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

1 Usage

The package `alphalph` can be used with both plain-TEX and LATEX:

plain-TEX: `\input alphalph.sty`

LATEX 2ε: `\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 (LaTeX: 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, LaTeX 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, ...
`\alphalph`	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 ε-TeXis 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*{\myvocals}[1]{%
  \ifcase#1\or A\or E\or I\or O\or U\fi
}
\newalphalph{\vocalsvocals}{\myvocals}{5}
```

2 Implementation

2.1 Begin of package

```
1 ⟨*package⟩
```

Reload check, especially if the package is not used with LaTeX.

```
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
```

```
    \fi
  \else
    \expandafter\ifx\csname PackageInfo\endcsname\relax
      \def\x#1#2{%
        \immediate\write-1{Package #1 Info: #2.}%
      }%
    \else
      \def\x#1#2{\PackageInfo{#1}{#2, stopped}}%
    \fi
    \x{alphalph}{The package is already loaded}%
    \endgroup
    \expandafter\endinput
  \fi
\endgroup
```

Package identification:

```
\begingroup
  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
    \def\x#1#2#3[#4]{\endgroup
      \immediate\write-1{Package: #3 #4}%
      \xdef#1{#4}%
    }%
  \else
    \def\x#1#2[#3]{\endgroup
      #2[{#3}]%
      \ifx#1\relax
        \xdef#1{#3}%
      \fi
    }%
  \fi
\expandafter\x\csname ver@alphalph.sty\endcsname
\ProvidesPackage{alphalph}%
  [2006/05/30 v1.4 Converting numbers to letters (HO)]
```

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.

```
\expandafter\edef\csname aa@atcode\endcsname{\the\catcode`\@ }
\catcode`\@=11
```

2.2 Help macros

`\@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 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).

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

`\aa@alph` `\aa@Alph`

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

```
\def\aa@alph#1{%
  \ifcase#1%
    @%
  \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%
  \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%
  \else
    @%
  \fi
}
\def\aa@Alph#1{%
  \ifcase#1%
```

```
    @%
  \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%
  \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%
  \else
    @%
  \fi
}
```

2.3 User commands

`\alphalph` `\AlphAlph`

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

```
\def\alphalph{\aa@callmake\aa@alph}
\def\AlphAlph{\aa@callmake\aa@Alph}
```

`\aa@callmake`

`\aa@callmake` converts the number in the second argument `#2` into explicite decimal digits via the TEX primitive `\number`. (The closing curly brace stops reading the number at the latest.)

```
\def\aa@callmake#1#2{%
  \expandafter\aa@make\expandafter{\number#2}#1%
}
```

ε-TEXprovides 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 ε-TEX extension and a method that is restricted to the standard TEX means.

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

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname numexpr\endcsname\relax
```

2.4 Conversion with standard TEX means

`\aa@make`

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

```
\def\aa@make#1#2{%
  \ifnum#1<1
  \else
    \@ReturnAfterFi{%
      \aa@process1;#1;1..#2%
    }%
  \fi
}
```

`\aa@process`

`\aa@process` contains the algorithm for the conversion. TEXdoesn'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 expandible 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`.

```
\def\aa@process#1;#2;{%
  \ifnum#1=#2
    \expandafter\aa@getresult
  \else
    \@ReturnAfterFi{%
      \expandafter\aa@process\number\aa@inc{#1}\aa@alphinc{;#2;}%
    }%
  \fi
}
```

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.

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

`\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.

```
\def\aa@@getresult#1!#2#3.{%
  \ifx\\#3\\%
    \@ReturnAfterElseFi{#1}% ready
  \else
    \@ReturnAfterFi{%
      \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter
      \aa@@getresult
      \expandafter\expandafter\expandafter\expandafter
      #2{#3}#1!#2%
    }%
  \fi
}
```

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.)

```
% \aa@inc adds one to its argument #1.
  \def\aa@inc#1{%
    \ifnum#1<9
      \aa@nextdigit{#1}%
    \else
      \aa@reverse#1!!%
    \fi
  }
```

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

```
  \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 TEX'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.

```
  \def\aa@reverse#1#2!#3!{%
    \ifx\\#2\\%
      \aa@addone#1#3!!%
    \else
      \@ReturnAfterFi{%
        \aa@reverse#2!#1#3!%
      }%
    \fi
  }
```

\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`.

```
  \def\aa@addone#1#2!#3!{%
    \ifnum#1<9
      \expandafter\aa@lastreverse\number\aa@nextdigit#1 #2!#3!%
    \else
      \@ReturnAfterFi{%
        \ifx\\#2\\%
          10#3%
        \else
          \@ReturnAfterFi{%
            \aa@addone#2!0#3!%
          }%
        \fi
      }%
    \fi
  }
```

\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`.

```
  \def\aa@lastreverse#1#2!#3!{%
    \ifx\\#2\\%
      #1#3%
    \else
      \@ReturnAfterFi{%
        \aa@lastreverse#2!#1#3!%
      }%
    \fi
  }
```

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.

```
\def\aa@alphinc#1#2.{%
  \ifx\\#2\\%
    \@ReturnAfterElseFi{%
      #11..% ready
    }%
  \else
    \@ReturnAfterFi{%
      \ifnum#2<26
        \@ReturnAfterElseFi{%
          \expandafter\aa@alphinclast\expandafter
            {\number\aa@inc{#2}}{#1}%
        }%
      \else
        \@ReturnAfterFi{%
          \aa@alphinc{#11.}%
        }%
      \fi
    }%
  \fi
}
```

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

```
\def\aa@alphinclast#1#2{#2#1.}
```

\newalphalph

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

2.5 Conversion with ε-TEX features

```
\else
```

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

```
\def\aa@make#1#2{%
  \ifnum#1<1 %
  \else
    \@ReturnAfterFi{%
      \aa@eprocess#1;#2%
    }%
  \fi
}%
```

\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 ε-TEX we use the new primitive `\numexpr`. It provides expandible mathematical calculations.

```
\def\aa@eprocess#1;#2{%
  \ifnum#1<27
```

```
      \@ReturnAfterElseFi{%
        #2{#1}%
      }%
    \else
      \@ReturnAfterFi{%
        \expandafter\aa@eprocess\number\numexpr(#1-14)/26%
          \expandafter\expandafter\expandafter;%
          \expandafter\expandafter\expandafter#2%
          #2{\numexpr#1-((#1-14)/26)*26}%
      }%
    \fi
  }%
```

2.6 Generic version

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

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

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

```
  \def\aa@gen@make#1#2#3{%
    \ifnum#1<1 %
    \else
      \@ReturnAfterFi{%
        \aa@gen@eprocess{#3}#1;#2%
      }%
    \fi
  }%
```

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

```
  \def\aa@gen@eprocess#1#2;#3{%
    \ifnum#2>#1 %
      \@ReturnAfterElseFi{%
        \expandafter\aa@gen@eprocess\expandafter{%
          \number#1\expandafter
        }%
        \number\numexpr(#2-(\aa@half{#1}+1))/#1%
        \expandafter\expandafter\expandafter;%
        \expandafter\expandafter\expandafter#3%
        #3{\numexpr#2-((#2-(\aa@half{#1}+1))/#1)*#1\relax}%
      }%
    \else
      \@ReturnAfterFi{%
        #3{#2}%
      }%
    \fi
  }%
```

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

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

`\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.

```
  \newcommand*{\newalphalph}[3]{%
    \newcommand*{#1}{}%
    \edef#1{%
      \noexpand\aa@gen@callmake\noexpand#2{\number\numexpr#3}%
```

```
    }%
}%
```

2.7 End of package

Now we can terminate the differentiation between TEX and ε-TEX.

```
\fi
```

At the end the catcode of the character `@` is restored.

```
\catcode`\@=\aa@atcode
```

⟨/package⟩

3 Installation

CTAN. This package is available on CTAN[1]:

CTAN:macros/latex/contrib/oberdiek/alphalph.dtx The source file.

CTAN:macros/latex/contrib/oberdiek/alphalph.pdf Documentation.

Unpacking. The `.dtx` file is a self-extracting docstrip archive. The files are extracted by running the `.dtx` through plain-TEX:

```
tex alphalph.dtx
```

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

```
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.

Refresh file databases. If your TEX distribution (teTEX, mikTEX, ...) rely on file databases, you must refresh these. For example, teTEX users run `texhash` or `mktexlsr`.

3.1 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.

[1] ftp://ftp.ctan.org/tex-archive/

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 pdfLaTeX:

```
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[2]of Will Douglas[3] and the request[4]of Donald Arsenau[5], 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.

[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.

[2]Url: http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=455791936
[3]Will Douglas's email address: william.douglas@wolfson.ox.ac.uk
[4]Url: http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=456358639
[5]Donald Arsenau's email address: asnd@reg.triumf.ca
[6]Url: http://www.dejanews.com/[ST_rn=ps]/getdoc.xp?AN=456485421

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.