

The alphalph package

Heiko Oberdiek
<oberdiek@uni-freiburg.de>

2007/09/09 v2.0

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
7	Index	23

1 Documentation

1.1 Introduction

L^AT_EX 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 T_EX 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.

$\langle number \rangle$	$\backslash\text{AlphAlph}\{\langle number \rangle\}$	$\backslash\text{alphalph}\{\langle number \rangle\}$
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  $\text{\LaTeX}$ .
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\relax
54 \xdef#1{#3}%
55 \fi
56 }%
57 \fi
58 \expandafter\x\csname ver@alphalph.sty\endcsname
59 \ProvidesPackage{alphalph}%
60 [2007/09/09 v2.0 Converting numbers to letters (HO)]

```

2.2 Catcodes

```

61 \begingroup
62 \catcode123 1 % {
63 \catcode125 2 % }
64 \def\x{\endgroup
65 \expandafter\edef\csname AlPh@AtEnd\endcsname{%
66 \catcode35 \the\catcode35\relax
67 \catcode64 \the\catcode64\relax
68 \catcode123 \the\catcode123\relax
69 \catcode125 \the\catcode125\relax
70 }%
71 }%
72 \x
73 \catcode35 6 % #
74 \catcode64 11 % @
75 \catcode123 1 % {
76 \catcode125 2 % }
77 \def\TMP@EnsureCode#1#2{%
78 \edef\AlPh@AtEnd{%
79 \AlPh@AtEnd
80 \catcode#1 \the\catcode#1\relax
81 }%
82 \catcode#1 #2\relax
83 }
84 \TMP@EnsureCode{33}{12}% !
85 \TMP@EnsureCode{39}{12}% '
86 \TMP@EnsureCode{40}{12}% (
87 \TMP@EnsureCode{41}{12}% )
88 \TMP@EnsureCode{43}{12}% +
89 \TMP@EnsureCode{44}{12}% ,

```

```

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

```

2.3 Package loading

```

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

```

2.4 ε -TeX detection

```

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

```

2.5 Help macros

`\AlPh@Error`

```

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

```

`\AlPh@ifDefinable`

```

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

```

`\AlPh@ifDefinable`

```

134   \let\AlPh@ifDefinable\@ifdefinable
135 \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.)

```

136 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
137 \long\def\@ReturnAfterFi#1\fi{\fi#1}

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

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

```

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

```

```

171    @%
172    \fi
173 }
174 \def\alphalph@alph#1{%
175   \ifcase#1%
176     @%
177     \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%
178     \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%
179   \else
180     \AlPh@ctrerr
181     @%
182   \fi
183 }

```

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

```

184 \def\AlPh@ctrerr{}

```

2.7 Finding number of symbols

`\AlPh@GetNumberOfSymbols` #1: symbols provider

```

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

```

`\ifAlPh@Unavailable`

```

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

```

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

#2: symbols provider

```

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

```

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

#2: symbols provider

```

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

```

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

```

242 \def\AlPh@BinSearch#1!#2!#3{%
243   \expandafter\AlPh@ProcessBinSearch
244   \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
245   #1!#2!{#3}%
246 }

```

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

```

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

```

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

2.8.2 Method ‘alph’

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

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

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

2.8.3 Method ‘wrap’

```

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

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

\AlPh@StepWrap #1: final number
               #2: symbols provider
323 \def\AlPh@StepWrap#1!#2{%
324   #2{#1}%
325 }

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

374 }%
375 \else
376 \ifx\#4\%
377 \AlPh@GetNumberOfSymbols{#3}%
378 \ifcase\AlPh@Number
379 \else
380 \begingroup
381 \escapechar=92 % backslash
382 \@PackageInfo{alphalph}{%
383   Number of symbols for \string#1 is \AlPh@Number
384 }%
385 \endgroup
386 \expandafter\AlPh@NewAlphAlph
387 \csname AlPh@Method@#2\expandafter\endcsname
388 \AlPh@Number!{#1}{#3}%
389 \fi
390 \else
391 \expandafter\AlPh@NewAlphAlph
392 \csname AlPh@Method@#2\expandafter\endcsname
393 | \number#4!%
394 + \the\numexpr#4!%
395 {#1}{#3}%
396 \fi
397 \fi
398 }%

```

```

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

```

\AlphAlph

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

\alphalph

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

```
413 \AlPh@AtEnd
```

```
414 \endpackage
```

3 Test

3.1 Catcode checks for loading

```

415 \test1
416 \catcode'\{=1 %
417 \catcode'\}=2 %
418 \catcode'\#=6 %
419 \catcode'\@=11 %

```

```

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

```



```

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

```

4 Macro tests

```

493 <*test2>
494 \NeedsTeXFormat{LaTeX2e}
495 \nofiles
496 \documentclass{article}
497 <*noetex>
498 \makeatletter
499 \let\saved@numexpr\numexpr
500 \newcommand*\DisableNumexpr{%
501   \let\numexpr\undefined
502 }
503 \newcommand*\RestoreNumexpr{%
504   \let\numexpr\saved@numexpr
505 }
506 \DisableNumexpr
507 </noetex>
508 \usepackage{alphalph}[2007/09/09]
509 <noetex> \RestoreNumexpr
510 \usepackage{qstest}
511 \IncludeTests{*}
512 \LogTests{log}{*}{*}
513
514 \newcommand*\TestCmd}[3]{%
515   \setbox0=\hbox{%
516     <noetex> \DisableNumexpr
517     \edef\TestString{#1{#2}}%
518     \expandafter\Expect\expandafter{\TestString}{#3}%
519     \edef\TestString{#1{#2} }%
520     \expandafter\Expect\expandafter{\TestString}{#3 }%
521   }%
522   \Expect*{\the\wd0}{0.0pt}%
523 }
524
525 \makeatletter
526 \newalphalph\LaTeXAlphAlph\@Alph{26}
527 \newalphalph\LaTeXalphalph\@alph{26}
528 \newalphalph\AlphWrap[wrap]\alphalph@Alph{26}
529 \newalphalph\alphwrap[wrap]\alphalph@alph{26}
530 \newalphalph\LaTeXAlphWrap[wrap]\@Alph{26}
531 \newalphalph\LaTeXalphwrap[wrap]\@alph{26}
532 \def\LastSymbol#1{%
533   \ifx\\#1\\%
534   \else
535     \@LastSymbol#1\@nil
536   \fi
537 }
538 \def\@LastSymbol#1#2\@nil{%
539   \ifx\\#2\\%
540     #1%

```

```

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

```

```

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

```

```

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

```
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_EX distribution (teT_EX, miK_TE_X, ...) relies on file name databases, you must refresh these. For example, teT_EX 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 question² of Will Douglas³ and the request⁴ 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://www.dejanews.com/\[ST_rn=ps\]/getdoc.xp?AN=455791936](http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=455791936)

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

⁴Url: [http://www.dejanews.com/\[ST_rn=ps\]/getdoc.xp?AN=456358639](http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=456358639)

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

⁶Url: [http://www.dejanews.com/\[ST_rn=ps\]/getdoc.xp?AN=456485421](http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=456485421)

[1999/06/26 v1.2]

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

[2006/02/20 v1.3]

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

[2006/05/30 v1.4]

- `\newalphalph` added.

[2007/04/11 v1.5]

- Line ends sanitized.

[2007/09/09 v2.0]

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

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.

Symbols	
<code>\#</code>	418, 480
<code>\%</code>	483
<code>\@</code>	419, 476
<code>\@Alph</code>	526, 530, 642, 680
<code>\@LastSymbol</code>	535, 538, 542
<code>\@PackageError</code>	119
<code>\@PackageInfo</code>	382
<code>\@ReturnAfterElseFi</code>	<u>136</u> , 284, 312, 335
<code>\@ReturnAfterFi</code>	<u>136</u> , 294, 318, 344, 352
<code>\@alph</code>	527, 531, 643, 679
<code>\@ctrerr</code>	207
<code>\@ehc</code>	119
<code>\@firstofone</code>	427, 430
<code>\@fnsymbol</code>	681
<code>\@gobble</code>	424, 432
<code>\@gobblefour</code>	<u>138</u> , 271
<code>\@ifdefinable</code>	134
<code>\@ifnextchar</code>	159
<code>\@nil</code>	535, 538, 542
<code>\@onelevel@sanitize</code>	701, 706
<code>\@undefined</code>	125, 501
<code>\[</code>	481
<code>\</code>	350, 376, 477, 533, 539
<code>\{</code>	416, 478
<code>\}</code>	417, 479
<code>\]</code>	482
<code>_</code>	484
A	
<code>\advance</code>	457, 465
<code>\AlPh@AtEnd</code>	78, 79, 413
<code>\AlPh@BinSearch</code>	221, 227, <u>242</u> , 254, 258
<code>\AlPh@BracketLeft</code>	150, 152
<code>\AlPh@CheckPositive</code>	<u>269</u> , 276, 304, 327
<code>\AlPh@ctrerr</code>	170, 180, <u>184</u> , 208
<code>\AlPh@Error</code>	<u>116</u> , 128, 189, 230, 370, 404
<code>\AlPh@ExpSearch</code>	192, <u>217</u>
<code>\AlPh@GetNumberOfSymbols</code>	<u>185</u> , 377
<code>\AlPh@IfDefinable</code>	<u>122</u> , <u>134</u> , 358, 400
<code>\AlPh@IfOptArg</code>	<u>141</u> , 145, 159, 162, 360
<code>\AlPh@IfOptArgNext</code>	148, 151
<code>\AlPh@Method@alph</code>	<u>275</u>
<code>\AlPh@Method@mult</code>	<u>326</u>
<code>\AlPh@Method@wrap</code>	<u>303</u>
<code>\AlPh@NewAlphaAlpha</code>	386, 391, <u>399</u>
<code>\AlPh@newalphalph</code>	361, 363, <u>367</u>

<code>\AlPh@Next</code>		F	
. 218, 235, 240, 248, 253, 257, 267		<code>\foo</code>	699, 704
<code>\AlPh@Number</code> . . . 188, 191, 221, 223,		<code>\FoundList</code>	690, 696, 706, 707
229, 262, 265, 378, 383, 388, 677		<code>\futurelet</code>	148
<code>\AlPh@ProcessAlpha</code> 279, 282, 300		H	
<code>\AlPh@ProcessBinSearch</code> 243, 247		<code>\hbox</code>	205, 515
<code>\AlPh@ProcessMult</code> 330, 333		I	
<code>\AlPh@ProcessWrap</code> 307, 310		<code>\ifAlPh@Unavailable</code>	
<code>\AlPh@StepAlpha</code> 285, 299	 187, 195, 220, 226, 252	
<code>\AlPh@StepMult</code> 336, 349		<code>\ifcase</code> . 9, 125, 165, 175, 378, 608, 670	
<code>\AlPh@StepWrap</code> 313, 323		<code>\ifdim</code> 212	
<code>\AlPh@TempA</code> 146, 153		<code>\iffalse</code> 200	
<code>\AlPh@TempB</code> 147, 155		<code>\ifnum</code> 224, 249, 250,	
<code>\AlPh@TestNumber</code> 186, 202, 219, 225, 251		270, 283, 311, 334, 401, 456, 464	
<code>\AlPh@Token</code> 148, 152		<code>\iftrue</code> 197	
<code>\AlPh@Unavailablefalse</code> 199, 203		<code>\ifx</code> 10, 12, 21,	
<code>\AlPh@Unavailabletrue</code>		45, 53, 101, 109, 123, 125, 138,	
. 196, 207, 208, 213		142, 144, 152, 350, 369, 376,	
<code>\AlphaAlpha</code> 4, 411, 547		420, 423, 426, 429, 468, 533, 539	
<code>\alphalph</code> 412, 548		<code>\immediate</code> 23, 47	
<code>\alphalph@Alpha</code> 5, 164, 411, 528, 640, 684		<code>\IncludeTests</code> 511	
<code>\alphalph@alph</code> 164, 412, 529, 641, 683		<code>\input</code> 102, 103, 469	
<code>\AlphaMult</code> 640, 646		<code>\intcalcAdd</code> 244	
<code>\alphmult</code> 641, 647		<code>\intcalcDec</code> . . . 287, 290, 314, 337, 340	
<code>\AlphaWrap</code> 528, 555		<code>\intcalcDiv</code> 290, 337	
<code>\alphwrap</code> 529, 556		<code>\intcalcInc</code> 286, 314, 337, 340	
B		<code>\intcalcMod</code> 287, 314, 340	
<code>\begin</code> 563, 591, 615, 651, 672, 687, 722		<code>\intcalcShl</code> 236	
<code>\body</code> 436, 440		<code>\intcalcShr</code> 244	
C		<code>\iterate</code> 437, 439, 441	
<code>\catch</code> 689, 690, 694, 698, 699,		K	
702, 704, 709, 710, 711, 712,		<code>\kernel@ifnextchar</code> 162	
713, 714, 715, 716, 717, 718, 719		L	
<code>\catcode</code> 3, 4, 5, 6, 7, 18,		<code>\LastSymbol</code> 532, 551	
19, 20, 34, 35, 36, 37, 38, 39, 40,		<code>\LaTeXAlphaAlpha</code> 526, 549	
41, 42, 43, 44, 62, 63, 66, 67, 68,		<code>\LaTeXalphalph</code> 527, 550	
69, 73, 74, 75, 76, 80, 82, 110,		<code>\LaTeXAlphaMult</code> 642, 648	
111, 113, 114, 416, 417, 418,		<code>\LaTeXalphmult</code> 643, 649	
419, 454, 463, 476, 477, 478,		<code>\LaTeXAlphaWrap</code> 530, 557	
479, 480, 481, 482, 483, 484, 485		<code>\LaTeXalphwrap</code> 531, 558	
<code>\count@</code> 421, 450,		<code>\LoadCommand</code> 469, 486	
454, 456, 457, 461, 463, 464, 465		<code>\LogTests</code> 512	
<code>\countdef</code> 421		<code>\loop</code> 435, 451, 462	
<code>\csname</code> . 8, 21, 45, 58, 65, 101, 109,		<code>\lowercase</code> 548, 550, 556, 558, 647, 649	
123, 138, 142, 144, 369, 387,		M	
392, 420, 423, 426, 429, 468, 490		<code>\makeatletter</code>	
D	 498, 525, 606, 639, 673, 688	
<code>\detokenize</code> 373		<code>\makeatother</code> 545, 610, 644	
<code>\DisableNumexpr</code> 500, 506, 516		<code>\MessageBreak</code> 405	
<code>\documentclass</code> 496		<code>\myvocal</code> s 607, 611	
E		<code>\myvocal</code> sB 669, 682	
<code>\empty</code> 12		N	
<code>\end</code> 491, 590, 604, 637, 667, 685, 720, 723		<code>\NeedsTeXFormat</code> 494	
<code>\endcsname</code> 8, 21, 45, 58, 65, 101, 109,		<code>\newalphalph</code> . 5, 358, 411, 412, 526,	
123, 138, 142, 144, 369, 387,		527, 528, 529, 530, 531, 611,	
392, 420, 423, 426, 429, 468, 490		640, 641, 642, 643, 676, 694, 704	
<code>\endinput</code> 30		<code>\newcommand</code> 500, 503,	
<code>\escapechar</code> 118, 381		514, 546, 554, 560, 607, 612, 645	
<code>\Expect</code> . . . 518, 520, 522, 677, 696, 707		<code>\newif</code> 195	

<code>\next</code>	441, 443, 445	589, 592, 593, 594, 595, 596,	
<code>\nofiles</code>	495	597, 598, 599, 600, 601, 602, 603	
<code>\number</code>	236, 244, 277,	<code>\TestAlphWrap</code>	552, 554
	285, 289, 305, 313, 328, 339, 393	<code>\TestCallCmd</code>	
<code>\numexpr</code> 278, 306, 329, 394, 499, 501, 504			547, 548, 549, 550, 555, 556,
			557, 558, 560, 646, 647, 648, 649
P			
<code>\PackageInfo</code>	26	<code>\TestCmd</code>	514, 561, 613, 675, 676
<code>\ProvidesPackage</code>	59	<code>\testcmd</code> ..	693, 694, 695, 703, 704, 705
R			
<code>\RangeCatcodeInvalid</code>		<code>\TestMult</code>	645, 652, 653,
	460, 472, 473, 474, 475		654, 655, 656, 657, 658, 659,
<code>\repeat</code>	435, 447, 458, 466		660, 661, 662, 663, 664, 665, 666
<code>\RequirePackage</code>	105, 106	<code>\TestString</code>	517, 518, 519, 520
<code>\RestoreCatcodes</code> ..	449, 452, 453, 487	<code>\TestVocals</code>	612, 616, 617,
<code>\RestoreNumexpr</code>	503, 509		618, 619, 620, 621, 622, 623,
<code>\Result</code>	700, 701, 707		624, 625, 626, 627, 628, 629,
<code>\romannumeral</code>	336		630, 631, 632, 633, 634, 635, 636
S			
<code>\saved@numexpr</code>	499, 504	<code>\the</code>	66, 67, 68,
<code>\SavedCatch</code>	698, 702		69, 80, 278, 306, 329, 394, 454, 522
<code>\setbox</code>	205, 515	<code>\TMP@EnsureCode</code>	
T			
<code>\Test</code>	471, 489,		77, 84, 85, 86, 87, 88, 89,
	674, 679, 680, 681, 682, 683,		90, 91, 92, 93, 94, 95, 96, 97, 98, 99
	684, 692, 709, 710, 711, 712,	U	
	713, 714, 715, 716, 717, 718, 719	<code>\uppercase</code>	547, 549, 555, 557, 646, 648
<code>\TestAlph</code>	546, 564,	<code>\usepackage</code>	508, 510
	565, 566, 567, 568, 569, 570,	V	
	571, 572, 573, 574, 575, 576,	<code>\vocalsvocals</code>	611, 613
	577, 578, 579, 580, 581, 582,	W	
	583, 584, 585, 586, 587, 588,	<code>\wd</code>	212, 522
X			
		<code>\WrapString</code>	551, 552
		<code>\write</code>	23, 47
		X	
		<code>\x</code>	8, 10, 12, 22, 26, 28, 46, 51, 58, 64, 72