

# The alphalph package

Heiko Oberdiek  
<oberdiek@uni-freiburg.de>

2008/08/11 v2.1

## Abstract

The package provides methods to represent numbers with a limited set of symbols. Both L<sup>A</sup>T<sub>E</sub>X and plain-T<sub>E</sub>X are supported.

## Contents

<b>1</b>	<b>Documentation</b>	<b>2</b>
1.1	Introduction	2
1.2	Use cases	2
1.2.1	Number system based on symbols	2
1.2.2	Wrap symbols around	3
1.2.3	Multiple symbols	3
1.3	Glossary	4
1.4	Package usage	4
1.5	User commands	4
1.6	Programmer commands	5
1.7	Design principles	5
1.7.1	Number presentation commands	5
1.7.2	General usability	6
<b>2</b>	<b>Implementation</b>	<b>6</b>
2.1	Begin of package	6
2.2	Catcodes	7
2.3	Package loading	8
2.4	$\varepsilon$ -T <sub>E</sub> X detection	8
2.5	Help macros	8
2.6	Symbol provider	9
2.6.1	Alphabet	9
2.7	Finding number of symbols	10
2.8	Methods	12
2.8.1	Common methods	12
2.8.2	Method ‘alph’	12
2.8.3	Method ‘wrap’	13
2.8.4	Method ‘mult’	13
2.9	User interface	14
<b>3</b>	<b>Test</b>	<b>15</b>
3.1	Catcode checks for loading	15
<b>4</b>	<b>Macro tests</b>	<b>17</b>

<b>5</b>	<b>Installation</b>	<b>21</b>
5.1	Download . . . . .	21
5.2	Bundle installation . . . . .	21
5.3	Package installation . . . . .	21
5.4	Refresh file name databases . . . . .	21
5.5	Some details for the interested . . . . .	22
<b>6</b>	<b>History</b>	<b>22</b>
[1999/03/19 v0.1]	. . . . .	22
[1999/04/12 v1.0]	. . . . .	22
[1999/04/13 v1.1]	. . . . .	22
[1999/06/26 v1.2]	. . . . .	23
[2006/02/20 v1.3]	. . . . .	23
[2006/05/30 v1.4]	. . . . .	23
[2007/04/11 v1.5]	. . . . .	23
[2007/09/09 v2.0]	. . . . .	23
[2008/08/11 v2.1]	. . . . .	23
<b>7</b>	<b>Index</b>	<b>23</b>

# 1 Documentation

## 1.1 Introduction

$\LaTeX$  counter 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  $\TeX$  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

Asume 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 be also be used for letters. After chapter 26 with Z we us `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-T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X:

**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 {⟨number⟩}</code> <code>\alphalph {⟨number⟩}</code>
--

Both macros are number presentation commands that expects a number as argument. L<sup>A</sup>T<sub>E</sub>X 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.

$\langle number \rangle$	<code>\AlphAlph{<math>\langle number \rangle</math>}</code>	<code>\alphalph{<math>\langle number \rangle</math>}</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 {<math>\langle cmd \rangle</math>} [<math>\langle method \rangle</math>] {<math>\langle symbols provider \rangle</math>} {<math>\langle number of symbols \rangle</math>}</code>
---

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

The  $\langle method \rangle$  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. L<sup>A</sup>T<sub>E</sub>X'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 {<math>\langle number \rangle</math>}</code> <code>\alphalph@alph {<math>\langle number \rangle</math>}</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 expandability. 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  $\text{\TeX}$  number. Anything that would be accepted by `\number` is a valid argument:
  - explicite constants,
  - macros that expand to a number,
  - count registers,  $\text{\LaTeX}$  counter can used via `\value`, e.g.: `\alphalph{\value{page}}`
  - ...
- $\varepsilon\text{-TeX}$ 's numeric expressions are supported, if  $\varepsilon\text{-TeX}$  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

**$\text{\TeX}$  format:** The package does not depend on  $\text{\LaTeX}$ , it can also be used by plain- $\text{\TeX}$ , for example.

**$\varepsilon\text{-TeX}$ :**  $\varepsilon\text{-TeX}$  is supported, the macros are shorter and faster. But  $\varepsilon\text{-TeX}$ 's extensions are not requirements. Without  $\varepsilon\text{-TeX}$ , 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  $\text{\LaTeX}$ .

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

Package identification:

```

30 \begingroup
31 \catcode35 6 % #
32 \catcode40 12 % (
33 \catcode41 12 % )
34 \catcode44 12 % ,
35 \catcode45 12 % -
36 \catcode46 12 % .
37 \catcode47 12 % /
38 \catcode58 12 % :
39 \catcode64 11 % @
40 \catcode91 12 % [
41 \catcode93 12 % ]
42 \catcode123 1 % {
43 \catcode125 2 % }
44 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
45 \def\x#1#2#3[#4]{\endgroup
46 \immediate\write-1{Package: #3 #4}%
47 \xdef#1{#4}%
48 }%
49 \else
50 \def\x#1#2[#3]{\endgroup
51 #2[#3]}%
52 \ifx#1\@undefined
53 \xdef#1{#3}%
54 \fi
55 \ifx#1\relax
56 \xdef#1{#3}%
57 \fi
58 }%
59 \fi
60 \expandafter\x\csname ver@alphalph.sty\endcsname
61 \ProvidesPackage{alphalph}%
62 [2008/08/11 v2.1 Converting numbers to letters (HO)]

```

## 2.2 Catcodes

```

63 \begingroup
64 \catcode123 1 % {
65 \catcode125 2 % }
66 \def\x{\endgroup
67 \expandafter\edef\csname AlPh@AtEnd\endcsname{%
68 \catcode35 \the\catcode35\relax
69 \catcode64 \the\catcode64\relax
70 \catcode123 \the\catcode123\relax
71 \catcode125 \the\catcode125\relax
72 }%
73 }%
74 \x
75 \catcode35 6 % #
76 \catcode64 11 % @
77 \catcode123 1 % {
78 \catcode125 2 % }
79 \def\TMP@EnsureCode#1#2{%
80 \edef\AlPh@AtEnd{%
81 \AlPh@AtEnd
82 \catcode#1 \the\catcode#1\relax
83 }%
84 \catcode#1 #2\relax
85 }
86 \TMP@EnsureCode{33}{12}% !
87 \TMP@EnsureCode{39}{12}% '
88 \TMP@EnsureCode{40}{12}% (
89 \TMP@EnsureCode{41}{12}% )

```

```

90 \TMP@EnsureCode{43}{12}% +
91 \TMP@EnsureCode{44}{12}% ,
92 \TMP@EnsureCode{46}{12}% .
93 \TMP@EnsureCode{47}{12}% /
94 \TMP@EnsureCode{59}{12}% ;
95 \TMP@EnsureCode{60}{12}% <
96 \TMP@EnsureCode{61}{12}% =
97 \TMP@EnsureCode{62}{12}% >
98 \TMP@EnsureCode{91}{12}% [
99 \TMP@EnsureCode{93}{12}% ]
100 \TMP@EnsureCode{96}{12}% '
101 \TMP@EnsureCode{124}{12}% |

```

## 2.3 Package loading

```

102 \begingroup\expandafter\expandafter\expandafter\endgroup
103 \expandafter\ifx\csname RequirePackage\endcsname\relax
104   \input infwarerr.sty\relax
105   \input intcalc.sty\relax
106 \else
107   \RequirePackage{infwarerr}[2007/09/09]%
108   \RequirePackage{intcalc}[2007/09/09]%
109 \fi

```

## 2.4 $\varepsilon$ -TeX detection

```

110 \begingroup\expandafter\expandafter\expandafter\endgroup
111 \expandafter\ifx\csname numexpr\endcsname\relax
112   \catcode124=9 % '!' : ignore
113   \catcode43=14 % '+' : comment
114 \else
115   \catcode124=14 % '!' : comment
116   \catcode43=9 % '+' : ignore
117 \fi

```

## 2.5 Help macros

\AlPh@Error

```

118 \def\AlPh@Error#1{%
119   \begingroup
120     \escapechar=92 % backslash
121     \@PackageError{alphalph}{#1}\@ehc
122   \endgroup
123 }

```

\AlPh@ifDefinable

```

124 \begingroup\expandafter\expandafter\expandafter\endgroup
125 \expandafter\ifx\csname @ifdefinable\endcsname\relax
126   \def\AlPh@ifDefinable#1#2{%
127     \ifcase\ifx#1\@undefined\else\ifx#1\relax\else1\fi\fi0 %
128       #2%
129     \else
130       \AlPh@Error{%
131         Command \string#1 already defined%
132       }%
133     \fi
134   }%
135 \else

```

\AlPh@ifDefinable

```

136   \let\AlPh@ifDefinable\@ifdefinable
137 \fi

```

`\@ReturnAfterElseFi` 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.)

```

138 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
139 \long\def\@ReturnAfterFi#1\fi{\fi#1}

\@gobblefour LATEX defines commands for eating arguments. Define \@gobblefour if it is not defined (plain-TEX).
140 \expandafter\ifx\csname @gobblefour\endcsname\relax
141 \long\def\@gobblefour#1#2#3#4{%
142 \fi

AlPh@ifOptArg
143 \begingroup\expandafter\expandafter\expandafter\endgroup
144 \expandafter\ifx\csname kernel@ifnextchar\endcsname\relax
145 \begingroup\expandafter\expandafter\expandafter\endgroup
146 \expandafter\ifx\csname @ifnextchar\endcsname\relax
147 \def\AlPh@ifOptArg#1#2{%
148 \def\AlPh@TempA{#1}%
149 \def\AlPh@TempB{#2}%
150 \futurelet\AlPh@Token\AlPh@ifOptArgNext
151 }%
152 \let\AlPh@BracketLeft=[%]
153 \def\AlPh@ifOptArgNext{%
154 \ifx\AlPh@Token\AlPh@BracketLeft
155 \expandafter\AlPh@TempA
156 \else
157 \expandafter\AlPh@TempB
158 \fi
159 }%
160 \else
161 \def\AlPh@ifOptArg{\@ifnextchar[]}%
162 \fi
163 \else
164 \def\AlPh@ifOptArg{\kernel@ifnextchar[]}%
165 \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` 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).

`\alphalph@alph`

```

166 \def\alphalph@Alph#1{%
167 \ifcase#1%
168 @%
169 \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%
170 \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%
171 \else
172 \AlPh@ctrerr

```

```

173   @%
174   \fi
175 }
176 \def\alphalph@alph#1{%
177   \ifcase#1%
178     @%
179     \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%
180     \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%
181   \else
182     \AlPh@ctrerr
183   @%
184   \fi
185 }

```

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

```

186 \def\AlPh@ctrerr{}

```

## 2.7 Finding number of symbols

`\AlPh@GetNumberOfSymbols` #1: symbols provider

```

187 \def\AlPh@GetNumberOfSymbols#1{%
188   \AlPh@TestNumber1!{#1}%
189   \ifAlPh@Unavailable
190     \def\AlPh@Number{0}%
191     \AlPh@Error{No symbols found}%
192   \else
193     \def\AlPh@Number{1}%
194     \AlPh@ExpSearch2!{#1}%
195   \fi
196 }

```

`\ifAlPh@Unavailable`

```

197 \newif\ifAlPh@Unavailable
198 \def\AlPh@Unavailabletrue{%
199   \global\let\ifAlPh@Unavailable\iftrue
200 }
201 \def\AlPh@Unavailablefalse{%
202   \global\let\ifAlPh@Unavailable\iffalse
203 }

```

`\AlPh@TestNumber` #1: number to be tested

#2: symbols provider

```

204 \def\AlPh@TestNumber#1!#2{%
205   \AlPh@Unavailablefalse
206   \begingroup
207     \setbox0=\hbox{%
208       \begingroup % color
209         \let\@ctrerr\AlPh@Unavailabletrue
210         \let\AlPh@ctrerr\AlPh@Unavailabletrue
211         #2{#1}%
212       \endgroup
213     }%
214     \ifdim\wd0=0pt %
215       \AlPh@Unavailabletrue
216     \fi
217   \endgroup
218 }

```

`\AlPh@ExpSearch` #1: number to be tested

#2: symbols provider

```

219 \def\AlPh@ExpSearch#1!#2{%
220   \let\AlPh@Next\relax
221   \AlPh@TestNumber#1!{#2}%
222   \ifAlPh@Unavailable
223     \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
224   \else
225     \def\AlPh@Number{#1}%
226     \ifnum#1>1073741823 %
227       \AlPh@TestNumber2147483647!{#2}%
228       \ifAlPh@Unavailable
229         \AlPh@BinSearch#1!2147483647!{#2}%
230       \else
231         \def\AlPh@Number{0}%
232         \AlPh@Error{%
233           Maximal symbol number not found%
234         }%
235       \fi
236     \else
237       \def\AlPh@Next{%
238         \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}!{#2}%
239       }%
240     \fi
241   \fi
242   \AlPh@Next
243 }

```

```

\AlPh@BinSearch #1: available number
#2: unavailable number, #2 > #1
#3: symbols provider
244 \def\AlPh@BinSearch#1!#2!#3{%
245   \expandafter\AlPh@ProcessBinSearch
246   \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
247   #1!#2!{#3}%
248 }

```

```

\AlPh@ProcessBinSearch #1: number to be tested, #2 ≤ #1 ≤ #3
#2: available number
#3: unavailable number
#4: symbols provider
249 \def\AlPh@ProcessBinSearch#1!#2!#3!#4{%
250   \let\AlPh@Next\relax
251   \ifnum#1>#2 %
252     \ifnum#1<#3 %
253       \AlPh@TestNumber#1!{#4}%
254       \ifAlPh@Unavailable
255         \def\AlPh@Next{%
256           \AlPh@BinSearch#2!#1!{#4}%
257         }%
258       \else
259         \def\AlPh@Next{%
260           \AlPh@BinSearch#1!#3!{#4}%
261         }%
262       \fi
263     \else
264       \def\AlPh@Number{#2}%
265     \fi
266   \else
267     \def\AlPh@Number{#2}%
268   \fi
269   \AlPh@Next
270 }

```

## 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)
271 \def\AlPh@CheckPositive#1!#2{%
272   \ifnum#1<1 %
273     \expandafter\@gobblefour
274   \fi
275   #2{#1}%
276 }
```

### 2.8.2 Method ‘alph’

```
\AlPh@Method@alph #1: number of symbols
#2: symbols provider
#3: number to be converted
277 \def\AlPh@Method@alph#1#2#3{%
278   \expandafter\AlPh@CheckPositive
279 |   \number#3!%
280 +   \the\numexpr#3!%
281   \AlPh@ProcessAlph
282   {#1}{#2}%
283 }

\AlPh@ProcessAlph #1: current number
#2: number of symbols
#3: symbols provider
284 \def\AlPh@ProcessAlph#1#2#3{%
285   \ifnum#1>#2 %
286     \@ReturnAfterElseFi{%
287       \expandafter\AlPh@StepAlph\number
288       \intcalcInc{%
289         \intcalcMod{\intcalcDec{#1}}{#2}%
290       }%
291       \expandafter!\number
292       \intcalcDiv{\intcalcDec{#1}}{#2}%
293       !{#2}{#3}%
294     }%
295   \else
296     \@ReturnAfterFi{%
297       #3{#1}%
298     }%
299   \fi
300 }

\AlPh@StepAlph #1: current last digit
#2: new current number
#3: number of symbols
#4: symbols provider
301 \def\AlPh@StepAlph#1!#2!#3#4{%
302   \AlPh@ProcessAlph{#2}{#3}{#4}%
303   #4{#1}%
304 }
```

### 2.8.3 Method ‘wrap’

```

\AlPh@Method@wrap #1: number of symbols
                  #2: symbols provider
                  #3: number to be converted
305 \def\AlPh@Method@wrap#1#2#3{%
306   \expandafter\AlPh@CheckPositive
307 |   \number#3!%
308 +   \the\numexpr#3!%
309   \AlPh@ProcessWrap
310   {#1}{#2}%
311 }

\AlPh@ProcessWrap #1: number to be converted
                  #2: number of symbols
                  #3: symbols provider
312 \def\AlPh@ProcessWrap#1#2#3{%
313   \ifnum#1>#2 %
314     \@ReturnAfterElseFi{%
315       \expandafter\AlPh@StepWrap\number
316       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
317       !{#3}%
318     }%
319   \else
320     \@ReturnAfterFi{%
321       #3{#1}%
322     }%
323   \fi
324 }

\AlPh@StepWrap #1: final number
               #2: symbols provider
325 \def\AlPh@StepWrap#1!#2{%
326   #2{#1}%
327 }

```

### 2.8.4 Method ‘mult’

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

$$\begin{aligned}
 x &:= \text{number to be converted} \\
 n &:= \text{number of symbols} \\
 r &:= \text{repetition length} \\
 s &:= \text{symbol slot} \\
 r &= ((x - 1) \div n) + 1 \\
 s &= ((x - 1) \bmod n) + 1
 \end{aligned}$$

```

\AlPh@Method@mult #1: number of symbols
                  #2: symbols provider
                  #3: number to be converted
328 \def\AlPh@Method@mult#1#2#3{%
329   \expandafter\AlPh@CheckPositive
330 |   \number#3!%
331 +   \the\numexpr#3!%
332   \AlPh@ProcessMult
333   {#1}{#2}%
334 }

```

```

\AlPh@ProcessMult #1: number to be converted
#2: number of symbols
#3: symbols provider
335 \def\AlPh@ProcessMult#1#2#3{%
336   \ifnum#1>#2 %
337     \@ReturnAfterElseFi{%
338       \expandafter\AlPh@StepMult\romannumeral
339       \intcalcInc{\intcalcDiv{\intcalcDec{#1}}{#2}}%
340       000%
341       \expandafter!\number
342       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
343       !{#3}%
344     }%
345   \else
346     \@ReturnAfterFi{%
347       #3{#1}%
348     }%
349   \fi
350 }

```

```

\AlPh@StepMult #1#2: repetitions coded as list of character ‘m’
#3: symbol slot
#4: symbols provider
351 \def\AlPh@StepMult#1#2!#3!#4{%
352   \ifx\\#2\\%
353   \else
354     \@ReturnAfterFi{%
355       \AlPh@StepMult#2!#3!{#4}%
356     }%
357   \fi
358   #4{#3}%
359 }

```

## 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 in first position.

#1: cmd  
 [#2]: method name: `alph` (default), `wrap`, `mult`  
 hash-ok #3: symbols provider  
 #4: number of symbols

```

360 \AlPh@IfDefinable\newalphalph{%
361   \def\newalphalph#1{%
362     \AlPh@IfOptArg{%
363       \AlPh@newalphalph{#1}%
364     }{%
365       \AlPh@newalphalph{#1}[alph]%
366     }%
367   }%
368 }

```

```

\AlPh@newalphalph #1: cmd #2: method name
#3: symbols provider
#4: number of symbols
369 \def\AlPh@newalphalph#1[#2]#3#4{%
370   \begingroup\expandafter\expandafter\expandafter\endgroup
371   \expandafter\ifx\csname AlPh@Method@#2\endcsname\relax
372     \AlPh@Error{%
373       Unknown method %
374       ‘#2’%
375     + ‘\detokenize{#2}’%

```

```

376     }%
377 \else
378   \ifx\#4\%
379     \AlPh@GetNumberOfSymbols{#3}%
380     \ifcase\AlPh@Number
381     \else
382       \begingroup
383         \escapechar=92 % backslash
384         \@PackageInfo{alphalph}{%
385           Number of symbols for \string#1 is \AlPh@Number
386         }%
387       \endgroup
388       \expandafter\AlPh@NewAlphAlph
389       \csname AlPh@Method@#2\expandafter\endcsname
390       \AlPh@Number!{#1}{#3}%
391     \fi
392   \else
393     \expandafter\AlPh@NewAlphAlph
394     \csname AlPh@Method@#2\expandafter\endcsname
395     \number#4!%
396     \the\numexpr#4!%
397     {#1}{#3}%
398   \fi
399 \fi
400 }%

```

```

\AlPh@NewAlphAlph #1: method macro
                  #2: number of symbols
                  #3: cmd
                  #4: symbols provider
401 \def\AlPh@NewAlphAlph#1#2!#3#4{%
402   \AlPh@IfDefinable#3{%
403     \ifnum#2>0 %
404       \def#3{#1{#2}{#4}}%
405     \else
406       \AlPh@Error{%
407         Definition of \string#3 failed,\MessageBreak
408         because number of symbols (#2) is not positive%
409       }%
410     \fi
411   }%
412 }

```

\AlphAlph

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

\alphalph

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

```
415 \AlPh@AtEnd
```

```
416 \</package>
```

## 3 Test

### 3.1 Catcode checks for loading

```

417 <*test1>
418 \catcode'\{=1 %
419 \catcode'\}=2 %
420 \catcode'\#=6 %
421 \catcode'\@=11 %

```

```

422 \expandafter\ifx\csname count@\endcsname\relax
423   \countdef\count@=255 %
424 \fi
425 \expandafter\ifx\csname @gobble\endcsname\relax
426   \long\def\@gobble#1{}%
427 \fi
428 \expandafter\ifx\csname @firstofone\endcsname\relax
429   \long\def\@firstofone#1{#1}%
430 \fi
431 \expandafter\ifx\csname loop\endcsname\relax
432   \expandafter\@firstofone
433 \else
434   \expandafter\@gobble
435 \fi
436 {%
437   \def\loop#1\repeat{%
438     \def\body{#1}%
439     \iterate
440   }%
441   \def\iterate{%
442     \body
443     \let\next\iterate
444   \else
445     \let\next\relax
446   \fi
447   \next
448 }%
449 \let\repeat=\fi
450 }%
451 \def\RestoreCatcodes{}
452 \count@=0 %
453 \loop
454   \edef\RestoreCatcodes{%
455     \RestoreCatcodes
456     \catcode\the\count@=\the\catcode\count@\relax
457   }%
458 \ifnum\count@<255 %
459   \advance\count@ 1 %
460 \repeat
461
462 \def\RangeCatcodeInvalid#1#2{%
463   \count@=#1\relax
464   \loop
465     \catcode\count@=15 %
466   \ifnum\count@<#2\relax
467     \advance\count@ 1 %
468   \repeat
469 }
470 \expandafter\ifx\csname LoadCommand\endcsname\relax
471   \def\LoadCommand{\input alphalph.sty\relax}%
472 \fi
473 \def\Test{%
474   \RangeCatcodeInvalid{0}{47}%
475   \RangeCatcodeInvalid{58}{64}%
476   \RangeCatcodeInvalid{91}{96}%
477   \RangeCatcodeInvalid{123}{255}%
478   \catcode'\@=12 %
479   \catcode'\=0 %
480   \catcode'\{=1 %
481   \catcode'\}=2 %
482   \catcode'\#=6 %
483   \catcode'\[=12 %

```

```

484 \catcode'\]=12 %
485 \catcode'\%=14 %
486 \catcode'\ =10 %
487 \catcode13=5 %
488 \LoadCommand
489 \RestoreCatcodes
490 }
491 \Test
492 \csname @@end\endcsname
493 \end
494 </test1>

```

## 4 Macro tests

```

495 <*test2>
496 \NeedsTeXFormat{LaTeX2e}
497 \nofiles
498 \documentclass{article}
499 <*noetex>
500 \makeatletter
501 \let\saved@numexpr\numexpr
502 \newcommand*\DisableNumexpr{%
503   \let\numexpr@undefined
504 }
505 \newcommand*\RestoreNumexpr{%
506   \let\numexpr\saved@numexpr
507 }
508 \DisableNumexpr
509 </noetex>
510 \usepackage{alphalph}[2008/08/11]
511 <noetex>\RestoreNumexpr
512 \usepackage{qstest}
513 \IncludeTests{*}
514 \LogTests{log}{*}{*}
515
516 \newcommand*\TestCmd{[3]{%
517   \setbox0=\hbox{%
518     <noetex> \DisableNumexpr
519     \edef\TestString{#1{#2}}%
520     \expandafter\Expect\expandafter{\TestString}{#3}%
521     \edef\TestString{#1{#2} }%
522     \expandafter\Expect\expandafter{\TestString}{#3 }%
523   }%
524   \Expect*\the\wd0}{0.0pt}%
525 }
526
527 \makeatletter
528 \newalphalph\LaTeXAlphAlph\@Alph{26}
529 \newalphalph\LaTeXalphalph\@alph{26}
530 \newalphalph\AlphWrap[wrap]\alphalph@Alph{26}
531 \newalphalph\alphwrap[wrap]\alphalph@alph{26}
532 \newalphalph\LaTeXAlphWrap[wrap]\@Alph{26}
533 \newalphalph\LaTeXalphwrap[wrap]\@alph{26}
534 \def\LastSymbol#1{%
535   \ifx\#1\%
536   \else
537     \@LastSymbol#1\@nil
538   \fi
539 }
540 \def\@LastSymbol#1#2\@nil{%
541   \ifx\#2\%
542     #1%

```

```

543 \else
544   \@LastSymbol#2\@nil
545 \fi
546 }
547 \makeatother
548 \newcommand*\TestAlph}[2]{%
549   \uppercase{\TestCallCmd\AlphAlph{#2}}{#1}%
550   \lowercase{\TestCallCmd\alphalph{#2}}{#1}%
551   \uppercase{\TestCallCmd\LaTeXAlphAlph{#2}}{#1}%
552   \lowercase{\TestCallCmd\LaTeXalphalph{#2}}{#1}%
553   \edef\WrapString{\LastSymbol{#2}}%
554   \expandafter\TestAlphWrap\expandafter{\WrapString}{#1}%
555 }
556 \newcommand*\TestAlphWrap}[2]{%
557   \uppercase{\TestCallCmd\AlphWrap{#1}}{#2}%
558   \lowercase{\TestCallCmd\alphwrap{#1}}{#2}%
559   \uppercase{\TestCallCmd\LaTeXAlphWrap{#1}}{#2}%
560   \lowercase{\TestCallCmd\LaTeXalphwrap{#1}}{#2}%
561 }
562 \newcommand*\TestCallCmd}[3]{%
563   \TestCmd#1{#3}{#2}%
564 }
565 \begin{qstest}{AlphSymbols}{alphalph, AlphAlph, symbols}
566   \TestAlph{1}{a}%
567   \TestAlph{2}{b}%
568   \TestAlph{3}{c}%
569   \TestAlph{4}{d}%
570   \TestAlph{5}{e}%
571   \TestAlph{6}{f}%
572   \TestAlph{7}{g}%
573   \TestAlph{8}{h}%
574   \TestAlph{9}{i}%
575   \TestAlph{10}{j}%
576   \TestAlph{11}{k}%
577   \TestAlph{12}{l}%
578   \TestAlph{13}{m}%
579   \TestAlph{14}{n}%
580   \TestAlph{15}{o}%
581   \TestAlph{16}{p}%
582   \TestAlph{17}{q}%
583   \TestAlph{18}{r}%
584   \TestAlph{19}{s}%
585   \TestAlph{20}{t}%
586   \TestAlph{21}{u}%
587   \TestAlph{22}{v}%
588   \TestAlph{23}{w}%
589   \TestAlph{24}{x}%
590   \TestAlph{25}{y}%
591   \TestAlph{26}{z}%
592 \end{qstest}
593 \begin{qstest}{AlphRange}{alphalph, range}
594   \TestAlph{0}{}%
595   \TestAlph{-1}{}%
596   \TestAlph{-2147483647}{}%
597   \TestAlph{27}{aa}%
598   \TestAlph{28}{ab}%
599   \TestAlph{52}{az}%
600   \TestAlph{53}{ba}%
601   \TestAlph{78}{bz}%
602   \TestAlph{79}{ca}%
603   \TestAlph{702}{zz}%
604   \TestAlph{703}{aaa}%

```

```

605 \TestAlph{2147483647}{fxshrxw}%
606 \end{qstest}
607
608 \makeatletter
609 \newcommand*{\myvocal}{1}{%
610 \ifcase#1X\or A\or E\or I\or O\or U\else Y\fi
611 }
612 \makeatother
613 \newalphalph\vocalsvocal\myvocal{5}
614 \newcommand*{\TestVocal}{%
615 \TestCmd\vocalsvocal
616 }
617 \begin{qstest}{\vocal}{\vocal}
618 \TestVocal{0}{}%
619 \TestVocal{1}{A}%
620 \TestVocal{2}{E}%
621 \TestVocal{3}{I}%
622 \TestVocal{4}{O}%
623 \TestVocal{5}{U}%
624 \TestVocal{6}{AA}%
625 \TestVocal{7}{AE}%
626 \TestVocal{8}{AI}%
627 \TestVocal{9}{AO}%
628 \TestVocal{10}{AU}%
629 \TestVocal{11}{EA}%
630 \TestVocal{24}{OO}%
631 \TestVocal{25}{OU}%
632 \TestVocal{26}{UA}%
633 \TestVocal{29}{UO}%
634 \TestVocal{30}{UU}%
635 \TestVocal{31}{AAA}%
636 \TestVocal{155}{UUU}%
637 \TestVocal{156}{AAAA}%
638 \TestVocal{2147483647}{AIIIOEEIOIIOUE}%
639 \end{qstest}
640
641 \makeatletter
642 \newalphalph\AlphMult[mult]{\alphalph@Alph}{26}
643 \newalphalph\alphmult[mult]{\alphalph@alph}{26}
644 \newalphalph\LaTeXAlphMult[mult]{\@Alph}{26}
645 \newalphalph\LaTeXalphmult[mult]{\@alph}{26}
646 \makeatother
647 \newcommand*{\TestMult}[2]{%
648 \uppercase{\TestCallCmd\AlphMult{#2}{#1}%
649 \lowercase{\TestCallCmd\alphmult{#2}{#1}%
650 \uppercase{\TestCallCmd\LaTeXAlphMult{#2}{#1}%
651 \lowercase{\TestCallCmd\LaTeXalphmult{#2}{#1}%
652 }
653 \begin{qstest}{mult}{mult}
654 \TestMult{0}{}%
655 \TestMult{-1}{}%
656 \TestMult{-2147483647}{}%
657 \TestMult{1}{a}%
658 \TestMult{2}{b}%
659 \TestMult{26}{z}%
660 \TestMult{27}{aa}%
661 \TestMult{28}{bb}%
662 \TestMult{52}{zz}%
663 \TestMult{53}{aaa}%
664 \TestMult{54}{bbb}%
665 \TestMult{259}{yyyyyyyyyy}%
666 \TestMult{260}{zzzzzzzzzz}%

```

```

667 \TestMult{261}{aaaaaaaaaaa}%
668 \TestMult{262}{bbbbbbbbbbb}%
669 \end{qstest}
670
671 \def\myvocalB#1{%
672   \ifcase#1\or A\or E\or I\or O\or U\fi
673 }
674 \begin{qstest}{symbolnum}{symbolnum}
675   \makeatletter
676   \def\Test#1#2{%
677     \let\TestCmd\relax
678     \newalphalph\TestCmd{#1}{}%
679     \Expect*{\AlPh@Number}{#2}%
680   }%
681   \Test\@alph{26}%
682   \Test\@Alph{26}%
683   \Test\@fnsymbol{9}%
684   \Test\myvocalB{5}%
685   \Test\alphalph@alph{26}%
686   \Test\alphalph@Alph{26}%
687 \end{qstest}
688
689 \begin{qstest}{list}{list}
690   \makeatletter
691   \def\catch#1\relax{%
692     \def\FoundList{\catch#1}%
693   }%
694   \def\Test[#1]#2#3#4{%
695     \let\testcmd\relax
696     \newalphalph\testcmd[{#1}]{\catch}{#2}%
697     \testcmd{#3}|\relax
698     \expandafter\Expect\expandafter{\FoundList}{#4|}%
699     %
700     \let\SavedCatch\catch
701     \def\catch{\noexpand\catch\noexpand\foo}%
702     \edef\Result{#4|}%
703     \@onelevel@sanitize\Result
704     \let\catch\SavedCatch
705     \let\testcmd\relax
706     \newalphalph\testcmd[{#1}]{\catch\foo}{#2}%
707     \testcmd{#3}|\relax
708     \@onelevel@sanitize\FoundList
709     \Expect*{\FoundList}*{\Result}%
710   }%
711   \Test[alph]{26}{3}{\catch{3}}%
712   \Test[alph]{26}{12}{\catch{12}}%
713   \Test[alph]{26}{27}{\catch{1}\catch{1}}%
714   \Test[alph]{26}{78}{\catch{2}\catch{26}}%
715   \Test[wrap]{26}{7}{\catch{7}}%
716   \Test[wrap]{26}{14}{\catch{14}}%
717   \Test[wrap]{26}{80}{\catch{2}}%
718   \Test[wrap]{26}{700}{\catch{24}}%
719   \Test[mult]{26}{4}{\catch{4}}%
720   \Test[mult]{26}{17}{\catch{17}}%
721   \Test[mult]{26}{54}{\catch{2}\catch{2}\catch{2}}%
722 \end{qstest}
723
724 \begin{document}
725 \end{document}
726 </test2>

```

## 5 Installation

### 5.1 Download

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

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

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

**Bundle.** All the packages of the bundle ‘oberdiek’ 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/oberdiek.tds.zip](#)

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

### 5.2 Bundle installation

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

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

**Script installation.** Check the directory `TDS:scripts/oberdiek/` for scripts that need further installation steps. Package `attachfile2` comes with the Perl script `pdfatfi.pl` that should be installed in such a way that it can be called as `pdfatfi`. Example (linux):

```
chmod +x scripts/oberdiek/pdfatfi.pl
cp scripts/oberdiek/pdfatfi.pl /usr/local/bin/
```

### 5.3 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
```

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

<code>alphalph.sty</code>	→ <code>tex/generic/oberdiek/alphalph.sty</code>
<code>alphalph.pdf</code>	→ <code>doc/latex/oberdiek/alphalph.pdf</code>
<code>test/alphalph-test1.tex</code>	→ <code>doc/latex/oberdiek/test/alphalph-test1.tex</code>
<code>test/alphalph-test2.tex</code>	→ <code>doc/latex/oberdiek/test/alphalph-test2.tex</code>
<code>test/alphalph-test3.tex</code>	→ <code>doc/latex/oberdiek/test/alphalph-test3.tex</code>
<code>alphalph.dtx</code>	→ <code>source/latex/oberdiek/alphalph.dtx</code>

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`.

### 5.4 Refresh file name databases

If your T<sub>E</sub>X distribution (teT<sub>E</sub>X, miK<sub>T</sub>E<sub>X</sub>, ...) relies on file name databases, you must refresh these. For example, teT<sub>E</sub>X users run `texhash` or `mktextlsr`.

---

<sup>1</sup><http://ftp.ctan.org/tex-archive/>

## 5.5 Some details for the interested

**Attached source.** The PDF documentation on CTAN also includes the `.dtx` source file. It can be extracted by AcrobatReader 6 or higher. Another option is `pdftk`, e.g. unpack the file into the current directory:

```
pdftk alphalph.pdf unpack_files output .
```

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

**plain- $\text{\TeX}$ :** Run `docstrip` and extract the files.

**$\text{\LaTeX}$ :** Generate the documentation.

If you insist on using  $\text{\LaTeX}$  for `docstrip` (really, `docstrip` does not need  $\text{\LaTeX}$ ), 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 $\text{\LaTeX}$` :

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

## 6 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](#))

[1999/04/12 v1.0]

- Documentation added in `dtx` format.
- $\varepsilon$ - $\text{\TeX}$  support added.

[1999/04/13 v1.1]

- Minor documentation change.
- First CTAN release.

---

<sup>2</sup>Url: <http://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: <http://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: <http://groups.google.com/group/comp.text.tex/msg/cec563eef8bf65d0>

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

- Reload check (for plain- $\text{\TeX}$ )
- New DTX framework.
- LPPL 1.3

- \newalphalph added.

- Line ends sanitized.

- 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.

- Code is not changed.
- URLs updated from [www.dejanews.com](http://www.dejanews.com) to [groups.google.com](http://groups.google.com).

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

Symbols	
\# . . . . .	420, 482
% . . . . .	485
\@ . . . . .	421, 478
\@Alpha . . . . .	528, 532, 644, 682
\@LastSymbol . . . . .	537, 540, 544
\@PackageError . . . . .	121
\@PackageInfo . . . . .	384
\@ReturnAfterElseFi . . . . .	138, 286, 314, 337
\@ReturnAfterFi . . . . .	138, 296, 320, 346, 354
\@alph . . . . .	529, 533, 645, 681
\@ctrerrr . . . . .	209
\@ehc . . . . .	121
\@firstofone . . . . .	429, 432
\@fnsymbol . . . . .	683
\@gobble . . . . .	426, 434
\@gobblefour . . . . .	140, 273
\@ifdefinable . . . . .	136
\@ifnextchar . . . . .	161
\@nil . . . . .	537, 540, 544
\@onelevel@sanitize . . . . .	703, 708
\@undefined . . . . .	52, 127, 503
\[ . . . . .	483
\\ . . . . .	352, 378, 479, 535, 541
\{ . . . . .	418, 480
\} . . . . .	419, 481
\] . . . . .	484
\_ . . . . .	486
<b>A</b>	
\advance . . . . .	459, 467
\aftergroup . . . . .	26
\AlPh@AtEnd . . . . .	80, 81, 415
\AlPh@BinSearch . . . . .	223, 229, 244, 256, 260
\AlPh@BracketLeft . . . . .	152, 154

<code>\AlPh@CheckPositive</code>	271, 278, 306, 329		
<code>\AlPh@ctrerr</code>	172, 182, 186, 210		
<code>\AlPh@Error</code>	118, 130, 191, 232, 372, 406		
<code>\AlPh@ExpSearch</code>	194, 219		
<code>\AlPh@GetNumberOfSymbols</code>	187, 379		
<code>\AlPh@IfDefinable</code>	124, 136, 360, 402		
<code>\AlPh@IfOptArg</code>	143, 147, 161, 164, 362		
<code>\AlPh@IfOptArgNext</code>	150, 153		
<code>\AlPh@Method@alph</code>	277		
<code>\AlPh@Method@mult</code>	328		
<code>\AlPh@Method@wrap</code>	305		
<code>\AlPh@NewAlphaAlph</code>	388, 393, 401		
<code>\AlPh@newalphalph</code>	363, 365, 369		
<code>\AlPh@Next</code>	220, 237, 242, 250, 255, 259, 269		
<code>\AlPh@Number</code>	190, 193, 223, 225, 231, 264, 267, 380, 385, 390, 679		
<code>\AlPh@ProcessAlph</code>	281, 284, 302		
<code>\AlPh@ProcessBinSearch</code>	245, 249		
<code>\AlPh@ProcessMult</code>	332, 335		
<code>\AlPh@ProcessWrap</code>	309, 312		
<code>\AlPh@StepAlph</code>	287, 301		
<code>\AlPh@StepMult</code>	338, 351		
<code>\AlPh@StepWrap</code>	315, 325		
<code>\AlPh@TempA</code>	148, 155		
<code>\AlPh@TempB</code>	149, 157		
<code>\AlPh@TestNumber</code>	188, 204, 221, 227, 253		
<code>\AlPh@Token</code>	150, 154		
<code>\AlPh@Unavailablefalse</code>	201, 205		
<code>\AlPh@Unavailabletrue</code>	198, 209, 210, 215		
<code>\AlphaAlph</code>	4, 413, 549		
<code>\alphalph</code>	414, 550		
<code>\alphalph@Alph</code>	5, 166, 413, 530, 642, 686		
<code>\alphalph@alph</code>	166, 414, 531, 643, 685		
<code>\AlphMult</code>	642, 648		
<code>\alphmult</code>	643, 649		
<code>\AlphWrap</code>	530, 557		
<code>\alphwrap</code>	531, 558		
<b>B</b>			
<code>\begin</code>	565, 593, 617, 653, 674, 689, 724		
<code>\body</code>	438, 442		
<b>C</b>			
<code>\catch</code>	691, 692, 696, 700, 701, 704, 706, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721		
<code>\catcode</code>	3, 4, 5, 6, 7, 8, 9, 17, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 64, 65, 68, 69, 70, 71, 75, 76, 77, 78, 82, 84, 112, 113, 115, 116, 418, 419, 420, 421, 456, 465, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487		
<code>\count@</code>	423, 452, 456, 458, 459, 463, 465, 466, 467		
<code>\countdef</code>	423		
<code>\csname</code>	10, 18, 44, 60, 67, 103, 111, 125, 140, 144, 146, 371, 389, 394, 422, 425, 428, 431, 470, 492		
<b>D</b>			
<code>\detokenize</code>	375		
<code>\DisableNumexpr</code>	502, 508, 518		
<code>\documentclass</code>	498		
<b>E</b>			
<code>\empty</code>	13, 14		
<code>\end</code>	493, 592, 606, 639, 669, 687, 722, 725		
<code>\endcsname</code>	10, 18, 44, 60, 67, 103, 111, 125, 140, 144, 146, 371, 389, 394, 422, 425, 428, 431, 470, 492		
<code>\endinput</code>	26		
<code>\escapechar</code>	120, 383		
<code>\Expect</code>	520, 522, 524, 679, 698, 709		
<b>F</b>			
<code>\foo</code>	701, 706		
<code>\FoundList</code>	692, 698, 708, 709		
<code>\futurelet</code>	150		
<b>H</b>			
<code>\hbox</code>	207, 517		
<b>I</b>			
<code>\ifAlPh@Unavailable</code>	189, 197, 222, 228, 254		
<code>\ifcase</code>	127, 167, 177, 380, 610, 672		
<code>\ifdim</code>	214		
<code>\iffalse</code>	202		
<code>\ifnum</code>	226, 251, 252, 272, 285, 313, 336, 403, 458, 466		
<code>\iftrue</code>	199		
<code>\ifx</code>	11, 14, 18, 44, 52, 55, 103, 111, 125, 127, 140, 144, 146, 154, 352, 371, 378, 422, 425, 428, 431, 470, 535, 541		
<code>\immediate</code>	20, 46		
<code>\IncludeTests</code>	513		
<code>\input</code>	104, 105, 471		
<code>\intcalcAdd</code>	246		
<code>\intcalcDec</code>	289, 292, 316, 339, 342		
<code>\intcalcDiv</code>	292, 339		
<code>\intcalcInc</code>	288, 316, 339, 342		
<code>\intcalcMod</code>	289, 316, 342		
<code>\intcalcShl</code>	238		
<code>\intcalcShr</code>	246		
<code>\iterate</code>	439, 441, 443		
<b>K</b>			
<code>\kernel@ifnextchar</code>	164		
<b>L</b>			
<code>\LastSymbol</code>	534, 553		
<code>\LaTeXAlphAlph</code>	528, 551		
<code>\LaTeXalphalph</code>	529, 552		
<code>\LaTeXAlphMult</code>	644, 650		
<code>\LaTeXalphmult</code>	645, 651		
<code>\LaTeXAlphWrap</code>	532, 559		
<code>\LaTeXalphwrap</code>	533, 560		
<code>\LoadCommand</code>	471, 488		
<code>\LogTests</code>	514		
<code>\loop</code>	437, 453, 464		
<code>\lowercase</code>	550, 552, 558, 560, 649, 651		

<b>M</b>	
<code>\makeatletter</code> .....	686, 694, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721
.....	500, 527, 608, 641, 675, 690
<code>\makeatother</code> .....	547, 612, 646
<code>\MessageBreak</code> .....	407
<code>\myvocals</code> .....	609, 613
<code>\myvocalsB</code> .....	671, 684
<b>N</b>	
<code>\NeedsTeXFormat</code> .....	496
<code>\newalphalph</code> .	5, 360, 413, 414, 528, 529, 530, 531, 532, 533, 613, 642, 643, 644, 645, 678, 696, 706
<code>\newcommand</code> .....	502, 505, 516, 548, 556, 562, 609, 614, 647
<code>\newif</code> .....	197
<code>\next</code> .....	443, 445, 447
<code>\nofiles</code> .....	497
<code>\number</code> .....	238, 246, 279, 287, 291, 307, 315, 330, 341, 395
<code>\numexpr</code> .....	280, 308, 331, 396, 501, 503, 506
<b>P</b>	
<code>\PackageInfo</code> .....	23
<code>\ProvidesPackage</code> .....	15, 61
<b>R</b>	
<code>\RangeCatcodeInvalid</code> .....	462, 474, 475, 476, 477
<code>\repeat</code> .....	437, 449, 460, 468
<code>\RequirePackage</code> .....	107, 108
<code>\RestoreCatcodes</code> ..	451, 454, 455, 489
<code>\RestoreNumexpr</code> .....	505, 511
<code>\Result</code> .....	702, 703, 709
<code>\romannumeral</code> .....	338
<b>S</b>	
<code>\saved@numexpr</code> .....	501, 506
<code>\SavedCatch</code> .....	700, 704
<code>\setbox</code> .....	207, 517
<b>T</b>	
<code>\Test</code> .....	473, 491, 676, 681, 682, 683, 684, 685,
	686, 694, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721
<code>\TestAlph</code> .....	548, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605
<code>\TestAlphWrap</code> .....	554, 556
<code>\TestCallCmd</code> .....	549, 550, 551, 552, 557, 558, 559, 560, 562, 648, 649, 650, 651
<code>\TestCmd</code> .....	516, 563, 615, 677, 678
<code>\testcmd</code> ..	695, 696, 697, 705, 706, 707
<code>\TestMult</code> .....	647, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668
<code>\TestString</code> .....	519, 520, 521, 522
<code>\TestVocals</code> .....	614, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638
<code>\the</code> .....	68, 69, 70, 71, 82, 280, 308, 331, 396, 456, 524
<code>\TMP@EnsureCode</code> .....	79, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101
<b>U</b>	
<code>\uppercase</code> .....	549, 551, 557, 559, 648, 650
<code>\usepackage</code> .....	510, 512
<b>V</b>	
<code>\vocalsvocals</code> .....	613, 615
<b>W</b>	
<code>\wd</code> .....	214, 524
<code>\WrapString</code> .....	553, 554
<code>\write</code> .....	20, 46
<b>X</b>	
<code>\x</code> ..	10, 11, 14, 19, 23, 25, 45, 50, 60, 66, 74