The alphalph package

Heiko Oberdiek
<heiko.oberdiek at googlemail.com>

2010/04/18 v2.3

Abstract

The package provides methods to represent numbers with a limited set of symbols. Both LaTeX and plain TeXare supported.

Contents

1 Documentation

1.1 Introduction

LaTeX counter can be represented in different ways by using presentation commands:

`\arabic`, `\roman`, `\Roman`,
`\alph`, `\Alph`, `\fnsymbol`

The ranges of supported counter values are more or less restricted. Only `\arabic` can be used with any counter value TeX supports.

Presentation command	Supported domain	Ignored values	Error message "Counter too large"
`\arabic`	`-MAX..MAX`		
`\roman`, `\Roman`	`1..MAX`	`-MAX..0`	
`\alph`, `\Alph`	`1..26`	0	`-MAX..-1, 27..MAX`
`\fnsymbol`	`1..9`	0	`-MAX..-1, 10..MAX`

`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 presentated.

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 LaTeX 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

LaTeX'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 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 LaTeX 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 TeX 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 LaTeX 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 TeX and LaTeX:

plain TeX: `\input alphalph.sty`

LaTeX 2ε: `\usepackage{alphalph}`
There aren't any options.

1.5 User commands

`\AlphAlph` {⟨*number*⟩}
`\alphalph` {⟨*number*⟩}

Both macros are number presentation commands that expects a number as argument. LaTeX 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.

⟨*number*⟩	`\AlphAlph{`⟨*number*⟩`}`	`\alphalph{`⟨*number*⟩`}`
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

`\newalphalph` {⟨*cmd*⟩} [⟨*method*⟩] {⟨*symbols provider*⟩} {⟨*number of symbols*⟩}

Macro `\newalphalph` defines ⟨*cmd*⟩ as new number presentation command. Like `\newcommand` an error is thrown, if macro ⟨*cmd*⟩ already exists.

The ⟨*method*⟩ 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. LaTeX'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 catched, 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

`\alphalph@Alph` {⟨*number*⟩}
`\alphalph@alph` {⟨*number*⟩}

They are basically the same as `\@Alph` and `\@alph`. Some languages of package `babel` redefine LaTeX's macros to include some font setup that breaks expandibility. 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 (LaTeX: 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}}`
  - ...
- ε-TEX's numeric expressions are supported, if ε-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.

ε-TEX: ε-TEXis supported, the macros are shorter and faster. But ε-TEX's extensions are not requirements. Without ε-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\catcode61\catcode48\catcode32=10\relax%
3   \catcode13=5 % ^^M
4   \endlinechar=13 %
5   \catcode35=6 % #
6   \catcode39=12 % '
7   \catcode44=12 % ,
8   \catcode45=12 % -
9   \catcode46=12 % .
10  \catcode58=12 % :
11  \catcode64=11 % @
12  \catcode123=1 % {
13  \catcode125=2 % }
14  \expandafter\let\expandafter\x\csname ver@alphalph.sty\endcsname
15  \ifx\x\relax % plain-TeX, first loading
16  \else
17    \def\empty{}%
18    \ifx\x\empty % LaTeX, first loading,
19      % variable is initialized, but \ProvidesPackage not yet seen
20    \else
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      \aftergroup\endinput
30    \fi
```

```
  \fi
\endgroup%
```
Package identification:
```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode35=6 % #
  \catcode39=12 % '
  \catcode40=12 % (
  \catcode41=12 % )
  \catcode44=12 % ,
  \catcode45=12 % -
  \catcode46=12 % .
  \catcode47=12 % /
  \catcode58=12 % :
  \catcode64=11 % @
  \catcode91=12 % [
  \catcode93=12 % ]
  \catcode123=1 % {
  \catcode125=2 % }
  \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\@undefined
        \xdef#1{#3}%
      \fi
      \ifx#1\relax
        \xdef#1{#3}%
      \fi
    }%
  \fi
\expandafter\x\csname ver@alphalph.sty\endcsname
\ProvidesPackage{alphalph}%
  [2010/04/18 v2.3 Converting numbers to letters (HO)]%
```

2.2 Catcodes

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode123=1 % {
  \catcode125=2 % }
  \catcode64=11 % @
  \def\x{\endgroup
    \expandafter\edef\csname AlPh@AtEnd\endcsname{%
      \endlinechar=\the\endlinechar\relax
      \catcode13=\the\catcode13\relax
      \catcode32=\the\catcode32\relax
      \catcode35=\the\catcode35\relax
      \catcode61=\the\catcode61\relax
      \catcode64=\the\catcode64\relax
      \catcode123=\the\catcode123\relax
      \catcode125=\the\catcode125\relax
    }%
  }%
\x\catcode61\catcode48\catcode32=10\relax%
\catcode13=5 % ^^M
\endlinechar=13 %
```

```
\catcode35=6 % #
\catcode64=11 % @
\catcode123=1 % {
\catcode125=2 % }
\def\TMP@EnsureCode#1#2{%
  \edef\AlPh@AtEnd{%
    \AlPh@AtEnd
    \catcode#1=\the\catcode#1\relax
  }%
  \catcode#1=#2\relax
}
\TMP@EnsureCode{33}{12}% !
\TMP@EnsureCode{39}{12}% '
\TMP@EnsureCode{40}{12}% (
\TMP@EnsureCode{41}{12}% )
\TMP@EnsureCode{43}{12}% +
\TMP@EnsureCode{44}{12}% ,
\TMP@EnsureCode{46}{12}% .
\TMP@EnsureCode{47}{12}% /
\TMP@EnsureCode{59}{12}% ;
\TMP@EnsureCode{60}{12}% <
\TMP@EnsureCode{62}{12}% >
\TMP@EnsureCode{91}{12}% [
\TMP@EnsureCode{93}{12}% ]
\TMP@EnsureCode{96}{12}% `
\TMP@EnsureCode{124}{12}% |
\edef\AlPh@AtEnd{\AlPh@AtEnd\noexpand\endinput}
```

2.3 Package loading

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input infwarerr.sty\relax
  \input intcalc.sty\relax
\else
  \RequirePackage{infwarerr}[2007/09/09]%
  \RequirePackage{intcalc}[2007/09/09]%
\fi
```

2.4 ε-TEX detection

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname numexpr\endcsname\relax
  \catcode124=9 % '!': ignore
  \catcode43=14 % '+': comment
\else
  \catcode124=14 % '!': comment
  \catcode43=9   % '+': ignore
\fi
```

2.5 Help macros

\AlPh@Error

```
\def\AlPh@Error#1{%
  \begingroup
    \escapechar=92 % backslash
    \@PackageError{alphalph}{#1}\@ehc
  \endgroup
}
```

\AlPh@IfDefinable

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname @ifdefinable\endcsname\relax
  \def\AlPh@IfDefinable#1#2{%
```

```
    \ifcase\ifx#1\@undefined\else\ifx#1\relax\else1\fi\fi0 %
      #2%
    \else
      \AlPh@Error{%
        Command \string#1 already defined%
      }%
    \fi
  }%
\else
```

\AlPh@IfDefinable

```
  \let\AlPh@IfDefinable\@ifdefinable
\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 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}
```

\@gobblefour

LATEX defines commands for eating arguments. Define \@gobblefour if it is not defined (plain TEX).

```
\expandafter\ifx\csname @gobblefour\endcsname\relax
  \long\def\@gobblefour#1#2#3#4{}%
\fi
```

AlPh@IfOptArg

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname kernel@ifnextchar\endcsname\relax
  \begingroup\expandafter\expandafter\expandafter\endgroup
  \expandafter\ifx\csname @ifnextchar\endcsname\relax
    \def\AlPh@IfOptArg#1#2{%
      \def\AlPh@TempA{#1}%
      \def\AlPh@TempB{#2}%
      \futurelet\AlPh@Token\AlPh@IfOptArgNext
    }%
    \let\AlPh@BracketLeft=[%]
    \def\AlPh@IfOptArgNext{%
      \ifx\AlPh@Token\AlPh@BracketLeft
        \expandafter\AlPh@TempA
      \else
        \expandafter\AlPh@TempB
      \fi
    }%
  \else
    \def\AlPh@IfOptArg{\@ifnextchar[}%]
  \fi
\else
  \def\AlPh@IfOptArg{\kernel@ifnextchar[}%]
\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 LATEX'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
\alphalph@alph

The two commands \AlPh@Alph and \AlPh@alph convert a number into a letter (uppercase and lowercase 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. This error symbol should not appear anywhere (except for bugs).

```
\def\alphalph@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
    \AlPh@ctrerr
    @%
  \fi
}
\def\alphalph@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
    \AlPh@ctrerr
    @%
  \fi
}
```

\AlPh@ctrerr

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

```
\def\AlPh@ctrerr{}
```

2.7 Finding number of symbols

\AlPh@GetNumberOfSymbols

#1: symbols provider

```
\def\AlPh@GetNumberOfSymbols#1{%
  \AlPh@TestNumber1!{#1}%
  \ifAlPh@Unavailable
    \def\AlPh@Number{0}%
    \AlPh@Error{No symbols found}%
  \else
    \def\AlPh@Number{1}%
    \AlPh@ExpSearch2!{#1}%
  \fi
}
```

\ifAlPh@Unavailable

```
\let\ifAlPh@Unavailable\iffalse
\def\AlPh@Unavailabletrue{%
  \global\let\ifAlPh@Unavailable\iftrue
}
\def\AlPh@Unavailablefalse{%
  \global\let\ifAlPh@Unavailable\iffalse
}
```

\AlPh@TestNumber

#1: number to be tested
#2: symbols provider

```
\def\AlPh@TestNumber#1!#2{%
  \AlPh@Unavailablefalse
  \begingroup
```

```
    \setbox0=\hbox{%
      \begingroup % color
        \let\@ctrerr\AlPh@Unavailabletrue
        \let\AlPh@ctrerr\AlPh@Unavailabletrue
        #2{#1}%
      \endgroup
    }%
    \ifdim\wd0=0pt %
      \AlPh@Unavailabletrue
    \fi
  \endgroup
}
```

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

```
\def\AlPh@ExpSearch#1!#2{%
  \let\AlPh@Next\relax
  \AlPh@TestNumber#1!{#2}%
  \ifAlPh@Unavailable
    \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
  \else
    \def\AlPh@Number{#1}%
    \ifnum#1>1073741823 %
      \AlPh@TestNumber2147483647!{#2}%
      \ifAlPh@Unavailable
        \AlPh@BinSearch#1!2147483647!{#2}%
      \else
        \def\AlPh@Number{0}%
        \AlPh@Error{%
          Maximal symbol number not found%
        }%
      \fi
    \else
      \def\AlPh@Next{%
        \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}!{#2}%
      }%
    \fi
  \fi
  \AlPh@Next
}
```

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

```
\def\AlPh@BinSearch#1!#2!#3{%
  \expandafter\AlPh@ProcessBinSearch
  \number\intcalcShr{\intcalcAdd{#1}{#2}}!%
  #1!#2!{#3}%
}
```

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

```
\def\AlPh@ProcessBinSearch#1!#2!#3!#4{%
  \let\AlPh@Next\relax
  \ifnum#1>#2 %
    \ifnum#1<#3 %
      \AlPh@TestNumber#1!{#4}%
      \ifAlPh@Unavailable
        \def\AlPh@Next{%
          \AlPh@BinSearch#2!#1!{#4}%
```

```
        }%
      \else
        \def\AlPh@Next{%
          \AlPh@BinSearch#1!#3!{#4}%
        }%
      \fi
    \else
      \def\AlPh@Number{#2}%
    \fi
  \else
    \def\AlPh@Number{#2}%
  \fi
  \AlPh@Next
}
```

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)

```
\def\AlPh@CheckPositive#1!#2{%
  \ifnum#1<1 %
    \expandafter\@gobblefour
  \fi
  #2{#1}%
}
```

2.8.2 Method 'alph'

`\AlPh@Method@alph`

#1: number of symbols
#2: symbols provider
#3: number to be converted

```
\def\AlPh@Method@alph#1#2#3{%
  \expandafter\AlPh@CheckPositive
|   \number#3!%
+   \the\numexpr#3!%
    \AlPh@ProcessAlph
    {#1}{#2}%
}
```

`\AlPh@ProcessAlph`

#1: current number
#2: number of symbols
#3: symbols provider

```
\def\AlPh@ProcessAlph#1#2#3{%
  \ifnum#1>#2 %
    \@ReturnAfterElseFi{%
      \expandafter\AlPh@StepAlph\number
        \intcalcInc{%
          \intcalcMod{\intcalcDec{#1}}{#2}%
        }%
      \expandafter!\number
        \intcalcDiv{\intcalcDec{#1}}{#2}%
      !{#2}{#3}%
```

```
    }%
  \else
    \@ReturnAfterFi{%
      #3{#1}%
    }%
  \fi
}
```

`\AlPh@StepAlph` #1: current last digit
#2: new current number
#3: number of symbols
#4: symbols provider

```
\def\AlPh@StepAlph#1!#2!#3#4{%
  \AlPh@ProcessAlph{#2}{#3}{#4}%
  #4{#1}%
}
```

2.8.3 Method 'wrap'

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

```
\def\AlPh@Method@wrap#1#2#3{%
  \expandafter\AlPh@CheckPositive
|   \number#3!%
+   \the\numexpr#3!%
    \AlPh@ProcessWrap
    {#1}{#2}%
}
```

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

```
\def\AlPh@ProcessWrap#1#2#3{%
  \ifnum#1>#2 %
    \@ReturnAfterElseFi{%
      \expandafter\AlPh@StepWrap\number
        \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
      !{#3}%
    }%
  \else
    \@ReturnAfterFi{%
      #3{#1}%
    }%
  \fi
}
```

`\AlPh@StepWrap` #1: final number
#2: symbols provider

```
\def\AlPh@StepWrap#1!#2{%
  #2{#1}%
}
```

2.8.4 Method 'mult'

After the number of symbols is exhausted, repetitions of the symbol are used.

$$x := \text{number to be converted}$$
$$n := \text{number of symbols}$$
$$r := \text{repetition length}$$
$$s := \text{symbol slot}$$
$$r = ((x-1) \div n) + 1$$
$$s = ((x-1) \mod n) + 1$$

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

```
\def\AlPh@Method@mult#1#2#3{%
  \expandafter\AlPh@CheckPositive
|   \number#3!%
+   \the\numexpr#3!%
    \AlPh@ProcessMult
    {#1}{#2}%
}
```

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

```
\def\AlPh@ProcessMult#1#2#3{%
  \ifnum#1>#2 %
    \@ReturnAfterElseFi{%
      \expandafter\AlPh@StepMult\romannumeral
        \intcalcInc{\intcalcDiv{\intcalcDec{#1}}{#2}}%
        000%
      \expandafter!\number
        \intcalcInc{\intcalcMod{\intcalcDec{#1}}{#2}}%
      !{#3}%
    }%
  \else
    \@ReturnAfterFi{%
      #3{#1}%
    }%
  \fi
}
```

`\AlPh@StepMult` #1#2: repetitions coded as list of character 'm'
#3: symbol slot
#4: symbols provider

```
\def\AlPh@StepMult#1#2!#3!#4{%
  \ifx\\#2\\%
  \else
    \@ReturnAfterFi{%
      \AlPh@StepMult#2!#3!{#4}%
    }%
  \fi
  #4{#3}%
}
```

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

```
\AlPh@IfDefinable\newalphalph{%
  \def\newalphalph#1{%
    \AlPh@IfOptArg{%
      \AlPh@newalphalph{#1}%
    }{%
      \AlPh@newalphalph{#1}[alph]%
    }%
  }%
}
```

`\AlPh@newalphalph` #1: cmd #2: method name
#3: symbols provider
#4: number of symbols

```
\def\AlPh@newalphalph#1[#2]#3#4{%
  \begingroup\expandafter\expandafter\expandafter\endgroup
  \expandafter\ifx\csname AlPh@Method@#2\endcsname\relax
    \AlPh@Error{%
      Unknown method %
|     `#2'%
+     `\detokenize{#2}'%
    }%
  \else
    \ifx\\#4\\%
      \AlPh@GetNumberOfSymbols{#3}%
      \ifcase\AlPh@Number
      \else
        \begingroup
          \escapechar=92 % backslash
          \@PackageInfo{alphalph}{%
            Number of symbols for \string#1 is \AlPh@Number
          }%
        \endgroup
        \expandafter\AlPh@NewAlphAlph
        \csname AlPh@Method@#2\expandafter\endcsname
        \AlPh@Number!{#1}{#3}%
      \fi
    \else
      \expandafter\AlPh@NewAlphAlph
      \csname AlPh@Method@#2\expandafter\endcsname
|     \number#4!%
+     \the\numexpr#4!%
      {#1}{#3}%
    \fi
  \fi
}%
```

`\AlPh@NewAlphAlph` #1: method macro
#2: number of symbols
#3: cmd
#4: symbols provider

```
\def\AlPh@NewAlphAlph#1#2!#3#4{%
  \AlPh@IfDefinable#3{%
    \ifnum#2>0 %
      \def#3{#1{#2}{#4}}%
    \else
      \AlPh@Error{%
          Definition of \string#3 failed,\MessageBreak
          because number of symbols (#2) is not positive%
        }%
```

```
    \fi
  }%
}
```

\AlphAlph

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

\alphalph

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

\AlPh@AtEnd%
```
⟨/package⟩

3 Test

3.1 Catcode checks for loading

⟨*test1⟩

```
\catcode`\{=1 %
\catcode`\}=2 %
\catcode`\#=6 %
\catcode`\@=11 %
\expandafter\ifx\csname count@\endcsname\relax
  \countdef\count@=255 %
\fi
\expandafter\ifx\csname @gobble\endcsname\relax
  \long\def\@gobble#1{}%
\fi
\expandafter\ifx\csname @firstofone\endcsname\relax
  \long\def\@firstofone#1{#1}%
\fi
\expandafter\ifx\csname loop\endcsname\relax
  \expandafter\@firstofone
\else
  \expandafter\@gobble
\fi
{%
  \def\loop#1\repeat{%
    \def\body{#1}%
    \iterate
  }%
  \def\iterate{%
    \body
      \let\next\iterate
    \else
      \let\next\relax
    \fi
    \next
  }%
  \let\repeat=\fi
}%
\def\RestoreCatcodes{}
\count@=0 %
\loop
  \edef\RestoreCatcodes{%
    \RestoreCatcodes
    \catcode\the\count@=\the\catcode\count@\relax
  }%
\ifnum\count@<255 %
  \advance\count@ 1 %
\repeat
```

```
\def\RangeCatcodeInvalid#1#2{%
  \count@=#1\relax
  \loop
    \catcode\count@=15 %
  \ifnum\count@<#2\relax
    \advance\count@ 1 %
  \repeat
}
\def\RangeCatcodeCheck#1#2#3{%
  \count@=#1\relax
  \loop
    \ifnum#3=\catcode\count@
    \else
      \errmessage{%
        Character \the\count@\space
        with wrong catcode \the\catcode\count@\space
        instead of \number#3%
      }%
    \fi
  \ifnum\count@<#2\relax
    \advance\count@ 1 %
  \repeat
}
\def\space{ }
\expandafter\ifx\csname LoadCommand\endcsname\relax
  \def\LoadCommand{\input alphalph.sty\relax}%
\fi
\def\Test{%
  \RangeCatcodeInvalid{0}{47}%
  \RangeCatcodeInvalid{58}{64}%
  \RangeCatcodeInvalid{91}{96}%
  \RangeCatcodeInvalid{123}{255}%
  \catcode`\@=12 %
  \catcode`\\=0 %
  \catcode`\%=14 %
  \LoadCommand
  \RangeCatcodeCheck{0}{36}{15}%
  \RangeCatcodeCheck{37}{37}{14}%
  \RangeCatcodeCheck{38}{47}{15}%
  \RangeCatcodeCheck{48}{57}{12}%
  \RangeCatcodeCheck{58}{63}{15}%
  \RangeCatcodeCheck{64}{64}{12}%
  \RangeCatcodeCheck{65}{90}{11}%
  \RangeCatcodeCheck{91}{91}{15}%
  \RangeCatcodeCheck{92}{92}{0}%
  \RangeCatcodeCheck{93}{96}{15}%
  \RangeCatcodeCheck{97}{122}{11}%
  \RangeCatcodeCheck{123}{255}{15}%
  \RestoreCatcodes
}
\Test
\csname @@end\endcsname
\end
```

⟨/test1⟩

4 Macro tests

```
⟨*test2⟩
\NeedsTeXFormat{LaTeX2e}
\nofiles
\documentclass{article}
```

```
⟨*noetex⟩
\makeatletter
\let\saved@numexpr\numexpr
\newcommand*{\DisableNumexpr}{%
  \let\numexpr\@undefined
}
\newcommand*{\RestoreNumexpr}{%
  \let\numexpr\saved@numexpr
}
\DisableNumexpr
⟨/noetex⟩
\usepackage{alphalph}[2010/04/18]
⟨noetex⟩\RestoreNumexpr
\usepackage{qstest}
\IncludeTests{*}
\LogTests{log}{*}{*}

\newcommand*{\TestCmd}[3]{%
  \setbox0=\hbox{%
⟨noetex⟩    \DisableNumexpr
    \edef\TestString{#1{#2}}%
    \expandafter\Expect\expandafter{\TestString}{#3}%
    \edef\TestString{#1{#2} }%
    \expandafter\Expect\expandafter{\TestString}{#3 }%
  }%
  \Expect*{\the\wd0}{0.0pt}%
}

\makeatletter
\newalphalph\LaTeXAlphAlph\@Alph{26}
\newalphalph\LaTeXalphalph\@alph{26}
\newalphalph\AlphWrap[wrap]\alphalph@Alph{26}
\newalphalph\alphwrap[wrap]\alphalph@alph{26}
\newalphalph\LaTeXAlphWrap[wrap]\@Alph{26}
\newalphalph\LaTeXalphwrap[wrap]\@alph{26}
\def\LastSymbol#1{%
  \ifx\\#1\\%
  \else
    \@LastSymbol#1\@nil
  \fi
}
\def\@LastSymbol#1#2\@nil{%
  \ifx\\#2\\%
    #1%
  \else
    \@LastSymbol#2\@nil
  \fi
}
\makeatother
\newcommand*{\TestAlph}[2]{%
  \uppercase{\TestCallCmd\AlphAlph{#2}}{#1}%
  \lowercase{\TestCallCmd\alphalph{#2}}{#1}%
  \uppercase{\TestCallCmd\LaTeXAlphAlph{#2}}{#1}%
  \lowercase{\TestCallCmd\LaTeXalphalph{#2}}{#1}%
  \edef\WrapString{\LastSymbol{#2}}%
  \expandafter\TestAlphWrap\expandafter{\WrapString}{#1}%
}
\newcommand*{\TestAlphWrap}[2]{%
  \uppercase{\TestCallCmd\AlphWrap{#1}}{#2}%
  \lowercase{\TestCallCmd\alphwrap{#1}}{#2}%
  \uppercase{\TestCallCmd\LaTeXAlphWrap{#1}}{#2}%
  \lowercase{\TestCallCmd\LaTeXalphwrap{#1}}{#2}%
```

```
}
\newcommand*{\TestCallCmd}[3]{%
  \TestCmd#1{#3}{#2}%
}
\begin{qstest}{AlphSymbols}{alphalph, AlphAlph, symbols}
  \TestAlph{1}{a}%
  \TestAlph{2}{b}%
  \TestAlph{3}{c}%
  \TestAlph{4}{d}%
  \TestAlph{5}{e}%
  \TestAlph{6}{f}%
  \TestAlph{7}{g}%
  \TestAlph{8}{h}%
  \TestAlph{9}{i}%
  \TestAlph{10}{j}%
  \TestAlph{11}{k}%
  \TestAlph{12}{l}%
  \TestAlph{13}{m}%
  \TestAlph{14}{n}%
  \TestAlph{15}{o}%
  \TestAlph{16}{p}%
  \TestAlph{17}{q}%
  \TestAlph{18}{r}%
  \TestAlph{19}{s}%
  \TestAlph{20}{t}%
  \TestAlph{21}{u}%
  \TestAlph{22}{v}%
  \TestAlph{23}{w}%
  \TestAlph{24}{x}%
  \TestAlph{25}{y}%
  \TestAlph{26}{z}%
\end{qstest}
\begin{qstest}{AlphRange}{alphalph, range}
  \TestAlph{0}{}%
  \TestAlph{-1}{}%
  \TestAlph{-2147483647}{}%
  \TestAlph{27}{aa}%
  \TestAlph{28}{ab}%
  \TestAlph{52}{az}%
  \TestAlph{53}{ba}%
  \TestAlph{78}{bz}%
  \TestAlph{79}{ca}%
  \TestAlph{702}{zz}%
  \TestAlph{703}{aaa}%
  \TestAlph{2147483647}{fxshrxw}%
\end{qstest}

\makeatletter
\newcommand*{\myvocals}[1]{%
  \ifcase#1X\or A\or E\or I\or O\or U\else Y\fi
}
\makeatother
\newalphalph\vocalsvocals\myvocals{5}
\newcommand*{\TestVocals}{%
  \TestCmd\vocalsvocals
}
\begin{qstest}{vocals}{vocals}
  \TestVocals{0}{}%
  \TestVocals{1}{A}%
  \TestVocals{2}{E}%
  \TestVocals{3}{I}%
  \TestVocals{4}{O}%
```

```
  \TestVocals{5}{U}%
  \TestVocals{6}{AA}%
  \TestVocals{7}{AE}%
  \TestVocals{8}{AI}%
  \TestVocals{9}{AO}%
  \TestVocals{10}{AU}%
  \TestVocals{11}{EA}%
  \TestVocals{24}{OO}%
  \TestVocals{25}{OU}%
  \TestVocals{26}{UA}%
  \TestVocals{29}{UO}%
  \TestVocals{30}{UU}%
  \TestVocals{31}{AAA}%
  \TestVocals{155}{UUU}%
  \TestVocals{156}{AAAA}%
  \TestVocals{2147483647}{AIIOOEEIOIIUOE}%
\end{qstest}

\makeatletter
\newalphalph\AlphMult[mult]{\alphalph@Alph}{26}
\newalphalph\alphmult[mult]{\alphalph@alph}{26}
\newalphalph\LaTeXAlphMult[mult]{\@Alph}{26}
\newalphalph\LaTeXalphmult[mult]{\@alph}{26}
\makeatother
\newcommand*{\TestMult}[2]{%
  \uppercase{\TestCallCmd\AlphMult{#2}}{#1}%
  \lowercase{\TestCallCmd\alphmult{#2}}{#1}%
  \uppercase{\TestCallCmd\LaTeXAlphMult{#2}}{#1}%
  \lowercase{\TestCallCmd\LaTeXalphmult{#2}}{#1}%
}
\begin{qstest}{mult}{mult}
  \TestMult{0}{}%
  \TestMult{-1}{}%
  \TestMult{-2147483647}{}%
  \TestMult{1}{a}%
  \TestMult{2}{b}%
  \TestMult{26}{z}%
  \TestMult{27}{aa}%
  \TestMult{28}{bb}%
  \TestMult{52}{zz}%
  \TestMult{53}{aaa}%
  \TestMult{54}{bbb}%
  \TestMult{259}{yyyyyyyyy}%
  \TestMult{260}{zzzzzzzzzz}%
  \TestMult{261}{aaaaaaaaaa}%
  \TestMult{262}{bbbbbbbbbb}%
\end{qstest}

\def\myvocalsB#1{%
  \ifcase#1\or A\or E\or I\or O\or U\fi
}
\begin{qstest}{symbolnum}{symbolnum}
  \makeatletter
  \def\Test#1#2{%
    \let\TestCmd\relax
    \newalphalph\TestCmd{#1}{}%
    \Expect*{\AlPh@Number}{#2}%
  }%
  \Test\@alph{26}%
  \Test\@Alph{26}%
  \Test\@fnsymbol{9}%
  \Test\myvocalsB{5}%
```

```
  \Test\alphalph@alph{26}%
  \Test\alphalph@Alph{26}%
\end{qstest}

\begin{qstest}{list}{list}
  \makeatletter
  \def\catch#1\relax{%
    \def\FoundList{\catch#1}%
  }%
  \def\Test[#1]#2#3#4{%
    \let\testcmd\relax
    \newalphalph\testcmd[{#1}]{\catch}{#2}%
    \testcmd{#3}|\relax
    \expandafter\Expect\expandafter{\FoundList}{#4|}%
    %
    \let\SavedCatch\catch
    \def\catch{\noexpand\catch\noexpand\foo}%
    \edef\Result{#4|}%
    \@onelevel@sanitize\Result
    \let\catch\SavedCatch
    \let\testcmd\relax
    \newalphalph\testcmd[{#1}]{\catch\foo}{#2}%
    \testcmd{#3}|\relax
    \@onelevel@sanitize\FoundList
    \Expect*{\FoundList}*{\Result}%
  }%
  \Test[alph]{26}{3}{\catch{3}}%
  \Test[alph]{26}{12}{\catch{12}}%
  \Test[alph]{26}{27}{\catch{1}\catch{1}}%
  \Test[alph]{26}{78}{\catch{2}\catch{26}}%
  \Test[wrap]{26}{7}{\catch{7}}%
  \Test[wrap]{26}{14}{\catch{14}}%
  \Test[wrap]{26}{80}{\catch{2}}%
  \Test[wrap]{26}{700}{\catch{24}}%
  \Test[mult]{26}{4}{\catch{4}}%
  \Test[mult]{26}{17}{\catch{17}}%
  \Test[mult]{26}{54}{\catch{2}\catch{2}\catch{2}}%
\end{qstest}

\begin{document}
\end{document}
⟨/test2⟩
```

5 Installation

5.1 Download

Package. 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.

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

CTAN:install/macros/latex/contrib/oberdiek.tds.zip

TDS refers to the standard "A Directory Structure for TEX Files" (CTAN:tds/tds.pdf). Directories with `texmf` in their name are usually organized this way.

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

5.2 Bundle installation

Unpacking. Unpack the `oberdiek.tds.zip` in the TDS tree (also known as `texmf` tree) of your choice. Example (linux):

```
unzip oberdiek.tds.zip -d ~/texmf
```

Script installation. Check the directory `TDS:scripts/oberdiek/` for scripts that need further installation steps. Package attachfile2 comes with the Perl script `pdfatfi.pl` that should be installed in such a way that it can be called as `pdfatfi`. Example (linux):

```
chmod +x scripts/oberdiek/pdfatfi.pl
cp scripts/oberdiek/pdfatfi.pl /usr/local/bin/
```

5.3 Package installation

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

```
tex alphalph.dtx
```

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

```
alphalph.sty            → tex/generic/oberdiek/alphalph.sty
alphalph.pdf            → doc/latex/oberdiek/alphalph.pdf
test/alphalph-test1.tex → doc/latex/oberdiek/test/alphalph-test1.tex
test/alphalph-test2.tex → doc/latex/oberdiek/test/alphalph-test2.tex
test/alphalph-test3.tex → doc/latex/oberdiek/test/alphalph-test3.tex
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.

5.4 Refresh file name databases

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

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 pdfLaTeX:

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

6 History

[1999/03/19 v0.1]

- The first version was built as a response to a question[2]of Will 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.

[2007/04/11 v1.5]

- Line ends sanitized.

[2]Url: http://groups.google.com/group/comp.text.tex/msg/17a74cd721641038
[3]Will Douglas's email address: william.douglas@wolfson.ox.ac.uk
[4]Url: http://groups.google.com/group/comp.text.tex/msg/8f9768825640315f
[5]Donald Arsenau's email address: asnd@reg.triumf.ca
[6]Url: http://groups.google.com/group/comp.text.tex/msg/cec563eef8bf65d0

[2007/09/09 v2.0]

- New implementation that uses package `\intcalc`. This removes the dependency on ε-TEX.
- `\newalphalph` is extended to support new methods 'wrap' and 'multi'.
- Documentation rewritten.

[2008/08/11 v2.1]

- Code is not changed.
- URLs updated from `www.dejanews.com` to `groups.google.com`.

[2010/03/01 v2.2]

- Compatibility with ini-TEX.

[2010/04/18 v2.3]

- Documentation fixes (Martin Münch).

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; plain numbers refer to the code lines where the entry is used.

S

T

U

V

W

X