

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^AT_EX and plain-T_EX are supported.

Contents

1	Documentation	2
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
2	Implementation	6
2.1	Begin of package	6
2.2	Catcodes	7
2.3	Package loading	8
2.4	ε -T _E 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
3	Test	15
3.1	Catcode checks for loading	15
4	Macro tests	17

5	Installation	21
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
6	History	22
[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
7	Index	23

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^AT_EX 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^AT_EX’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^AT_EX 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_EX 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^AT_EX 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_EX and L^AT_EX:

plain-T_EX: `\input alphalph.sty`

L^AT_EX 2_ε: `\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^AT_EX 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{$\langle number \rangle$}</code>	<code>\alphalph{$\langle number \rangle$}</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 {$\langle cmd \rangle$} [$\langle method \rangle$] {$\langle symbols provider \rangle$} {$\langle number of symbols \rangle$}</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^AT_EX'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 {$\langle number \rangle$}</code> <code>\alphalph@alph {$\langle number \rangle$}</code>
--

They are basically the same as `\@Alph` and `\@alph`. Some languages of package `babel` redefine L^AT_EX'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^AT_EX: 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 \TeX number. Anything that would be accepted by `\number` is a valid argument:
 - explicite constants,
 - macros that expand to a number,
 - count registers, \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

\TeX format: The package does not depend on \LaTeX , it can also be used by plain- \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 \LaTeX .

```
2 \begingroup
3   \catcode44 12 % ,
4   \catcode45 12 % -
5   \catcode46 12 % .
6   \catcode58 12 % :
7   \catcode64 11 % @
8   \expandafter\let\expandafter\x\csname ver@alphalph.sty\endcsname
9   \ifcase 0%
10    \ifx\x\relax % plain
11    \else
12      \ifx\x\empty % LaTeX
13      \else
14        1%
15      \fi
16    \fi
17  \else
18    \catcode35 6 % #
19    \catcode123 1 % {
20    \catcode125 2 % }
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  \endgroup
30  \expandafter\endinput
```

```

31 \fi
32 \endgroup
Package identification:
33 \begingroup
34 \catcode35 6 % #
35 \catcode40 12 % (
36 \catcode41 12 % )
37 \catcode44 12 % ,
38 \catcode45 12 % -
39 \catcode46 12 % .
40 \catcode47 12 % /
41 \catcode58 12 % :
42 \catcode64 11 % @
43 \catcode123 1 % {
44 \catcode125 2 % }
45 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
46 \def\x#1#2#3[#4]{\endgroup
47 \immediate\write-1{Package: #3 #4}%
48 \xdef#1{#4}%
49 }%
50 \else
51 \def\x#1#2[#3]{\endgroup
52 #2[#{#3}]%
53 \ifx#1\@undefined
54 \xdef#1{#3}%
55 \fi
56 \ifx#1\relax
57 \xdef#1{#3}%
58 \fi
59 }%
60 \fi
61 \expandafter\x\csname ver@alphalph.sty\endcsname
62 \ProvidesPackage{alphalph}%
63 [2008/08/11 v2.1 Converting numbers to letters (HO)]

```

2.2 Catcodes

```

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

```

```

90 \TMP@EnsureCode{41}{12}% )
91 \TMP@EnsureCode{43}{12}% +
92 \TMP@EnsureCode{44}{12}% ,
93 \TMP@EnsureCode{46}{12}% .
94 \TMP@EnsureCode{47}{12}% /
95 \TMP@EnsureCode{59}{12}% ;
96 \TMP@EnsureCode{60}{12}% <
97 \TMP@EnsureCode{61}{12}% =
98 \TMP@EnsureCode{62}{12}% >
99 \TMP@EnsureCode{91}{12}% [
100 \TMP@EnsureCode{93}{12}% ]
101 \TMP@EnsureCode{96}{12}% '
102 \TMP@EnsureCode{124}{12}% |

```

2.3 Package loading

```

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

```

2.4 ε -TeX detection

```

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

```

2.5 Help macros

\AlPh@Error

```

119 \def\AlPh@Error#1{%
120   \begingroup
121     \escapechar=92 % backslash
122     \@PackageError{alphalph}{#1}\@ehc
123   \endgroup
124 }

```

\AlPh@ifDefinable

```

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

```

\AlPh@ifDefinable

```

137   \let\AlPh@ifDefinable\@ifdefinable
138 \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.)

```

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

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

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

```

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

```

```

174   @%
175   \fi
176 }
177 \def\alphalph@alph#1{%
178   \ifcase#1%
179     @%
180     \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%
181     \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%
182   \else
183     \AlPh@ctrerr
184     @%
185   \fi
186 }

```

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

```

187 \def\AlPh@ctrerr{}

```

2.7 Finding number of symbols

`\AlPh@GetNumberOfSymbols` #1: symbols provider

```

188 \def\AlPh@GetNumberOfSymbols#1{%
189   \AlPh@TestNumber1!{#1}%
190   \ifAlPh@Unavailable
191     \def\AlPh@Number{0}%
192     \AlPh@Error{No symbols found}%
193   \else
194     \def\AlPh@Number{1}%
195     \AlPh@ExpSearch2!{#1}%
196   \fi
197 }

```

`\ifAlPh@Unavailable`

```

198 \newif\ifAlPh@Unavailable
199 \def\AlPh@Unavailabletrue{%
200   \global\let\ifAlPh@Unavailable\iftrue
201 }
202 \def\AlPh@Unavailablefalse{%
203   \global\let\ifAlPh@Unavailable\iffalse
204 }

```

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

#2: symbols provider

```

205 \def\AlPh@TestNumber#1!#2{%
206   \AlPh@Unavailablefalse
207   \begingroup
208     \setbox0=\hbox{%
209       \begingroup % color
210         \let\@ctrerr\AlPh@Unavailabletrue
211         \let\AlPh@ctrerr\AlPh@Unavailabletrue
212         #2{#1}%
213       \endgroup
214     }%
215     \ifdim\wd0=0pt %
216       \AlPh@Unavailabletrue
217     \fi
218   \endgroup
219 }

```

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

#2: symbols provider

```

220 \def\AlPh@ExpSearch#1!#2{%
221   \let\AlPh@Next\relax
222   \AlPh@TestNumber#1!{#2}%
223   \ifAlPh@Unavailable
224     \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
225   \else
226     \def\AlPh@Number{#1}%
227     \ifnum#1>1073741823 %
228       \AlPh@TestNumber2147483647!{#2}%
229       \ifAlPh@Unavailable
230         \AlPh@BinSearch#1!2147483647!{#2}%
231       \else
232         \def\AlPh@Number{0}%
233         \AlPh@Error{%
234           Maximal symbol number not found%
235         }%
236       \fi
237     \else
238       \def\AlPh@Next{%
239         \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}!{#2}%
240       }%
241     \fi
242   \fi
243   \AlPh@Next
244 }

```

\AlPh@BinSearch #1: available number
#2: unavailable number, #2 > #1
#3: symbols provider

```

245 \def\AlPh@BinSearch#1!#2!#3{%
246   \expandafter\AlPh@ProcessBinSearch
247   \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
248   #1!#2!{#3}%
249 }

```

\AlPh@ProcessBinSearch #1: number to be tested, #2 ≤ #1 ≤ #3
#2: available number
#3: unavailable number
#4: symbols provider

```

250 \def\AlPh@ProcessBinSearch#1!#2!#3!#4{%
251   \let\AlPh@Next\relax
252   \ifnum#1>#2 %
253     \ifnum#1<#3 %
254       \AlPh@TestNumber#1!{#4}%
255       \ifAlPh@Unavailable
256         \def\AlPh@Next{%
257           \AlPh@BinSearch#2!#1!{#4}%
258         }%
259       \else
260         \def\AlPh@Next{%
261           \AlPh@BinSearch#1!#3!{#4}%
262         }%
263       \fi
264     \else
265       \def\AlPh@Number{#2}%
266     \fi
267   \else
268     \def\AlPh@Number{#2}%
269   \fi
270   \AlPh@Next
271 }

```

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)
272 \def\AlPh@CheckPositive#1!#2{%
273   \ifnum#1<1 %
274     \expandafter\@gobblefour
275   \fi
276   #2{#1}%
277 }
```

2.8.2 Method ‘alph’

```
\AlPh@Method@alph #1: number of symbols
#2: symbols provider
#3: number to be converted
278 \def\AlPh@Method@alph#1#2#3{%
279   \expandafter\AlPh@CheckPositive
280 |   \number#3!%
281 +   \the\numexpr#3!%
282   \AlPh@ProcessAlph
283   {#1}{#2}%
284 }

\AlPh@ProcessAlph #1: current number
#2: number of symbols
#3: symbols provider
285 \def\AlPh@ProcessAlph#1#2#3{%
286   \ifnum#1>#2 %
287     \@ReturnAfterElseFi{%
288       \expandafter\AlPh@StepAlph\number
289       \intcalcInc{%
290         \intcalcMod{\intcalcDec{#1}}{#2}%
291       }%
292       \expandafter!\number
293       \intcalcDiv{\intcalcDec{#1}}{#2}%
294       !{#2}{#3}%
295     }%
296   \else
297     \@ReturnAfterFi{%
298       #3{#1}%
299     }%
300   \fi
301 }

\AlPh@StepAlph #1: current last digit
#2: new current number
#3: number of symbols
#4: symbols provider
302 \def\AlPh@StepAlph#1!#2!#3#4{%
303   \AlPh@ProcessAlph{#2}{#3}{#4}%
304   #4{#1}%
305 }
```

2.8.3 Method ‘wrap’

```

\AlPh@Method@wrap #1: number of symbols
                  #2: symbols provider
                  #3: number to be converted
306 \def\AlPh@Method@wrap#1#2#3{%
307   \expandafter\AlPh@CheckPositive
308 |   \number#3!%
309 +   \the\numexpr#3!%
310   \AlPh@ProcessWrap
311   {#1}{#2}%
312 }

\AlPh@ProcessWrap #1: number to be converted
                  #2: number of symbols
                  #3: symbols provider
313 \def\AlPh@ProcessWrap#1#2#3{%
314   \ifnum#1>#2 %
315     \@ReturnAfterElseFi{%
316       \expandafter\AlPh@StepWrap\number
317       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
318       !{#3}%
319     }%
320   \else
321     \@ReturnAfterFi{%
322       #3{#1}%
323     }%
324   \fi
325 }

\AlPh@StepWrap #1: final number
               #2: symbols provider
326 \def\AlPh@StepWrap#1!#2{%
327   #2{#1}%
328 }

```

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
329 \def\AlPh@Method@mult#1#2#3{%
330   \expandafter\AlPh@CheckPositive
331 |   \number#3!%
332 +   \the\numexpr#3!%
333   \AlPh@ProcessMult
334   {#1}{#2}%
335 }

```

```

\AlPh@ProcessMult #1: number to be converted
#2: number of symbols
#3: symbols provider
336 \def\AlPh@ProcessMult#1#2#3{%
337   \ifnum#1>#2 %
338     \@ReturnAfterElseFi{%
339       \expandafter\AlPh@StepMult\romannumeral
340       \intcalcInc{\intcalcDiv{\intcalcDec{#1}}{#2}}%
341       000%
342       \expandafter!\number
343       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
344       !{#3}%
345     }%
346   \else
347     \@ReturnAfterFi{%
348       #3{#1}%
349     }%
350   \fi
351 }

```

```

\AlPh@StepMult #1#2: repetitions coded as list of character ‘m’
#3: symbol slot
#4: symbols provider
352 \def\AlPh@StepMult#1#2!#3!#4{%
353   \ifx\\#2\\%
354   \else
355     \@ReturnAfterFi{%
356       \AlPh@StepMult#2!#3!{#4}%
357     }%
358   \fi
359   #4{#3}%
360 }

```

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

```

361 \AlPh@IfDefinable\newalphalph{%
362   \def\newalphalph#1{%
363     \AlPh@IfOptArg{%
364       \AlPh@newalphalph{#1}%
365     }{%
366       \AlPh@newalphalph{#1}[alph]%
367     }%
368   }%
369 }

```

```

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

```

```

377 }%
378 \else
379 \ifx\\#4\\%
380 \AlPh@GetNumberOfSymbols{#3}%
381 \ifcase\AlPh@Number
382 \else
383 \begingroup
384 \escapechar=92 % backslash
385 \@PackageInfo{alphalph}{%
386   Number of symbols for \string#1 is \AlPh@Number
387 }%
388 \endgroup
389 \expandafter\AlPh@NewAlphAlph
390 \csname AlPh@Method@#2\expandafter\endcsname
391 \AlPh@Number!{#1}{#3}%
392 \fi
393 \else
394 \expandafter\AlPh@NewAlphAlph
395 \csname AlPh@Method@#2\expandafter\endcsname
396 | \number#4!%
397 + \the\numexpr#4!%
398 {#1}{#3}%
399 \fi
400 \fi
401 }%

```

```

\AlPh@NewAlphAlph #1: method macro
                  #2: number of symbols
                  #3: cmd
                  #4: symbols provider
402 \def\AlPh@NewAlphAlph#1#2!#3#4{%
403   \AlPh@IfDefinable#3{%
404     \ifnum#2>0 %
405       \def#3{#1{#2}{#4}}%
406     \else
407       \AlPh@Error{%
408         Definition of \string#3 failed,\MessageBreak
409         because number of symbols (#2) is not positive%
410       }%
411     \fi
412   }%
413 }

```

\AlphAlph

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

\alphalph

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

```
416 \AlPh@AtEnd
```

```
417 \</package>
```

3 Test

3.1 Catcode checks for loading

```

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

```

```

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

```



```

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

```

4 Macro tests

```

496 <*test2>
497 \NeedsTeXFormat{LaTeX2e}
498 \nofiles
499 \documentclass{article}
500 <*noetex>
501 \makeatletter
502 \let\saved@numexpr\numexpr
503 \newcommand*\DisableNumexpr{%
504   \let\numexpr\undefined
505 }
506 \newcommand*\RestoreNumexpr{%
507   \let\numexpr\saved@numexpr
508 }
509 \DisableNumexpr
510 </noetex>
511 \usepackage{alphalph}[2008/08/11]
512 <noetex> \RestoreNumexpr
513 \usepackage{qstest}
514 \IncludeTests{*}
515 \LogTests{log}{*}{*}
516
517 \newcommand*\TestCmd[3]{%
518   \setbox0=\hbox{%
519 <noetex>   \DisableNumexpr
520     \edef\TestString{#1{#2}}%
521     \expandafter\Expect\expandafter{\TestString}{#3}%
522     \edef\TestString{#1{#2} }%
523     \expandafter\Expect\expandafter{\TestString}{#3 }%
524   }%
525   \Expect*\{the\wd0}{0.0pt}%
526 }
527
528 \makeatletter
529 \newalphalph\LaTeXAlphAlph\@Alph{26}
530 \newalphalph\LaTeXalphalph\@alph{26}
531 \newalphalph\AlphWrap[wrap]\alphalph@Alph{26}
532 \newalphalph\alphwrap[wrap]\alphalph@alph{26}
533 \newalphalph\LaTeXAlphWrap[wrap]\@Alph{26}
534 \newalphalph\LaTeXalphwrap[wrap]\@alph{26}
535 \def\LastSymbol#1{%
536   \ifx\#1\%
537   \else
538     \@LastSymbol#1\@nil
539   \fi
540 }
541 \def\@LastSymbol#1#2\@nil{%
542   \ifx\#2\%
543     #1%

```

```

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

```

```

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

```

```

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

```

5 Installation

5.1 Download

Package. This package is available on CTAN¹:

[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 \TeX 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- \TeX :

```
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>	\rightarrow <code>tex/generic/oberdiek/alphalph.sty</code>
<code>alphalph.pdf</code>	\rightarrow <code>doc/latex/oberdiek/alphalph.pdf</code>
<code>test/alphalph-test1.tex</code>	\rightarrow <code>doc/latex/oberdiek/test/alphalph-test1.tex</code>
<code>test/alphalph-test2.tex</code>	\rightarrow <code>doc/latex/oberdiek/test/alphalph-test2.tex</code>
<code>test/alphalph-test3.tex</code>	\rightarrow <code>doc/latex/oberdiek/test/alphalph-test3.tex</code>
<code>alphalph.dtx</code>	\rightarrow <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 \TeX distribution (te \TeX , mi \TeX , ...) relies on file name databases, you must refresh these. For example, te \TeX users run `texhash` or `mktextlsr`.

¹<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 \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
```

6 History

[1999/03/19 v0.1]

- The first version was built as a response to a [question](#)² of Will Douglas³ and the [request](#)⁴ of Donald Arsenau⁵, published in the newsgroup `comp.text.tex`: “[Re: alph counters > 26](#)”⁶
- Copyright: LPPL ([CTAN:macros/latex/base/lppl.txt](#))

[1999/04/12 v1.0]

- Documentation added in `dtx` format.
- ε - \TeX support added.

[1999/04/13 v1.1]

- Minor documentation change.
- First CTAN release.

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

³Will Douglas’s email address: william.douglas@wolfson.ox.ac.uk

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

⁵Donald Arsenau’s email address: asnd@reg.triumf.ca

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

[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 ε -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.

7 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; numbers in **roman** refer to the code lines where the entry is used.

[illegible]

<code>\AlPh@ctrerr</code>	173, 183, 187, 211	<code>\DisableNumexpr</code>	503, 509, 519
<code>\AlPh@Error</code> 119 , 131 , 192 , 233 , 373 , 407		<code>\documentclass</code>	499
<code>\AlPh@ExpSearch</code>	195, 220	E	
<code>\AlPh@GetNumberOfSymbols</code> . . .	188 , 380	<code>\empty</code>	12
<code>\AlPh@IfDefinable</code>	125 , 137 , 361 , 403	<code>\end</code> 494 , 593 , 607 , 640 , 670 , 688 , 723 , 726	
<code>\AlPh@IfOptArg</code> 144 , 148 , 162 , 165 , 363		<code>\endcsname</code> 8 , 21 , 45 , 61 , 68 , 104 , 112 ,	
<code>\AlPh@IfOptArgNext</code>	151 , 154	126 , 141 , 145 , 147 , 372 , 390 ,	
<code>\AlPh@Method@alph</code>	278	395 , 423 , 426 , 429 , 432 , 471 , 493	
<code>\AlPh@Method@mult</code>	329	<code>\endinput</code>	30
<code>\AlPh@Method@wrap</code>	306	<code>\escapechar</code>	121, 384
<code>\AlPh@NewAlphaAlph</code>	389 , 394 , 402	<code>\Expect</code>	521 , 523 , 525 , 680 , 699 , 710
<code>\AlPh@newalphalph</code>	364 , 366 , 370	F	
<code>\AlPh@Next</code>		<code>\foo</code>	702, 707
. 221 , 238 , 243 , 251 , 256 , 260 , 270		<code>\FoundList</code>	693 , 699 , 709 , 710
<code>\AlPh@Number</code>	191 , 194 , 224 , 226 ,	<code>\futurelet</code>	151
232 , 265 , 268 , 381 , 386 , 391 , 680		H	
<code>\AlPh@ProcessAlph</code>	282 , 285 , 303	<code>\hbox</code>	208 , 518
<code>\AlPh@ProcessBinSearch</code>	246 , 250	I	
<code>\AlPh@ProcessMult</code>	333 , 336	<code>\ifAlPh@Unavailable</code>	
<code>\AlPh@ProcessWrap</code>	310 , 313	190 , 198 , 223 , 229 , 255
<code>\AlPh@StepAlph</code>	288 , 302	<code>\ifcase</code>	9 , 128 , 168 , 178 , 381 , 611 , 673
<code>\AlPh@StepMult</code>	339 , 352	<code>\ifdim</code>	215
<code>\AlPh@StepWrap</code>	316 , 326	<code>\iffalse</code>	203
<code>\AlPh@TempA</code>	149 , 156	<code>\ifnum</code>	227 , 252 , 253 ,
<code>\AlPh@TempB</code>	150 , 158	273 , 286 , 314 , 337 , 404 , 459 , 467	
<code>\AlPh@TestNumber</code> 189 , 205 , 222 , 228 , 254		<code>\iftrue</code>	200
<code>\AlPh@Token</code>	151 , 155	<code>\ifx</code>	10 , 12 , 21 , 45 ,
<code>\AlPh@Unavailablefalse</code>	202 , 206	53 , 56 , 104 , 112 , 126 , 128 , 141 ,	
<code>\AlPh@Unavailabletrue</code>		145 , 147 , 155 , 353 , 372 , 379 ,	
.	199 , 210 , 211 , 216	423 , 426 , 429 , 432 , 471 , 536 , 542	
<code>\AlphAlph</code>	4 , 414 , 550	<code>\immediate</code>	23 , 47
<code>\alphalph</code>	415 , 551	<code>\IncludeTests</code>	514
<code>\alphalph@Alph</code> 5 , 167 , 414 , 531 , 643 , 687		<code>\input</code>	105 , 106 , 472
<code>\alphalph@alph</code> 167 , 415 , 532 , 644 , 686		<code>\intcalcAdd</code>	247
<code>\AlphMult</code>	643 , 649	<code>\intcalcDec</code>	290 , 293 , 317 , 340 , 343
<code>\alphmult</code>	644 , 650	<code>\intcalcDiv</code>	293 , 340
<code>\AlphWrap</code>	531 , 558	<code>\intcalcInc</code>	289 , 317 , 340 , 343
<code>\alphwrap</code>	532 , 559	<code>\intcalcMod</code>	290 , 317 , 343
B		<code>\intcalcShl</code>	239
<code>\begin</code> 566 , 594 , 618 , 654 , 675 , 690 , 725		<code>\intcalcShr</code>	247
<code>\body</code>	439 , 443	<code>\iterate</code>	440 , 442 , 444
C		K	
<code>\catch</code>	692 , 693 , 697 , 701 , 702 ,	<code>\kernel@ifnextchar</code>	165
705 , 707 , 712 , 713 , 714 , 715 ,		L	
716 , 717 , 718 , 719 , 720 , 721 , 722		<code>\LastSymbol</code>	535 , 554
<code>\catcode</code>	3 , 4 , 5 , 6 , 7 , 18 ,	<code>\LaTeXAlphAlph</code>	529 , 552
19 , 20 , 34 , 35 , 36 , 37 , 38 , 39 , 40 ,		<code>\LaTeXalphalph</code>	530 , 553
41 , 42 , 43 , 44 , 65 , 66 , 69 , 70 , 71 ,		<code>\LaTeXAlphMult</code>	645 , 651
72 , 76 , 77 , 78 , 79 , 83 , 85 , 113 ,		<code>\LaTeXalphmult</code>	646 , 652
114 , 116 , 117 , 419 , 420 , 421 ,		<code>\LaTeXAlphWrap</code>	533 , 560
422 , 457 , 466 , 479 , 480 , 481 ,		<code>\LaTeXalphwrap</code>	534 , 561
482 , 483 , 484 , 485 , 486 , 487 , 488		<code>\LoadCommand</code>	472 , 489
<code>\count@</code>	424 , 453 ,	<code>\LogTests</code>	515
457 , 459 , 460 , 464 , 466 , 467 , 468		<code>\loop</code>	438 , 454 , 465
<code>\countdef</code>	424	<code>\lowercase</code> 551 , 553 , 559 , 561 , 650 , 652	
<code>\csname</code>	8 , 21 , 45 , 61 , 68 , 104 , 112 ,		
126 , 141 , 145 , 147 , 372 , 390 ,			
395 , 423 , 426 , 429 , 432 , 471 , 493			
D			
<code>\detokenize</code>	376		

M	
<code>\makeatletter</code>	687, 695, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722
.....	501, 528, 609, 642, 676, 691
<code>\makeatother</code>	548, 613, 647
<code>\MessageBreak</code>	408
<code>\myvocal</code> s	610, 614
<code>\myvocal</code> sB	672, 685
N	
<code>\NeedsTeXFormat</code>	497
<code>\newalphalph</code> ..	5, 361, 414, 415, 529, 530, 531, 532, 533, 534, 614, 643, 644, 645, 646, 679, 697, 707
<code>\newcommand</code>	503, 506, 517, 549, 557, 563, 610, 615, 648
<code>\newif</code>	198
<code>\next</code>	444, 446, 448
<code>\nofiles</code>	498
<code>\number</code>	239, 247, 280, 288, 292, 308, 316, 331, 342, 396
<code>\numexpr</code> ..	281, 309, 332, 397, 502, 504, 507
P	
<code>\PackageInfo</code>	26
<code>\ProvidesPackage</code>	62
R	
<code>\RangeCatcodeInvalid</code>	463, 475, 476, 477, 478
<code>\repeat</code>	438, 450, 461, 469
<code>\RequirePackage</code>	108, 109
<code>\RestoreCatcodes</code> ..	452, 455, 456, 490
<code>\RestoreNumexpr</code>	506, 512
<code>\Result</code>	703, 704, 710
<code>\romannumeral</code>	339
S	
<code>\saved@numexpr</code>	502, 507
<code>\SavedCatch</code>	701, 705
<code>\setbox</code>	208, 518
T	
<code>\Test</code>	474, 492, 677, 682, 683, 684, 685, 686,
.....	687, 695, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722
<code>\TestAlph</code>	549, 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, 592, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606
<code>\TestAlphWrap</code>	555, 557
<code>\TestCallCmd</code>	550, 551, 552, 553, 558, 559, 560, 561, 563, 649, 650, 651, 652
<code>\TestCmd</code>	517, 564, 616, 678, 679
<code>\testcmd</code> ..	696, 697, 698, 706, 707, 708
<code>\TestMult</code>	648, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669
<code>\TestString</code>	520, 521, 522, 523
<code>\TestVocal</code> s	615, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639
<code>\the</code>	69, 70, 71, 72, 83, 281, 309, 332, 397, 457, 525
<code>\TMP@EnsureCode</code>	80, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102
U	
<code>\uppercase</code> ..	550, 552, 558, 560, 649, 651
<code>\usepackage</code>	511, 513
V	
<code>\vocalsvocal</code> s	614, 616
W	
<code>\wd</code>	215, 525
<code>\WrapString</code>	554, 555
<code>\write</code>	23, 47
X	
<code>\x</code> ..	8, 10, 12, 22, 26, 28, 46, 51, 61, 67, 75