

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	7
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	11
2.8.1	Common methods	11
2.8.2	Method ‘alph’	12
2.8.3	Method ‘wrap’	12
2.8.4	Method ‘mult’	13
2.9	User interface	14
3	Test	15
3.1	Catcode checks for loading	15
4	Macro tests	16

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

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 \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 \expandafter\ifx\csname PackageInfo\endcsname\relax
19 \def\x#1#2{%
20 \immediate\write-1{Package #1 Info: #2.}%
21 }%
22 \else
23 \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
24 \fi
25 \x{alphalph}{The package is already loaded}%
26 \endgroup
27 \expandafter\endinput
28 \fi
29 \endgroup
```

Package identification:

```

30 \begingroup
31 \catcode40 12 % (
32 \catcode41 12 % )
33 \catcode44 12 % ,
34 \catcode45 12 % -
35 \catcode46 12 % .
36 \catcode47 12 % /
37 \catcode58 12 % :
38 \catcode64 11 % @
39 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
40 \def\x#1#2#3[#4]{\endgroup
41 \immediate\write-1{Package: #3 #4}%
42 \xdef#1{#4}%
43 }%
44 \else
45 \def\x#1#2[#3]{\endgroup
46 #2[{#3}]}%
47 \ifx#1\relax
48 \xdef#1{#3}%
49 \fi
50 }%
51 \fi
52 \expandafter\x\csname ver@alphalph.sty\endcsname
53 \ProvidesPackage{alphalph}%
54 [2007/09/09 v2.0 Converting numbers to letters (H0)]

```

2.2 Catcodes

```

55 \expandafter\edef\csname AlPh@AtEnd\endcsname{%
56 \catcode64 \the\catcode64\relax
57 }
58 \catcode64 11 % @
59 \def\TMP@EnsureCode#1#2{%
60 \edef\AlPh@AtEnd{%
61 \AlPh@AtEnd
62 \catcode#1 \the\catcode#1\relax
63 }%
64 \catcode#1 #2\relax
65 }
66 \TMP@EnsureCode{33}{12}% !
67 \TMP@EnsureCode{39}{12}% '
68 \TMP@EnsureCode{40}{12}% (
69 \TMP@EnsureCode{41}{12}% )
70 \TMP@EnsureCode{43}{12}% +
71 \TMP@EnsureCode{44}{12}% ,
72 \TMP@EnsureCode{46}{12}% .
73 \TMP@EnsureCode{47}{12}% /
74 \TMP@EnsureCode{59}{12}% ;
75 \TMP@EnsureCode{60}{12}% <
76 \TMP@EnsureCode{61}{12}% =
77 \TMP@EnsureCode{62}{12}% >
78 \TMP@EnsureCode{91}{12}% [
79 \TMP@EnsureCode{93}{12}% ]
80 \TMP@EnsureCode{96}{12}% '
81 \TMP@EnsureCode{124}{12}% |

```

2.3 Package loading

```

82 \begingroup\expandafter\expandafter\expandafter\endgroup
83 \expandafter\ifx\csname RequirePackage\endcsname\relax
84 \input infwarerr.sty\relax
85 \input intcalc.sty\relax
86 \else
87 \RequirePackage{infwarerr}[2007/09/09]%

```

```

88 \RequirePackage{intcalc}[2007/09/09]%
89 \fi

```

2.4 ε -T_EX detection

```

90 \begingroup\expandafter\expandafter\expandafter\endgroup
91 \expandafter\ifx\csname numexpr\endcsname\relax
92 \catcode124=9 % '!': ignore
93 \catcode43=14 % '+': comment
94 \else
95 \catcode124=14 % '!': comment
96 \catcode43=9 % '+': ignore
97 \fi

```

2.5 Help macros

`\AlPh@Error`

```

98 \def\AlPh@Error#1{%
99 \begingroup
100 \escapechar=92 % backslash
101 \@PackageError{alphalph}{#1}\@ehc
102 \endgroup
103 }

```

`\AlPh@ifDefinable`

```

104 \begingroup\expandafter\expandafter\expandafter\endgroup
105 \expandafter\ifx\csname @ifdefinable\endcsname\relax
106 \def\AlPh@ifDefinable#1#2{%
107 \ifcase\ifx#1\@undefined\else\ifx#1\relax\else1\fi\fi0 %
108 #2%
109 \else
110 \AlPh@Error{%
111 Command \string#1 already defined%
112 }%
113 \fi
114 }%
115 \else

```

`\AlPh@ifDefinable`

```

116 \let\AlPh@ifDefinable\@ifdefinable
117 \fi

```

`\@ReturnAfterElseFi`

`\@ReturnAfterFi`

The following commands moves the ‘then’ and ‘else’ part respectively behind the `\if`-construct. This prevents a too deep `\if`-nesting and so a T_EX 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.)

```

118 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
119 \long\def\@ReturnAfterFi#1\fi{\fi#1}

```

`\@gobblefour`

L^AT_EX defines commands for eating arguments. Define `\@gobblefour` if it is not defined (plain-T_EX).

```

120 \expandafter\ifx\csname @gobblefour\endcsname\relax
121 \long\def\@gobblefour#1#2#3#4{%
122 \fi

```

`AlPh@ifOptArg`

```

123 \begingroup\expandafter\expandafter\expandafter\endgroup
124 \expandafter\ifx\csname kernel@ifnextchar\endcsname\relax
125 \begingroup\expandafter\expandafter\expandafter\endgroup

```



```

126 \expandafter\ifx\csname @ifnextchar\endcsname\relax
127   \def\AlPh@ifOptArg#1#2{%
128     \def\AlPh@TempA{#1}%
129     \def\AlPh@TempB{#2}%
130     \futurelet\AlPh@Token\AlPh@ifOptArgNext
131   }%
132   \let\AlPh@BracketLeft=[%
133   \def\AlPh@ifOptArgNext{%
134     \ifx\AlPh@Token\AlPh@BracketLeft
135       \expandafter\AlPh@TempA
136     \else
137       \expandafter\AlPh@TempB
138     \fi
139   }%
140 \else
141   \def\AlPh@ifOptArg{\@ifnextchar[%]}
142 \fi
143 \else
144   \def\AlPh@ifOptArg{\kernel@ifnextchar[%]}
145 \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
`\alphalph@alph` (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).

```

146 \def\alphalph@Alph#1{%
147   \ifcase#1%
148     @%
149     \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%
150     \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%
151   \else
152     \AlPh@ctrerr
153     @%
154   \fi
155 }
156 \def\alphalph@alph#1{%
157   \ifcase#1%
158     @%
159     \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%
160     \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%
161   \else
162     \AlPh@ctrerr
163     @%
164   \fi
165 }

```

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

```

166 \def\AlPh@ctrerr{}

```

2.7 Finding number of symbols

```

\AlPh@GetNumberOfSymbols #1: symbols provider
167 \def\AlPh@GetNumberOfSymbols#1{%
168   \AlPh@TestNumber1!{#1}%
169   \ifAlPh@Unavailable
170     \def\AlPh@Number{0}%
171     \AlPh@Error{No symbols found}%
172   \else
173     \def\AlPh@Number{1}%
174     \AlPh@ExpSearch2!{#1}%
175   \fi
176 }

\ifAlPh@Unavailable
177 \newif\ifAlPh@Unavailable
178 \def\AlPh@Unavailabletrue{%
179   \global\let\ifAlPh@Unavailable\iftrue
180 }
181 \def\AlPh@Unavailablefalse{%
182   \global\let\ifAlPh@Unavailable\iffalse
183 }

\AlPh@TestNumber #1: number to be tested
#2: symbols provider
184 \def\AlPh@TestNumber#1!#2{%
185   \AlPh@Unavailablefalse
186   \begingroup
187     \setbox0=\hbox{%
188       \begingroup % color
189       \let\@ctrerr\AlPh@Unavailabletrue
190       \let\AlPh@ctrerr\AlPh@Unavailabletrue
191       #2{#1}%
192     \endgroup
193   }%
194   \ifdim\wd0=Opt %
195     \AlPh@Unavailabletrue
196   \fi
197 \endgroup
198 }

\AlPh@ExpSearch #1: number to be tested
#2: symbols provider
199 \def\AlPh@ExpSearch#1!#2{%
200   \let\AlPh@Next\relax
201   \AlPh@TestNumber#1!{#2}%
202   \ifAlPh@Unavailable
203     \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
204   \else
205     \def\AlPh@Number{#1}%
206     \ifnum#1>1073741823 %
207       \AlPh@TestNumber2147483647!{#2}%
208       \ifAlPh@Unavailable
209         \AlPh@BinSearch#1!2147483647!{#2}%
210       \else
211         \def\AlPh@Number{0}%
212         \AlPh@Error{%
213           Maximal symbol number not found%
214         }%
215       \fi
216     \else
217       \def\AlPh@Next{%

```

```

218         \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}!{#2}%
219     }%
220     \fi
221 \fi
222 \AlPh@Next
223 }

\AlPh@BinSearch #1: available number
#2: unavailable number, #2 > #1
#3: symbols provider
224 \def\AlPh@BinSearch#1!#2!#3{%
225     \expandafter\AlPh@ProcessBinSearch
226     \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
227     #1!#2!{#3}%
228 }

\AlPh@ProcessBinSearch #1: number to be tested, #2 ≤ #1 ≤ #3
#2: available number
#3: unavailable number
#4: symbols provider
229 \def\AlPh@ProcessBinSearch#1!#2!#3!#4{%
230     \let\AlPh@Next\relax
231     \ifnum#1>#2 %
232         \ifnum#1<#3 %
233             \AlPh@TestNumber#1!{#4}%
234             \ifAlPh@Unavailable
235                 \def\AlPh@Next{%
236                     \AlPh@BinSearch#2!#1!{#4}%
237                 }%
238             \else
239                 \def\AlPh@Next{%
240                     \AlPh@BinSearch#1!#3!{#4}%
241                 }%
242             \fi
243         \else
244             \def\AlPh@Number{#2}%
245             \fi
246         \else
247             \def\AlPh@Number{#2}%
248             \fi
249         \AlPh@Next
250 }

```

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)
251 \def\AlPh@CheckPositive#1!#2{%
252     \ifnum#1<1 %
253         \expandafter\@gobblefour
254     \fi
255     #2{#1}%

```

256 }

2.8.2 Method ‘alph’

```
\AlPh@Method@alph #1: number of symbols
#2: symbols provider
#3: number to be converted
257 \def\AlPh@Method@alph#1#2#3{%
258   \expandafter\AlPh@CheckPositive
259 |   \number#3!%
260 +   \the\numexpr#3!%
261   \AlPh@ProcessAlph
262   {#1}{#2}%
263 }

\AlPh@ProcessAlph #1: current number
#2: number of symbols
#3: symbols provider
264 \def\AlPh@ProcessAlph#1#2#3{%
265   \ifnum#1>#2 %
266     \@ReturnAfterElseFi{%
267       \expandafter\AlPh@StepAlph\number
268       \intcalcInc{%
269         \intcalcMod{\intcalcDec{#1}}{#2}%
270       }%
271       \expandafter!\number
272       \intcalcDiv{\intcalcDec{#1}}{#2}%
273       !{#2}{#3}%
274     }%
275   \else
276     \@ReturnAfterFi{%
277       #3{#1}%
278     }%
279   \fi
280 }

\AlPh@StepAlph #1: current last digit
#2: new current number
#3: number of symbols
#4: symbols provider
281 \def\AlPh@StepAlph#1!#2!#3#4{%
282   \AlPh@ProcessAlph{#2}{#3}{#4}%
283   #4{#1}%
284 }
```

2.8.3 Method ‘wrap’

```
\AlPh@Method@wrap #1: number of symbols
#2: symbols provider
#3: number to be converted
285 \def\AlPh@Method@wrap#1#2#3{%
286   \expandafter\AlPh@CheckPositive
287 |   \number#3!%
288 +   \the\numexpr#3!%
289   \AlPh@ProcessWrap
290   {#1}{#2}%
291 }

\AlPh@ProcessWrap #1: number to be converted
#2: number of symbols
#3: symbols provider
```

```

292 \def\AlPh@ProcessWrap#1#2#3{%
293   \ifnum#1>#2 %
294     \@ReturnAfterElseFi{%
295       \expandafter\AlPh@StepWrap\number
296       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
297       !{#3}%
298     }%
299   \else
300     \@ReturnAfterFi{%
301       #3{#1}%
302     }%
303   \fi
304 }

```

```

\AlPh@StepWrap #1: final number
                #2: symbols provider
305 \def\AlPh@StepWrap#1!#2{%
306   #2{#1}%
307 }

```

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
308 \def\AlPh@Method@mult#1#2#3{%
309   \expandafter\AlPh@CheckPositive
310 |   \number#3!%
311 +   \the\numexpr#3!%
312   \AlPh@ProcessMult
313   {#1}{#2}%
314 }

\AlPh@ProcessMult #1: number to be converted
                  #2: number of symbols
                  #3: symbols provider
315 \def\AlPh@ProcessMult#1#2#3{%
316   \ifnum#1>#2 %
317     \@ReturnAfterElseFi{%
318       \expandafter\AlPh@StepMult\romannumeral
319       \intcalcInc{\intcalcDiv{\intcalcDec{#1}}{#2}}%
320       000%
321       \expandafter!\number
322       \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
323       !{#3}%
324     }%
325   \else
326     \@ReturnAfterFi{%
327       #3{#1}%
328     }%
329   \fi
330 }

```

```

\AlPh@StepMult #1#2: repetitions coded as list of character ‘m’
#3: symbol slot
#4: symbols provider
331 \def\AlPh@StepMult#1#2!#3!#4{%
332   \ifx\\#2\\%
333   \else
334     \@ReturnAfterFi{%
335       \AlPh@StepMult#2!#3!#4}%
336   }%
337 \fi
338 #4{#3}%
339 }

```

2.9 User interface

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

```

#1: cmd
[#2]: method name: alph (default), wrap, mult
hash-ok #3: symbols provider
#4: number of symbols
340 \AlPh@IfDefinable\newalphalph{%
341   \def\newalphalph#1{%
342     \AlPh@IfOptArg{%
343       \AlPh@newalphalph{#1}%
344     }{%
345       \AlPh@newalphalph{#1}[alph]%
346     }%
347   }%
348 }

```

```

\AlPh@newalphalph #1: cmd #2: method name
#3: symbols provider
#4: number of symbols
349 \def\AlPh@newalphalph#1[#2]#3#4{%
350   \begingroup\expandafter\expandafter\expandafter\endgroup
351   \expandafter\ifx\csname AlPh@Method@#2\endcsname\relax
352     \AlPh@Error{%
353       Unknown method %
354 |     ‘#2’%
355 +     ‘\detokenize{#2}’%
356   }%
357   \else
358     \ifx\\#4\\%
359       \AlPh@GetNumberOfSymbols{#3}%
360       \ifcase\AlPh@Number
361       \else
362         \begingroup
363         \escapechar=92 % backslash
364         \@PackageInfo{alphalph}{%
365           Number of symbols for \string#1 is \AlPh@Number
366         }%
367         \endgroup
368         \expandafter\AlPh@NewAlphaAlpha
369         \csname AlPh@Method@#2\expandafter\endcsname
370         \AlPh@Number!{#1}{#3}%
371       \fi
372     \else
373       \expandafter\AlPh@NewAlphaAlpha
374       \csname AlPh@Method@#2\expandafter\endcsname
375 |     \number#4!%

```

```

376 + \the\numexpr#4!%
377 {#1}{#3}%
378 \fi
379 \fi
380 }%

\AlPh@NewAlphAlph #1: method macro
#2: number of symbols
#3: cmd
#4: symbols provider
381 \def\AlPh@NewAlphAlph#1#2!#3#4{%
382 \AlPh@IfDefinable#3{%
383 \ifnum#2>0 %
384 \def#3{#1{#2}{#4}}%
385 \else
386 \AlPh@Error{%
387 Definition of \string#3 failed,\MessageBreak
388 because number of symbols (#2) is not positive%
389 }%
390 \fi
391 }%
392 }

\AlphAlph
393 \newalphalph\AlphAlph\alphalph@Alph{26}

\alphalph
394 \newalphalph\alphalph\alphalph@alph{26}

395 \AlPh@AtEnd
396 \end{package}

```

3 Test

3.1 Catcode checks for loading

```

397 \test1
398 \catcode'\@=11 %
399 \def\RestoreCatcodes{
400 \count@=0 %
401 \loop
402 \edef\RestoreCatcodes{%
403 \RestoreCatcodes
404 \catcode\the\count@=\the\catcode\count@\relax
405 }%
406 \ifnum\count@<255 %
407 \advance\count@\@ne
408 \repeat
409
410 \def\RangeCatcodeInvalid#1#2{%
411 \count@=#1\relax
412 \loop
413 \catcode\count@=15 %
414 \ifnum\count@<#2\relax
415 \advance\count@\@ne
416 \repeat
417 }
418 \def\Test{%
419 \RangeCatcodeInvalid{0}{47}%
420 \RangeCatcodeInvalid{58}{64}%
421 \RangeCatcodeInvalid{91}{96}%

```

```

422 \RangeCatcodeInvalid{123}{255}%
423 \catcode'\@=12 %
424 \catcode'\=0 %
425 \catcode'\{=1 %
426 \catcode'\}=2 %
427 \catcode'\#=6 %
428 \catcode'\[=12 %
429 \catcode'\]=12 %
430 \catcode'\%=14 %
431 \catcode'\_ =10 %
432 \catcode13=5 %
433 \input alphasph.sty\relax
434 \RestoreCatcodes
435 }
436 \Test
437 \csname @@end\endcsname
438 \end
439 </test1>

```

4 Macro tests

```

440 <*test2>
441 \NeedsTeXFormat{LaTeX2e}
442 \nofiles
443 \documentclass{article}
444 <*noetex>
445 \makeatletter
446 \let\saved@numexpr\numexpr
447 \newcommand*\DisableNumexpr{%
448   \let\numexpr\@undefined
449 }
450 \newcommand*\RestoreNumexpr{%
451   \let\numexpr\saved@numexpr
452 }
453 \DisableNumexpr
454 </noetex>
455 \usepackage{alphalph}[2007/09/09]
456 <noetex>\RestoreNumexpr
457 \usepackage{qstest}
458 \IncludeTests{*}
459 \LogTests{log}{*}{*}
460
461 \newcommand*\TestCmd}[3]{%
462   \setbox0=\hbox{%
463     <noetex> \DisableNumexpr
464     \edef\TestString{#1{#2}}%
465     \expandafter\Expect\expandafter{\TestString}{#3}%
466     \edef\TestString{#1{#2} }%
467     \expandafter\Expect\expandafter{\TestString}{#3 }%
468   }%
469   \Expect*{\the\wd0}{0.0pt}%
470 }
471
472 \makeatletter
473 \newalphalph\LaTeXAlphAlph\@Alph{26}
474 \newalphalph\LaTeXalphalph\@alph{26}
475 \newalphalph\AlphWrap[wrap]\alphalph@Alph{26}
476 \newalphalph\alphwrap[wrap]\alphalph@alph{26}
477 \newalphalph\LaTeXAlphWrap[wrap]\@Alph{26}
478 \newalphalph\LaTeXalphwrap[wrap]\@alph{26}
479 \def\LastSymbol#1{%
480   \ifx\#1\%

```



```

481 \else
482   \@LastSymbol#1\@nil
483 \fi
484 }
485 \def\@LastSymbol#1#2\@nil{%
486   \ifx\#2\%
487     #1%
488   \else
489     \@LastSymbol#2\@nil
490   \fi
491 }
492 \makeatother
493 \newcommand*\TestAlph}[2]{%
494   \uppercase{\TestCallCmd\AlphAlph{#2}}{#1}%
495   \lowercase{\TestCallCmd\alphalph{#2}}{#1}%
496   \uppercase{\TestCallCmd\LaTeXAlphAlph{#2}}{#1}%
497   \lowercase{\TestCallCmd\LaTeXalphalph{#2}}{#1}%
498   \edef\WrapString{\LastSymbol{#2}}%
499   \expandafter\TestAlphWrap\expandafter{\WrapString}{#1}%
500 }
501 \newcommand*\TestAlphWrap}[2]{%
502   \uppercase{\TestCallCmd\AlphWrap{#1}}{#2}%
503   \lowercase{\TestCallCmd\alphwrap{#1}}{#2}%
504   \uppercase{\TestCallCmd\LaTeXAlphWrap{#1}}{#2}%
505   \lowercase{\TestCallCmd\LaTeXalphwrap{#1}}{#2}%
506 }
507 \newcommand*\TestCallCmd}[3]{%
508   \TestCmd#1{#3}{#2}%
509 }
510 \begin{qstest}{AlphSymbols}{alphalph, AlphAlph, symbols}
511   \TestAlph{1}{a}%
512   \TestAlph{2}{b}%
513   \TestAlph{3}{c}%
514   \TestAlph{4}{d}%
515   \TestAlph{5}{e}%
516   \TestAlph{6}{f}%
517   \TestAlph{7}{g}%
518   \TestAlph{8}{h}%
519   \TestAlph{9}{i}%
520   \TestAlph{10}{j}%
521   \TestAlph{11}{k}%
522   \TestAlph{12}{l}%
523   \TestAlph{13}{m}%
524   \TestAlph{14}{n}%
525   \TestAlph{15}{o}%
526   \TestAlph{16}{p}%
527   \TestAlph{17}{q}%
528   \TestAlph{18}{r}%
529   \TestAlph{19}{s}%
530   \TestAlph{20}{t}%
531   \TestAlph{21}{u}%
532   \TestAlph{22}{v}%
533   \TestAlph{23}{w}%
534   \TestAlph{24}{x}%
535   \TestAlph{25}{y}%
536   \TestAlph{26}{z}%
537 \end{qstest}
538 \begin{qstest}{AlphRange}{alphalph, range}
539   \TestAlph{0}{}%
540   \TestAlph{-1}{}%
541   \TestAlph{-2147483647}{}%
542   \TestAlph{27}{aa}%

```

```

543 \TestAlph{28}{ab}%
544 \TestAlph{52}{az}%
545 \TestAlph{53}{ba}%
546 \TestAlph{78}{bz}%
547 \TestAlph{79}{ca}%
548 \TestAlph{702}{zz}%
549 \TestAlph{703}{aaa}%
550 \TestAlph{2147483647}{fxshrxw}%
551 \end{qstest}
552
553 \makeatletter
554 \newcommand*{\myvocal}{1}{%
555   \ifcase#1X\or A\or E\or I\or O\or U\else Y\fi
556 }
557 \makeatother
558 \newalphalph\vocalsvocal{\myvocal}{5}
559 \newcommand*{\TestVocal}{%
560   \TestCmd\vocalsvocal
561 }
562 \begin{qstest}{\vocal}{\vocal}
563   \TestVocal{0}{}%
564   \TestVocal{1}{A}%
565   \TestVocal{2}{E}%
566   \TestVocal{3}{I}%
567   \TestVocal{4}{O}%
568   \TestVocal{5}{U}%
569   \TestVocal{6}{AA}%
570   \TestVocal{7}{AE}%
571   \TestVocal{8}{AI}%
572   \TestVocal{9}{AO}%
573   \TestVocal{10}{AU}%
574   \TestVocal{11}{EA}%
575   \TestVocal{24}{OO}%
576   \TestVocal{25}{OU}%
577   \TestVocal{26}{UA}%
578   \TestVocal{29}{UO}%
579   \TestVocal{30}{UU}%
580   \TestVocal{31}{AAA}%
581   \TestVocal{155}{UUU}%
582   \TestVocal{156}{AAAA}%
583   \TestVocal{2147483647}{AIIIOEEIOIIUOE}%
584 \end{qstest}
585
586 \makeatletter
587 \newalphalph\AlphMult[mult]{\alphalph@Alph}{26}
588 \newalphalph\alphmult[mult]{\alphalph@alph}{26}
589 \newalphalph\LaTeXAlphMult[mult]{\@Alph}{26}
590 \newalphalph\LaTeXalphmult[mult]{\@alph}{26}
591 \makeatother
592 \newcommand*{\TestMult}[2]{%
593   \uppercase{\TestCallCmd\AlphMult{#2}}{#1}%
594   \lowercase{\TestCallCmd\alphmult{#2}}{#1}%
595   \uppercase{\TestCallCmd\LaTeXAlphMult{#2}}{#1}%
596   \lowercase{\TestCallCmd\LaTeXalphmult{#2}}{#1}%
597 }
598 \begin{qstest}{\mult}{\mult}
599   \TestMult{0}{}%
600   \TestMult{-1}{}%
601   \TestMult{-2147483647}{}%
602   \TestMult{1}{a}%
603   \TestMult{2}{b}%
604   \TestMult{26}{z}%

```

```

605 \TestMult{27}{aa}%
606 \TestMult{28}{bb}%
607 \TestMult{52}{zz}%
608 \TestMult{53}{aaa}%
609 \TestMult{54}{bbb}%
610 \TestMult{259}{yyyyyyyyyy}%
611 \TestMult{260}{zzzzzzzzzz}%
612 \TestMult{261}{aaaaaaaaaaa}%
613 \TestMult{262}{bbbbbbbbbbb}%
614 \end{qstest}
615
616 \def\myvocalB#1{%
617 \ifcase#1\or A\or E\or I\or O\or U\fi
618 }
619 \begin{qstest}{symbolnum}{symbolnum}
620 \makeatletter
621 \def\Test#1#2{%
622 \let\TestCmd\relax
623 \newalphalph\TestCmd{#1}{}%
624 \Expect*{\AlPh@Number}{#2}%
625 }%
626 \Test\@alph{26}%
627 \Test\@Alph{26}%
628 \Test\@fnsymbol{9}%
629 \Test\myvocalB{5}%
630 \Test\alphalph@alph{26}%
631 \Test\alphalph@Alph{26}%
632 \end{qstest}
633
634 \begin{qstest}{list}{list}
635 \makeatletter
636 \def\catch#1\relax{%
637 \def\FoundList{\catch#1}%
638 }%
639 \def\Test[#1]#2#3#4{%
640 \let\testcmd\relax
641 \newalphalph\testcmd[{#1}]{\catch}{#2}%
642 \testcmd{#3}|\relax
643 \expandafter\Expect\expandafter{\FoundList}{#4|}%
644 %
645 \let\SavedCatch\catch
646 \def\catch{\noexpand\catch\noexpand\foo}%
647 \edef\Result{#4|}%
648 \@onelevel@sanitize\Result
649 \let\catch\SavedCatch
650 \let\testcmd\relax
651 \newalphalph\testcmd[{#1}]{\catch\foo}{#2}%
652 \testcmd{#3}|\relax
653 \@onelevel@sanitize\FoundList
654 \Expect*{\FoundList}*{\Result}%
655 }%
656 \Test[alph]{26}{3}{\catch{3}}%
657 \Test[alph]{26}{12}{\catch{12}}%
658 \Test[alph]{26}{27}{\catch{1}\catch{1}}%
659 \Test[alph]{26}{78}{\catch{2}\catch{26}}%
660 \Test[wrap]{26}{7}{\catch{7}}%
661 \Test[wrap]{26}{14}{\catch{14}}%
662 \Test[wrap]{26}{80}{\catch{2}}%
663 \Test[wrap]{26}{700}{\catch{24}}%
664 \Test[mult]{26}{4}{\catch{4}}%
665 \Test[mult]{26}{17}{\catch{17}}%
666 \Test[mult]{26}{54}{\catch{2}\catch{2}\catch{2}}%

```

```

667 \end{qstest}
668
669 \begin{document}
670 \end{document}
671 \end{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:macros/latex/contrib/oberdiek/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`.

¹<http://ftp.ctan.org/tex-archive/>

5.4 Refresh file name databases

If your $\text{T}_{\text{E}}\text{X}$ distribution ($\text{t}_{\text{E}}\text{T}_{\text{E}}\text{X}$, $\text{m}_{\text{ik}}\text{T}_{\text{E}}\text{X}$, ...) relies on file name databases, you must refresh these. For example, $\text{t}_{\text{E}}\text{T}_{\text{E}}\text{X}$ users run `texhash` or `mktextlsr`.

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 alphas.pdf unpack_files output .
```

Unpacking with $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$. The `.dtx` chooses its action depending on the format:

plain- $\text{T}_{\text{E}}\text{X}$: Run `docstrip` and extract the files.

$\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$: Generate the documentation.

If you insist on using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ for `docstrip` (really, `docstrip` does not need $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$), then inform the autodetect routine about your intention:

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

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

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

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

An example follows how to generate the documentation with `pdf $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$` :

```
pdflatex alphaph.dtx
makeindex -s gind.ist alphaph.idx
pdflatex alphaph.dtx
makeindex -s gind.ist alphaph.idx
pdflatex alphaph.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 - $\text{T}_{\text{E}}\text{X}$ support added.

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)

[1999/04/13 v1.1]

- Minor documentation change.
- First CTAN release.

[1999/06/26 v1.2]

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

[2006/02/20 v1.3]

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

[2006/05/30 v1.4]

- `\newalphalph` added.

[2007/04/11 v1.5]

- Line ends sanitized.

[2007/09/09 v2.0]

- New implementation that uses package `\intcalc`. This removes the dependency on ε - \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>	427	<code>\@nil</code> 482, 485, 489	
<code>\%</code>	430	<code>\@onelevel@sanitize</code> 648, 653	
<code>\@</code>	398, 423	<code>\@undefined</code> 107, 448	
<code>\@Alph</code>	473, 477, 589, 627	<code>\[</code>	428
<code>\@LastSymbol</code>	482, 485, 489	<code>\]</code>	332, 358, 424, 480, 486
<code>\@PackageError</code>	101	<code>\{</code>	425
<code>\@PackageInfo</code>	364	<code>\}</code>	426
<code>\@ReturnAfterElseFi</code> <u>118</u> , 266, 294, 317		<code>\]</code>	429
<code>\@ReturnAfterFi</code> <u>118</u> , 276, 300, 326, 334			
<code>\@alph</code>	474, 478, 590, 626	<code>\sqcup</code>	431
<code>\@ctrerr</code>	189		
<code>\@ehc</code>	101	A	
<code>\@fnsymbol</code>	628	<code>\advance</code>	407, 415
<code>\@gobblefour</code>	<u>120</u> , 253	<code>\AlPh@AtEnd</code>	60, 61, 395
<code>\@ifdefinable</code>	116	<code>\AlPh@BinSearch</code> 203, 209, <u>224</u> , 236, 240	
<code>\@ifnextchar</code>	141	<code>\AlPh@BracketLeft</code>	132, 134
<code>\@ne</code>	407, 415	<code>\AlPh@CheckPositive</code> <u>251</u> , 258, 286, 309	
		<code>\AlPh@ctrerr</code>	152, 162, 166, 190

<code>\AlPh@Error</code> . 98, 110, 171, 212, 352, 386	<code>\endcsname</code> 8, 18, 39, 52, 55, 83, 91, 105, 120, 124, 126, 351, 369, 374, 437
<code>\AlPh@ExpSearch</code> 174, 199	<code>\endinput</code> 27
<code>\AlPh@GetNumberOfSymbols</code> . . . 167, 359	<code>\escapechar</code> 100, 363
<code>\AlPh@IfDefinable</code> . 104, 116, 340, 382	<code>\Expect</code> . . . 465, 467, 469, 624, 643, 654
<code>\AlPh@IfOptArg</code> 123, 127, 141, 144, 342	
<code>\AlPh@IfOptArgNext</code> 130, 133	
<code>\AlPh@Method@alph</code> 257	
<code>\AlPh@Method@mult</code> 308	
<code>\AlPh@Method@wrap</code> 285	
<code>\AlPh@NewAlphaAlph</code> 368, 373, 381	
<code>\AlPh@newalphalph</code> 343, 345, 349	
<code>\AlPh@Next</code> 200, 217, 222, 230, 235, 239, 249	
<code>\AlPh@Number</code> . . . 170, 173, 203, 205, 211, 244, 247, 360, 365, 370, 624	
<code>\AlPh@ProcessAlph</code> 261, 264, 282	
<code>\AlPh@ProcessBinSearch</code> 225, 229	
<code>\AlPh@ProcessMult</code> 312, 315	
<code>\AlPh@ProcessWrap</code> 289, 292	
<code>\AlPh@StepAlph</code> 267, 281	
<code>\AlPh@StepMult</code> 318, 331	
<code>\AlPh@StepWrap</code> 295, 305	
<code>\AlPh@TempA</code> 128, 135	
<code>\AlPh@TempB</code> 129, 137	
<code>\AlPh@TestNumber</code> 168, 184, 201, 207, 233	
<code>\AlPh@Token</code> 130, 134	
<code>\AlPh@Unavailablefalse</code> 181, 185	
<code>\AlPh@Unavailabletrue</code> 178, 189, 190, 195	
<code>\AlphaAlph</code> 4, 393, 494	
<code>\alphalph</code> 394, 495	
<code>\alphalph@Alph</code> 5, 146, 393, 475, 587, 631	
<code>\alphalph@alph</code> 146, 394, 476, 588, 630	
<code>\AlphMult</code> 587, 593	
<code>\alphmult</code> 588, 594	
<code>\AlphWrap</code> 475, 502	
<code>\alphwrap</code> 476, 503	
B	K
<code>\begin</code> 510, 538, 562, 598, 619, 634, 669	<code>\kernel@ifnextchar</code> 144
C	L
<code>\catch</code> 636, 637, 641, 645, 646, 649, 651, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666	<code>\LastSymbol</code> 479, 498
<code>\catcode</code> 3, 4, 5, 6, 7, 31, 32, 33, 34, 35, 36, 37, 38, 56, 58, 62, 64, 92, 93, 95, 96, 398, 404, 413, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432	<code>\LaTeXAlphAlph</code> 473, 496
<code>\count@</code> 400, 404, 406, 407, 411, 413, 414, 415	<code>\LaTeXalphalph</code> 474, 497
<code>\csname</code> 8, 18, 39, 52, 55, 83, 91, 105, 120, 124, 126, 351, 369, 374, 437	<code>\LaTeXAlphMult</code> 589, 595
	<code>\LaTeXalphmult</code> 590, 596
	<code>\LaTeXAlphWrap</code> 477, 504
	<code>\LaTeXalphwrap</code> 478, 505
	<code>\LogTests</code> 459
	<code>\loop</code> 401, 412
	<code>\lowercase</code> 495, 497, 503, 505, 594, 596
D	M
<code>\detokenize</code> 355	<code>\makeatletter</code> 445, 472, 553, 586, 620, 635
<code>\DisableNumexpr</code> 447, 453, 463	<code>\makeatother</code> 492, 557, 591
<code>\documentclass</code> 443	<code>\MessageBreak</code> 387
	<code>\myvocal</code> s 554, 558
	<code>\myvocal</code> sB 616, 629
E	N
<code>\empty</code> 12	<code>\NeedsTeXFormat</code> 441
<code>\end</code> 438, 537, 551, 584, 614, 632, 667, 670	

<code>\newalphalph</code> .	5, 340, 393, 394, 473, 474, 475, 476, 477, 478, 558, 587, 588, 589, 590, 623, 641, 651	524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550	
<code>\newcommand</code>	447, 450, 461, 493, 501, 507, 554, 559, 592	<code>\TestAlphWrap</code>	499, 501
<code>\newif</code>	177	<code>\TestCallCmd</code>	
<code>\nofiles</code>	442	. 494, 495, 496, 497, 502, 503, 504, 505, 507, 593, 594, 595, 596	
<code>\number</code>	218, 226, 259, 267, 271, 287, 295, 310, 321, 375	<code>\TestCmd</code>	461, 508, 560, 622, 623
<code>\numexpr</code>	260, 288, 311, 376, 446, 448, 451	<code>\testcmd</code> ..	640, 641, 642, 650, 651, 652
P			
<code>\PackageInfo</code>	23	<code>\TestMult</code>	592, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613
<code>\ProvidesPackage</code>	53	<code>\TestString</code>	464, 465, 466, 467
R			
<code>\RangeCatcodeInvalid</code>		<code>\TestVocals</code>	559, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583
.....	410, 419, 420, 421, 422	<code>\the</code> .	56, 62, 260, 288, 311, 376, 404, 469
<code>\repeat</code>	408, 416	<code>\TMP@EnsureCode</code>	
<code>\RequirePackage</code>	87, 88	59, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81
<code>\RestoreCatcodes</code> ..	399, 402, 403, 434	U	
<code>\RestoreNumexpr</code>	450, 456	<code>\uppercase</code>	494, 496, 502, 504, 593, 595
<code>\Result</code>	647, 648, 654	<code>\usepackage</code>	455, 457
<code>\romannumeral</code>	318	V	
S			
<code>\saved@numexpr</code>	446, 451	<code>\vocalsvocals</code>	558, 560
<code>\SavedCatch</code>	645, 649	W	
<code>\setbox</code>	187, 462	<code>\wd</code>	194, 469
T			
<code>\Test</code>	418, 436, 621, 626, 627, 628, 629, 630, 631, 639, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666	<code>\WrapString</code>	498, 499
<code>\TestAlph</code>	493, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523,	<code>\write</code>	20, 41
		X	
		<code>\x</code>	8, 10, 12, 19, 23, 25, 40, 45, 52