
```

### 3.3 Package installation

**Unpacking.** The `.dtx` file is a self-extracting docstrip archive. The files are extracted by running the `.dtx` through plain T<sub>E</sub>X:

```
tex alphalph.dtx
```

---

<sup>1</sup>[CTAN:pkg/alphalph](#)

**TDS.** Now the different files must be moved into the different directories in your installation TDS tree (also known as `texmf` tree):

```
alphalph.sty → tex/generic/alphalph/alphalph.sty
alphalph.pdf → doc/latex/alphalph/alphalph.pdf
alphalph.dtx → source/latex/alphalph/alphalph.dtx
```

If you have a `docstrip.cfg` that configures and enables `docstrip`'s TDS installing feature, then some files can already be in the right place, see the documentation of `docstrip`.

### 3.4 Refresh file name databases

If your  $\TeX$  distribution ( $\TeX$ Live, `mikTeX`, ...) relies on file name databases, you must refresh these. For example,  $\TeX$ Live users run `texhash` or `mktexlsr`.

### 3.5 Some details for the interested

**Unpacking with  $\LaTeX$ .** The `.dtx` chooses its action depending on the format:

**plain  $\TeX$ :** Run `docstrip` and extract the files.

**$\LaTeX$ :** Generate the documentation.

If you insist on using  $\LaTeX$  for `docstrip` (really, `docstrip` does not need  $\LaTeX$ ), then inform the autodetect routine about your intention:

```
latex \let\install=y\input{alphalph.dtx}
```

Do not forget to quote the argument according to the demands of your shell.

**Generating the documentation.** You can use both the `.dtx` or the `.drv` to generate the documentation. The process can be configured by the configuration file `ltxdoc.cfg`. For instance, put this line into this file, if you want to have A4 as paper format:

```
\PassOptionsToClass{a4paper}{article}
```

An example follows how to generate the documentation with `pdf $\LaTeX$` :

```
pdflatex alphalph.dtx
makeindex -s gind.ist alphalph.idx
pdflatex alphalph.dtx
makeindex -s gind.ist alphalph.idx
pdflatex alphalph.dtx
```

## 4 History

[1999/03/19 v0.1]

- The first version was built as a response to a [question](#)<sup>2</sup> of Will Douglas<sup>3</sup> and the [request](#)<sup>4</sup> of Donald Arsenau<sup>5</sup>, published in the newsgroup `comp.text.tex`: “[Re: alph counters > 26](#)”<sup>6</sup>
- Copyright: LPPL ([CTAN:macros/latex/base/lppl.txt](#))

<sup>2</sup>Url: <https://groups.google.com/group/comp.text.tex/msg/17a74cd721641038>

<sup>3</sup>Will Douglas's email address: [william.douglas@wolfson.ox.ac.uk](mailto:william.douglas@wolfson.ox.ac.uk)

<sup>4</sup>Url: <https://groups.google.com/group/comp.text.tex/msg/8f9768825640315f>

<sup>5</sup>Donald Arsenau's email address: [asnd@reg.triumf.ca](mailto:asnd@reg.triumf.ca)

<sup>6</sup>Url: <https://groups.google.com/group/comp.text.tex/msg/cec563eef8bf65d0>

**[1999/04/12 v1.0]**

- Documentation added in dtx format.
- $\varepsilon$ -TeX support added.

**[1999/04/13 v1.1]**

- Minor documentation change.
- First CTAN release.

**[1999/06/26 v1.2]**

- First generic code about `\ProvidesPackage` improved.
- Documentation: Installation part revised.

**[2006/02/20 v1.3]**

- Reload check (for plain TeX)
- New DTX framework.
- LPPL 1.3

**[2006/05/30 v1.4]**

- `\newalphalph` added.

**[2007/04/11 v1.5]**

- Line ends sanitized.

**[2007/09/09 v2.0]**

- New implementation that uses package `\intcalc`. This removes the dependency on  $\varepsilon$ -TeX.
- `\newalphalph` is extended to support new methods ‘wrap’ and ‘multi’.
- Documentation rewritten.

**[2008/08/11 v2.1]**

- Code is not changed.
- URLs updated from `www.dejanews.com` to `groups.google.com`.

**[2010/03/01 v2.2]**

- Compatibility with `iniTeX`.

**[2010/04/18 v2.3]**

- Documentation fixes (Martin Münch).

[2011/05/13 v2.4]

- Documentation fixes (Jim Diamond) and using package hologo for the documentation.
- Catalogue file added.

[2016/05/16 v2.5]

- Documentation updates.

[2019/12/09 v2.6]

- Documentation updates.

## 5 Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; plain numbers refer to the code lines where the entry is used.

Symbols	
<code>\@PackageError</code> .....	136
<code>\@PackageInfo</code> .....	399
<code>\@ReturnAfterElseFi</code> <i>153, 301, 329, 352</i>	
<code>\@ReturnAfterFi</code> <i>153, 311, 335, 361, 369</i>	
<code>\@ctrerr</code> .....	224
<code>\@ehc</code> .....	136
<code>\@gobblefour</code> .....	<i>155, 288</i>
<code>\@ifdefinable</code> .....	151
<code>\@ifnextchar</code> .....	176
<code>\@undefined</code> .....	58, 142
<code>\@</code> .....	367, 393
<b>A</b>	
<code>\aftergroup</code> .....	29
<code>\AlPh@AtEnd</code> .....	95, 96, 116, 430
<code>\AlPh@BinSearch</code> <i>238, 244, 259, 271, 275</i>	
<code>\AlPh@BracketLeft</code> .....	167, 169
<code>\AlPh@CheckPositive</code> <i>286, 293, 321, 344</i>	
<code>\AlPh@ctrerr</code> .....	187, 197, <u>201</u> , 225
<code>\AlPh@Error</code> <i>133, 145, 206, 247, 387, 421</i>	
<code>\AlPh@ExpSearch</code> .....	209, <u>234</u>
<code>\AlPh@GetNumberOfSymbols</code> ...	<u>202</u> , 394
<code>\AlPh@IfDefinable</code> .	<u>139</u> , <u>151</u> , 375, 417
<code>\AlPh@IfOptArg</code> <i>158, 162, 176, 179, 377</i>	
<code>\AlPh@IfOptArgNext</code> .....	165, 168
<code>\AlPh@Method@alph</code> .....	<u>292</u>
<code>\AlPh@Method@mult</code> .....	<u>343</u>
<code>\AlPh@Method@wrap</code> .....	<u>320</u>
<code>\AlPh@NewAlphAlph</code> ....	403, 408, <u>416</u>
<code>\AlPh@newalphalph</code> ....	378, 380, <u>384</u>
<code>\AlPh@Next</code> .....	<i>235, 252, 257, 265, 270, 274, 284</i>
<code>\AlPh@Number</code> .....	205, 208, 238, <i>240, 246, 279, 282, 395, 400, 405</i>
<code>\AlPh@ProcessAlph</code> ....	<i>296, 299, 317</i>
<code>\AlPh@ProcessBinSearch</code> ....	260, <u>264</u>
<code>\AlPh@ProcessMult</code> .....	347, <u>350</u>
<code>\AlPh@ProcessWrap</code> .....	324, <u>327</u>
<code>\AlPh@StepAlph</code> .....	302, <u>316</u>
<code>\AlPh@StepMult</code> .....	353, <u>366</u>
<code>\AlPh@StepWrap</code> .....	330, <u>340</u>
<code>\AlPh@TempA</code> .....	163, 170
<code>\AlPh@TempB</code> .....	164, 172
<code>\AlPh@TestNumber</code> <i>203, 219, 236, 242, 268</i>	
<code>\AlPh@Token</code> .....	165, 169
<code>\AlPh@Unavailablefalse</code> ....	216, 220
<code>\AlPh@Unavailabletrue</code> .....	<i>213, 224, 225, 230</i>
<code>\AlphAlph</code> .....	5, <u>428</u>
<code>\alphalph</code> .....	<u>429</u>
<code>\alphalph@Alph</code> .....	6, <u>181</u> , 428
<code>\alphalph@alph</code> .....	<u>181</u> , 429
<b>C</b>	
<code>\catcode</code> .....	2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 69, 70, 72, 73, 74, 78, 79, 80, 81, 82, 83, 84, 87, 88, 90, 91, 92, 93, 97, 99, 127, 128, 130, 131
<code>\csname</code> 14, 21, 50, 66, 76, 118, 126, 140, 155, 159, 161, 386, 404, 409	
<b>D</b>	
<code>\detokenize</code> .....	390
<b>E</b>	
<code>\empty</code> .....	17, 18
<code>\endcsname</code> 14, 21, 50, 66, 76, 118, 126, 140, 155, 159, 161, 386, 404, 409	
<code>\endinput</code> .....	29, 116

<code>\endlinechar</code> . . . . .	4, 35, 71, 77, 89		
<code>\escapechar</code> . . . . .	135, 398		
<b>F</b>		<b>M</b>	
<code>\futurelet</code> . . . . .	165	<code>\MessageBreak</code> . . . . .	422
<b>H</b>		<b>N</b>	
<code>\hbox</code> . . . . .	222	<code>\newalphalph</code> . . . . .	5, 375, 428, 429
<b>I</b>		<code>\number</code> . . . . .	253, 261, 294, 302, 306, 322, 330, 345, 356, 410
<code>\ifAlPh@Unavailable</code> . . . . .	204, 212, 237, 243, 269	<code>\numexpr</code> . . . . .	295, 323, 346, 411
<code>\ifcase</code> . . . . .	142, 182, 192, 395	<b>P</b>	
<code>\ifdim</code> . . . . .	229	<code>\PackageInfo</code> . . . . .	26
<code>\iffalse</code> . . . . .	212, 217	<code>\ProvidesPackage</code> . . . . .	19, 67
<code>\ifnum</code> . . . . .	241, 266, 267, 287, 300, 328, 351, 418	<b>R</b>	
<code>\iftrue</code> . . . . .	214	<code>\RequirePackage</code> . . . . .	122, 123
<code>\ifx</code> . . . . .	15, 18, 21, 50, 58, 61, 118, 126, 140, 142, 155, 159, 161, 169, 367, 386, 393	<code>\romannumeral</code> . . . . .	353
<code>\immediate</code> . . . . .	23, 52	<b>S</b>	
<code>\input</code> . . . . .	119, 120	<code>\setbox</code> . . . . .	222
<code>\intcalcAdd</code> . . . . .	261	<b>T</b>	
<code>\intcalcDec</code> . . . . .	304, 307, 331, 354, 357	<code>\the</code> . . . . .	77, 78, 79, 80, 81, 82, 83, 84, 97, 295, 323, 346, 411
<code>\intcalcDiv</code> . . . . .	307, 354	<code>\TMP@EnsureCode</code> . . . . .	94, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115
<code>\intcalcInc</code> . . . . .	303, 331, 354, 357	<b>W</b>	
<code>\intcalcMod</code> . . . . .	304, 331, 357	<code>\wd</code> . . . . .	229
<code>\intcalcShl</code> . . . . .	253	<code>\write</code> . . . . .	23, 52
<code>\intcalcShr</code> . . . . .	261	<b>X</b>	
<b>K</b>		<code>\x</code> 14, 15, 18, 22, 26, 28, 51, 56, 66, 75, 87	
<code>\kernel@ifnextchar</code> . . . . .	179		