

# The `alphalph` package

Heiko Oberdiek

<oberdiek@uni-freiburg.de>

2006/05/30 v1.4

## Abstract

The package provides the new expandable commands `\alphalph` and `\AlphAlph`. They are like `\number`, but the expansion consists of lowercase and uppercase letters respectively.

## Contents

<b>1</b>	<b>Usage</b>	<b>1</b>
1.1	User commands	2
1.1.1	New commands like <code>\alphalph</code>	2
<b>2</b>	<b>Implementation</b>	<b>2</b>
2.1	Begin of package	2
2.2	Help macros	3
2.3	User commands	4
2.4	Conversion with standard $\text{\TeX}$ means	4
2.4.1	Convert the separated digits to the letter result	5
2.4.2	Addition by one	6
2.5	Conversion with $\epsilon\text{-}\text{\TeX}$ features	7
2.6	Generic version	8
2.7	End of package	9
<b>3</b>	<b>Installation</b>	<b>9</b>
3.1	Download	9
3.2	Bundle installation	9
3.3	Package installation	9
3.4	Refresh file name databases	10
3.5	Some details for the interested	10
<b>4</b>	<b>History</b>	<b>10</b>
	[1999/03/19 v0.1]	10
	[1999/04/12 v1.0]	11
	[1999/04/13 v1.1]	11
	[1999/06/26 v1.2]	11
	[2006/02/20 v1.3]	11
	[2006/05/30 v1.4]	11
<b>5</b>	<b>Index</b>	<b>11</b>

## 1 Usage

The package `alphalph` can be used with both plain- $\text{\TeX}$  and  $\text{\LaTeX}$ :

plain- $\text{\TeX}$ : `\input alphalph.sty`

**L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>:** `\usepackage{alphalph}`  
 There aren't any options.

## 1.1 User commands

`\alphalph` **\alphalph:** This works like `\number`, but the expansion consists of lowercase letters.

`\AlphAlph` **\AlphAlph:** It converts a number into uppercase letters.

Both commands have following properties:

- They are fully expandable. This means that they can safely
  - be written to a file,
  - used in moving arguments (L<sup>A</sup>T<sub>E</sub>X: they are *robust*),
  - used in a `\csname`-`\endcsname` pair.
- If the argument is zero or negative, the commands expand to nothing like `\romannumeral`.
- As argument is allowed all that can be used after a `\number`:
  - explicite constants,
  - macros that expand to a number,
  - count registers, L<sup>A</sup>T<sub>E</sub>X counter can used via `\value`, e.g.:  
`\alphalph{\value{page}}`

The following table shows, how the conversion is made:

number	1, 2, ..., 26, 27, ..., 52, 53, ..., 78, 79, ..., 702, 703, ...
<code>\alphalph</code>	a, b, ..., z, aa, ..., az, ba, ..., bz, ca, ..., zz, aaa, ...

### 1.1.1 New commands like `\alphalph`

`\newalphalph` **\newalphalph:** This macro defines a new command that acts like `\alphalph`. The use of  $\varepsilon$ -T<sub>E</sub>X is required. The macro has three arguments:

- #1:** The command to be defined.
- #2:** A macro that converts a positive number to a symbol.
- #3:** The number of available symbols.

Example:

```
\newcommand*{\myvocal}[1]{%
  \ifcase#1\or A\or E\or I\or O\or U\fi
}
\newalphalph{\vocalsvocal}{\myvocal}{5}
```

## 2 Implementation

### 2.1 Begin of package

```
1 <*package>
```

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

```
2 \begingroup
3 \expandafter\let\expandafter\x\csname ver@alphalph.sty\endcsname
4 \ifcase 0%
5 \ifx\x\relax % plain
6 \else
```

```

7      \ifx\x\empty % LaTeX
8      \else
9      1%
10     \fi
11     \fi
12   \else
13     \expandafter\ifx\csname PackageInfo\endcsname\relax
14       \def\x#1#2{%
15         \immediate\write-1{Package #1 Info: #2.}%
16       }%
17     \else
18       \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
19     \fi
20     \x{alphalph}{The package is already loaded}%
21   \endgroup
22   \expandafter\endinput
23   \fi
24 \endgroup

```

Package identification:

```

25 \begingroup
26   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
27     \def\x#1#2#3[#4]{\endgroup
28       \immediate\write-1{Package: #3 #4}%
29       \xdef#1{#4}%
30     }%
31   \else
32     \def\x#1#2[#3]{\endgroup
33       #2[{#3}]%
34       \ifx#1\relax
35         \xdef#1{#3}%
36       \fi
37     }%
38   \fi
39 \expandafter\x\csname ver@alphalph.sty\endcsname
40 \ProvidesPackage{alphalph}%
41 [2006/05/30 v1.4 Converting numbers to letters (H0)]

```

For unique command names this package uses `aa@` as prefix for internal command names. Because we need `@` as a letter we save the current catcode value.

```

42 \expandafter\edef\csname aa@atcode\endcsname{\the\catcode'\@ }
43 \catcode'\@=11

```

## 2.2 Help macros

`\@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).

```

44 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
45 \long\def\@ReturnAfterFi#1\fi{\fi#1}

```

`\aa@alph` The two commands `\aa@alph` and `\aa@Alph` convert a number into a letter (lowercase and uppercase 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.

```

46 \def\aa@alph#1{%
47   \ifcase#1%
48     @%
49   \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%
50   \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%
51   \else
52     @%

```

```

53 \fi
54 }
55 \def\aa@Alph#1{%
56 \ifcase#1%
57 @%
58 \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%
59 \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%
60 \else
61 @%
62 \fi
63 }

```

## 2.3 User commands

`\alphalph` The whole difference between `\alphalph` and `\AlphAlph` is that the output consists of lowercase or uppercase letters.

```

\AlphAlph
64 \def\alphalph{\aa@callmake\aa@alph}
65 \def\AlphAlph{\aa@callmake\aa@Alph}

```

`\aa@callmake` `\aa@callmake` converts the number in the second argument `#2` into explicit decimal digits via the  $\text{\TeX}$  primitive `\number`. (The closing curly brace stops reading the number at the latest.)

```

66 \def\aa@callmake#1#2{%
67 \expandafter\aa@make\expandafter{\number#2}#1%
68 }

```

$\varepsilon$ - $\text{\TeX}$  provides the new primitive `\numexpr`. With this command the implementation is very simple (see 2.5). Therefore the package provides two methods: a fast and simple one that uses the  $\varepsilon$ - $\text{\TeX}$  extension and a method that is restricted to the standard  $\text{\TeX}$  means.

Now we distinguish between  $\text{\TeX}$  and  $\varepsilon$ - $\text{\TeX}$  by checking whether `\numexpr` is defined or isn't. Because the  $\text{\TeX}$  primitive `\csname` defines an undefined command to be `\relax`, `\csname` is executed in a group.

```

69 \begingroup\expandafter\expandafter\expandafter\endgroup
70 \expandafter\ifx\csname numexpr\endcsname\relax

```

## 2.4 Conversion with standard $\text{\TeX}$ means

`\aa@make` `\aa@make` catches the cases, if the number is zero or negative. Then it expands to nothing like `\romannumeral`.

```

71 \def\aa@make#1#2{%
72 \ifnum#1<1
73 \else
74 \@ReturnAfterFi{%
75 \aa@process1;#1;1..#2%
76 }%
77 \fi
78 }

```

`\aa@process` `\aa@process` contains the algorithm for the conversion.  $\text{\TeX}$  doesn't provide a simple method to divide or multiply numbers in a fully expandable way. An expandable addition by one is complicated enough. Therefore `\aa@process` uses only expandable versions of additions by one. The algorithm starts with one and increments it until the size of the wanted number is reached. The intermediate number that is incremented is present in two kinds:

- the normal decimal form for the `\ifnum`-comparison,
- a digit format: the end of each digit is marked by an dot, and the digits are in reserved order. An empty digit ends this format. The meaning of a digit is here the decimal representation of a letter, the range is from 1 until 26.

Example: The aim number is 100, the intermediate number 50, so following would be on the argument stack:

50;100;24.1..\aa@alph

\aa@process increments the first argument #1 (50), and calls \aa@alphinc to increment the digit form (24.1..). The middle part with the aim number ;#2; (;100;) will not be changed. Neither \aa@process nor \aa@alphinc need the conversion command \aa@alph nor \aa@Alph. This command is read by \aa@getresult, if the digit form is ready.

The expansion motor is \number. It reads and expands token to get decimal numbers until a token is reached that isn't a decimal digit. So the expansion doesn't stop, if \aa@inc is ready, because \aa@inc produces only decimal digits. \aa@alphinc is expanded to look for further digits. Now \aa@alphinc makes its job and returns with its argument ;#2;. At last the first character ; finishes \number.

```

79  \def\aa@process#1;#2;{%
80    \ifnum#1=#2
81      \expandafter\aa@getresult
82    \else
83      \@ReturnAfterFi{%
84        \expandafter\aa@process\number\aa@inc{#1}\aa@alphinc{;#2;}%
85      }%
86    \fi
87  }
```

#### 2.4.1 Convert the separated digits to the letter result

The single decimal digits of the final letter number are limited by a dot and come in reverse order. The end is marked by an empty digit. The next token is the command to convert a digit (\aa@alph or \aa@Alph), e.g.:

11.3.1..\alph ⇒ ack

\aa@getresult \aa@getresult reads the digits #1 and the converting command #2. Then it calls \aa@@getresult with its arguments.

```

88  \def\aa@getresult#1..#2{%
89    \aa@@getresult!#2#1..%
90  }
```

\aa@@getresult In its first argument #1 \aa@@getresult collects the converted letters in the correct order. Character ! is used as a parameter separator. The next token #2 is the converting command (\aa@alph or \aa@Alph). The next digit #3 is read, converted, and \aa@@getresult is called again. If the digit #3 is empty, the end of the digit form is reached and the process stops and the ready letter number is output.

```

91  \def\aa@@getresult#1!#2#3.{%
92    \ifx\#3\%
93      \@ReturnAfterElseFi{#1}% ready
94    \else
95      \@ReturnAfterFi{%
96        \expandafter\expandafter\expandafter\expandafter
97        \expandafter\expandafter\expandafter
98        \aa@@getresult
99        \expandafter\expandafter\expandafter\expandafter
100      #2{#3}#1!#2%
101    }%
102  \fi
103  }
```

## 2.4.2 Addition by one

Expandable addition of a decimal integer.

`\aa@inc` `\aa@inc` increments its argument #1 by one. The case, that the whole number is less than nine, is specially treated because of speed. (The space after 9 is neccessary.)

```

104 % \aa@inc adds one to its argument #1.
105 \def\aa@inc#1{%
106   \ifnum#1<9
107     \aa@nextdigit{#1}%
108   \else
109     \aa@reverse#1!!%
110   \fi
111 }
```

`\aa@nextdigit` `\aa@nextdigit` increments the digit #1. The result is a digit again. `\aa@addone` works off the case “9+1”.

```

112 \def\aa@nextdigit#1{\ifcase#1 1\or2\or3\or4\or5\or6\or7\or8\or9\fi}
```

`\aa@reverse` Because the addition starts with the lowest significant digit of the number. But with the means of T<sub>E</sub>X’s macro expansion is the first digit of a number available. So `\aa@reverse` reverses the order of the digits and calls `\aa@addone`, if it is ready.

```

113 \def\aa@reverse#1#2!#3!{%
114   \ifx\#2\%
115     \aa@addone#1#3!%
116   \else
117     \@ReturnAfterFi{%
118       \aa@reverse#2!#1#3!%
119     }%
120   \fi
121 }
```

`\aa@addone` The addition is performed by the macro `\aa@addone`. The digits are in reversed order. The parameter text #1#2 separates the next digit #1 that have to be incremented. Already incremented digits are stored in #3 in reversed order to take some work of `\aa@lastreverse`.

```

122 \def\aa@addone#1#2!#3!{%
123   \ifnum#1<9
124     \expandafter\aa@lastreverse\number\aa@nextdigit#1 #2!#3!%
125   \else
126     \@ReturnAfterFi{%
127       \ifx\#2\%
128         10#3%
129       \else
130         \@ReturnAfterFi{%
131           \aa@addone#2!0#3!%
132         }%
133       \fi
134     }%
135   \fi
136 }
```

`\aa@lastreverse` With `\aa@reverse` the order of the digits is changed to perform the addition in `\aa@addone`. Now we have to return to the original order that is done by `\aa@lastreverse`.

```

137 \def\aa@lastreverse#1#2!#3!{%
138   \ifx\#2\%
139     #1#3%
140   \else
141     \@ReturnAfterFi{%
```

```

142      \aa@lastreverse#2!#1#3!%
143    }%
144    \fi
145  }

```

### Increment of the decimal digit result form.

`\aa@alphinc` `\aa@alphinc` adds one to the intermediate number in the decimal digit result form (see 2.4.1). Parameter #1 consists of the tokens that come before the addition result (see ;#2; of `\aa@process`). Then it is also used to store already incremented digits. #2 contains the next digit in the range of 1 until 26. An empty #2 marks the end of the number.

```

146  \def\aa@alphinc#1#2.{%
147    \ifx\#2\%
148      \@ReturnAfterElseFi{%
149        #11..% ready
150      }%
151    \else
152      \@ReturnAfterFi{%
153        \ifnum#2<26
154          \@ReturnAfterElseFi{%
155            \expandafter\aa@alphinc\expandafter
156              {\number\aa@inc{#2}}{#1}%
157          }%
158        \else
159          \@ReturnAfterFi{%
160            \aa@alphinc{#11.}%
161          }%
162        \fi
163      }%
164    \fi
165  }

```

`\aa@alphincast` `\aa@alphincast` is a help macro. Because #2 consists of several tokens (e.g. ;100;), we cannot jump over it via `\expandafter` in `\aa@alphinc`.

```

166  \def\aa@alphincast#1#2{#2#1.}

```

`\newalphalph`

```

167  \newcommand*{\newalphalph}[3]{%
168    \PackageError{alphalph}{%
169      \string\newalphalph\space requires e-TeX%
170    }\@ehc

```

## 2.5 Conversion with $\varepsilon$ -TeX features

```

171 \else

```

`\aa@make` `\aa@make` catches the cases, if the number is zero or negative. Then it expands to nothing like `\romannumeral`.

```

172  \def\aa@make#1#2{%
173    \ifnum#1<1 %
174    \else
175      \@ReturnAfterFi{%
176        \aa@eprocess#1;#2%
177      }%
178    \fi
179  }%

```

`\aa@eprocess` The first argument #1 contains the number that have to be converted yet, the next argument #2 the command for making the conversion of a digit (`\aa@alph` or `\aa@Alph`). The number is divided by 26 to get the rest. Command #2 converts the rest to a letter that is put after the arguments of the next call of `\aa@eprocess`.

The only feature of  $\varepsilon$ -TeX we use the new primitive `\numexpr`. It provides expandable mathematical calculations.

```

180 \def\aa@eprocess#1;#2{%
181   \ifnum#1<27
182     \@ReturnAfterElseFi{%
183       #2{#1}%
184     }%
185   \else
186     \@ReturnAfterFi{%
187       \expandafter\aa@eprocess\number\numexpr(#1-14)/26%
188       \expandafter\expandafter\expandafter;%
189       \expandafter\expandafter\expandafter#2%
190       #2{\numexpr#1-((#1-14)/26)*26}%
191     }%
192   \fi
193 }%
```

## 2.6 Generic version

`\aa@gen@callmake` See macro `\aa@callmake`. Argument #3 holds the number of available symbols.

```

194 \def\aa@gen@callmake#1#2#3{%
195   \expandafter\aa@gen@make\expandafter{\number#3}#1{#2}%
196 }%
```

`\aa@gen@make` See macro `\aa@make`. Argument #3 holds the number of available symbols.

```

197 \def\aa@gen@make#1#2#3{%
198   \ifnum#1<1 %
199   \else
200     \@ReturnAfterFi{%
201       \aa@gen@eprocess{#3}#1;#2%
202     }%
203   \fi
204 }%
```

`\aa@gen@eprocess` See macro `\aa@eprocess`. Argument #1 holds the number of available symbols.

```

205 \def\aa@gen@eprocess#1#2;#3{%
206   \ifnum#2>#1 %
207     \@ReturnAfterElseFi{%
208       \expandafter\aa@gen@eprocess\expandafter{%
209         \number#1\expandafter
210       }%
211       \number\numexpr(#2-(\aa@half{#1}+1))/#1%
212       \expandafter\expandafter\expandafter;%
213       \expandafter\expandafter\expandafter#3%
214       #3{\numexpr#2-((#2-(\aa@half{#1}+1))/#1)*#1\relax}%
215     }%
216   \else
217     \@ReturnAfterFi{%
218       #3{#2}%
219     }%
220   \fi
221 }%
```

`\aa@half` Macro `\aa@half` implements integer division by two without rounding.

```

222 \def\aa@half#1{%
223   \number\dimexpr.5\dimexpr #1sp\relax\relax
224 }%
```

`\newalphalph` New macros are defined by `\newalphalph` that act like `\alphalph`. The macro to be defined is #1. Argument #2 contains the macro that converts a number to a symbol and argument #3 holds the number of available symbols.



```

225 \newcommand*\newalphalph}[3]{%
226   \newcommand*{#1}{}%
227   \edef#1{%
228     \noexpand\aa@gen@callmake\noexpand#2{\number\numexpr#3}%
229   }%
230 }%

```

## 2.7 End of package

Now we can terminate the differentiation between  $\text{T}_{\text{E}}\text{X}$  and  $\varepsilon\text{-T}_{\text{E}}\text{X}$ .

```

231 \fi
    At the end the catcode of the character @ is restored.
232 \catcode'\@=\aa@atcode
233 </package>

```

## 3 Installation

### 3.1 Download

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

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

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

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

[CTAN:macros/latex/contrib/oberdiek/oberdiek-tds.zip](#)

### 3.2 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/

```

### 3.3 Package installation

**Unpacking.** The .dtx file is a self-extracting docstrip archive. The files are extracted by running the .dtx through plain- $\text{T}_{\text{E}}\text{X}$ :

```
tex alphalph.dtx
```

---

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

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

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

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

### 3.4 Refresh file name databases

If your  $\text{\TeX}$  distribution (`te $\text{\TeX}$` , `mik $\text{\TeX}$` , ...) relies on file name databases, you must refresh these. For example, `te $\text{\TeX}$`  users run `texhash` or `mktextlsr`.

### 3.5 Some details for the interested

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

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

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

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

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

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

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

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

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

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

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

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

## 4 History

[1999/03/19 v0.1]

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

---

<sup>2</sup>Url: [http://www.dejanews.com/\[ST\\_rn=ps\]/getdoc.xp?AN=455791936](http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=455791936)

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

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

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

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

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

- Documentation added in dtx format.
- $\varepsilon$ -T<sub>E</sub>X support added.

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

- Minor documentation change.
- First CTAN release.

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

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

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

- Reload check (for plain-T<sub>E</sub>X)
- New DTX framework.
- LPPL 1.3

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

- `\newalphalph` added.

## 5 Index

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

Symbols	
<code>\@</code> .....	42, 43, 232
<code>\@ReturnAfterElseFi</code> .....	..... 44, 93, 148, 154, 182, 207
<code>\@ReturnAfterFi</code> .....	. 44, 74, 83, 95, 117, 126, 130, 141, 152, 159, 175, 186, 200, 217
<code>\@ehc</code> .....	170
<code>\@</code> .....	92, 114, 127, 138, 147
A	
<code>\aa@getresult</code> .....	89, 91
<code>\aa@addone</code> .....	115, 122
<code>\aa@Alph</code> .....	46, 65
<code>\aa@alph</code> .....	46, 64
<code>\aa@alphinc</code> .....	84, 146
<code>\aa@alphinclast</code> .....	155, 166
<code>\aa@atcode</code> .....	232
<code>\aa@callmake</code> .....	64, 65, 66
<code>\aa@eprocess</code> .....	176, 180
<code>\aa@gen@callmake</code> .....	194, 228
<code>\aa@gen@eprocess</code> .....	201, 205
<code>\aa@gen@make</code> .....	195, 197
<code>\aa@getresult</code> .....	81, 88
<code>\aa@half</code> .....	211, 214, 222
<code>\aa@inc</code> .....	84, 104, 156
<code>\aa@lastreverse</code> .....	124, 137
<code>\aa@make</code> .....	67, 71, 172
<code>\aa@nextdigit</code> .....	107, 112, 124
<code>\aa@process</code> .....	75, 79
<code>\aa@reverse</code> .....	109, 113
<code>\AlphAlph</code> .....	2, 64
<code>\alphalph</code> .....	2, 64
C	
<code>\catcode</code> .....	42, 43, 232
<code>\csname</code> .....	3, 13, 26, 39, 42, 70
D	
<code>\dimexpr</code> .....	223
E	
<code>\empty</code> .....	7
<code>\endcsname</code> .....	3, 13, 26, 39, 42, 70
<code>\endinput</code> .....	22
I	
<code>\ifcase</code> .....	4, 47, 56, 112
<code>\ifnum</code> .....	72, 80, 106, 123, 153, 173, 181, 198, 206
<code>\ifx</code> .....	5, 7, 13, 26, 34, 70, 92, 114, 127, 138, 147
<code>\immediate</code> .....	15, 28

N		S	
<code>\newalphalph</code>	2, 167, 225	<code>\space</code>	169
<code>\newcommand</code>	167, 225, 226		
<code>\number</code>	67, 84, 124, 156, 187, 195, 209, 211, 223, 228	T	
<code>\numexpr</code>	187, 190, 211, 214, 228	<code>\the</code>	42
P		W	
<code>\PackageError</code>	168	<code>\write</code>	15, 28
<code>\PackageInfo</code>	18	X	
<code>\ProvidesPackage</code>	40	<code>\x</code>	3, 5, 7, 14, 18, 20, 27, 32, 39