The \texttt{alphalph} package

Heiko Oberdiek\footnote{Please report any issues at https://github.com/ho-tex/oberdiek/issues}

<heiko.oberdiek at googlemail.com>

2016/05/16 v2.5

Abstract

The package provides methods to represent numbers with a limited set of symbols. Both \LaTeX{} and plain \TeX{} are supported.

Contents

1 Documentation \hspace{1cm} 2
  1.1 Introduction \hspace{1cm} 2
  1.2 Use cases \hspace{1cm} 3
    1.2.1 Number system based on symbols \hspace{1cm} 3
    1.2.2 Wrap symbols around \hspace{1cm} 3
    1.2.3 Multiple symbols \hspace{1cm} 3
  1.3 Glossary \hspace{1cm} 4
  1.4 Package usage \hspace{1cm} 4
  1.5 User commands \hspace{1cm} 5
  1.6 Programmer commands \hspace{1cm} 5
  1.7 Design principles \hspace{1cm} 6
    1.7.1 Number presentation commands \hspace{1cm} 6
    1.7.2 General usability \hspace{1cm} 6

2 Implementation \hspace{1cm} 6
  2.1 Begin of package \hspace{1cm} 6
  2.2 Catcodes \hspace{1cm} 7
  2.3 Package loading \hspace{1cm} 8
  2.4 \texttt{\varepsilon\TeX} detection \hspace{1cm} 8
  2.5 Help macros \hspace{1cm} 9
  2.6 Symbol provider \hspace{1cm} 10
    2.6.1 Alphabet \hspace{1cm} 10
  2.7 Finding number of symbols \hspace{1cm} 10
  2.8 Methods \hspace{1cm} 12
    2.8.1 Common methods \hspace{1cm} 12
    2.8.2 Method \texttt{`alph`} \hspace{1cm} 12
    2.8.3 Method \texttt{`wrap`} \hspace{1cm} 13
    2.8.4 Method \texttt{`mult`} \hspace{1cm} 14
  2.9 User interface \hspace{1cm} 15

3 Test \hspace{1cm} 16
  3.1 Catcode checks for loading \hspace{1cm} 16

4 Macro tests \hspace{1cm} 18
1 Documentation

1.1 Introduction

\LaTeX{} counters can be represented in different ways by using presentation commands:

\begin{verbatim}
\arabic, \roman, \Roman,
\alph, \Alph, \fnsymbol
\end{verbatim}

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

<table>
<thead>
<tr>
<th>Presentation command</th>
<th>Supported values</th>
<th>Ignored values</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>\arabic</td>
<td>-MAX..MAX</td>
<td>-MAX..0</td>
<td>“Counter too large”</td>
</tr>
<tr>
<td>\roman, \Roman</td>
<td>1..MAX</td>
<td>-MAX..0</td>
<td></td>
</tr>
<tr>
<td>\alph, \Alph</td>
<td>1..26</td>
<td>0</td>
<td>-MAX..-1, 27..MAX</td>
</tr>
<tr>
<td>\fnsymbol</td>
<td>1..9</td>
<td>0</td>
<td>-MAX..-1, 10..MAX</td>
</tr>
</tbody>
</table>

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 alphas 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

Assume 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 use 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}{\@fnsymbol}{\fnsymbolwrap}{\value{footnote}}%
\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
\renewcommand*{\labelfootnotemark}{\fnsymbolsingle{\value{footnote}}}%
\fnsymbolsingle generates: * (1), † (2), ‡ (3), ..., ‡‡ (9), * (10), † 11, ...

3
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 2e: \usepackage{alphalph}

There aren’t any options.
1.5 User commands

Both macros are number presentation commands that expects a number as argument. \LaTeX counters are used with \texttt{\textbackslash 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 \texttt{\textbackslash AlphAlph} uses uppercase letters, \texttt{\textbackslash alphalph} generates the lowercase variant.

\begin{verbatim}
\begin{tabular}{c|c|c}
  \texttt{(number)} & \texttt{\textbackslash AlphAlph{(number)}} & \texttt{\textbackslash alphalph{(number)}} \\
  \hline
  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 \\
\end{tabular}
\end{verbatim}

Macro \texttt{\textbackslash newalphalph} defines \texttt{(cmd)} as new number presentation command. Like \texttt{\newcommand} an error is thrown, if macro \texttt{(cmd)} already exists.

The \texttt{(method)} is one of \texttt{alph}, \texttt{wrap}, or \texttt{mult}. The default is \texttt{alph}.

As symbol provider a number presentation command can be used, e.g. \texttt{\textbackslash fn-symbol}, \texttt{\textbackslash Alph}, or \texttt{\alphalph@alph}.

The last argument is the number of symbols. If the argument is empty, then \texttt{\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 \texttt{\textbackslash @ctrerr}. Thus \texttt{\newalphalph} calls the symbol provider and tests a number by typesetting it in a temporary box. The error macro \texttt{\textbackslash @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

They are basically the same as \texttt{\textbackslash Alph} and \texttt{\textbackslash alph}. Some languages of package \texttt{babel} redefine \LaTeX’s macros to include some font setup that breaks expandibility. Therefore \texttt{\textbackslash AlphAlph} and \texttt{\textbackslash alphalph} are based on \texttt{\alphalph@Alph} and \texttt{\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 \texttt{\csname -\endcsname} pair.

- If the argument is zero or negative, the commands expand to nothing like \texttt{\romannumeral}.

- The argument is a \TeX number. Anything that would be accepted by \texttt{\num} is a valid argument:
  - explicite constants,
  - macros that expand to a number,
  - count registers, \LaTeX counter can used via \texttt{\value}, e. g.:
  \begin{verbatim}
\alphalph{\value{page}}
\end{verbatim}
  - ...

- \e-\TeX's numeric expressions are supported, if \e-\TeX is available. Then \texttt{\numexpr} is applied to the argument. Package \texttt{\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.

\e-\TeX: \e-\TeX is supported, the macros are shorter and faster. But \e-\TeX's extensions are not requirements. Without \e-\TeX, just the implementation changes. The properties remain unchanged.

2 Implementation

2.1 Begin of package

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

\begin{verbatim}
\begin{group}
code13=5 \relax\% ~\~M
\endcodechar=13 \%
\catcode35=6 \%
\catcode39=12 \%
\catcode44=12 \%
\catcode45=12 \%
\catcode46=12 \%
\catcode58=12 \%
\catcode64=11 \%
\catcode123=1 \%
\catcode125=2 \%
\expandafter\let\expandafter\x\csname ver@alphalph.sty\endcsname
\ifx\x\relax \else\def\empty{}\fi
\endgroup
\end{verbatim}

\begin{verbatim}
\begin{group}
\echoafter{\let\expandafter\x\csname ver@alphalph.sty\endcsname}
\ifx\x\relax \else \relax \fi
\endgroup
\end{verbatim}

plain-\TeX, first loading

\begin{verbatim}
\def\empty{\}
\end{verbatim}

6
\ifx\empty % LaTeX, first loading,
% variable is initialized, but \ProvidesPackage not yet seen
\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}%
\aftergroup\endinput
\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 % /
\catcode48=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\ifx\csname ver@alphalph.sty\endcsname
\ProvidesPackage{alphalph}[
2016/05/16 v2.5 Convert numbers to letters (HO)]%
\else
\begin{appendix}

\section{Catcodes}
\end{appendix}
\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
}\%
} %
\catcode61\catcode48\catcode32=10\relax\%
\catcode13=5 \^^M
\catcode6=6 \#
\catcode14=1 \{
\catcode15=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{49}{12}% 
\TMP@EnsureCode{59}{12}% <
\TMP@EnsureCode{60}{12}% >
\TMP@EnsureCode{62}{12}% [
\TMP@EnsureCode{93}{12}% ]
\TMP@EnsureCode{96}{12}%`
\TMP@EnsureCode{124}{12}% |
\edef\AlPh@AtEnd{\AlPh@AtEnd
\nobreak\endinput}

\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

\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

2.3 Package loading

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

2.4 $\varepsilon$-T\TeX{} detection

\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname require\endcsname\relax
\catcode124=9 % '!': ignore
\catcode43=14 % '+': comment
\else
\catcode124=14 % '!': comment
\catcode43=9 % '+': ignore
\fi
2.5 Help macros

\AlPh@Error
133 \def\AlPh@Error#1{%
134 \begingroup
135 \escapechar=92 % backslash
136 \@PackageError{alphalph}{#1}\@ehc
137 \endgroup
138 }

\AlPh@IfDefinable
139 \begingroup\expandafter\expandafter\expandafter\endgroup
140 \expandafter\ifx\csname \ifdefinable\endsname\relax
141 \def\AlPh@IfDefinable#1#2{%
142 \ifcase\ifx#1\@undefined\else\ifx#1\relax\else1\fi\fi0 %
143 #2%
144 \else
145 \AlPh@Error{%
146 Command \string#1 already defined%
147 }%
148 \fi
149 }%
150 \else

\AlPh@IfDefinable
151 \let\AlPh@IfDefinable\ifdefinable
152 \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.)
153 \long\def\@ReturnAfterElseFi#1\else#2\fi{\fi#1}
154 \long\def\@ReturnAfterFi#1\fi{\fi#1}

@gobblefour \TPeX{} defines commands for eating arguments. Define \@gobblefour if it is not
defined (plain \TeX{}).
155 \expandafter\ifx\csname \gobblefour\endsname\relax
156 \long\def\gobblefour#1#2#3#4{}%
157 \fi

AlPh@IfOptArg
158 \begingroup\expandafter\expandafter\expandafter\endgroup
159 \expandafter\ifx\csname kernel@ifnextchar\endsname\relax
160 \begingroup\expandafter\expandafter\expandafter\endgroup
161 \expandafter\ifx\csname ifnextchar\endsname\relax
162 \def\AlPh@IfOptArg#1#2{%
163 \def\AlPh@TempA{#1}%
164 \def\AlPh@TempB{#2}%
165 \futurelet\AlPh@Token\AlPh@IfOptArgNext
166 }%
167 \let\AlPh@BracketLeft=[[
168 \def\AlPh@IfOptArgNext{%
169 \ifx\AlPh@Token\AlPh@BracketLeft
170 \expandafter\AlPh@TempA
171 \else
172 \expandafter\AlPh@TempB
173 \fi
174 }%
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 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).

\AlPh@ctrerr

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

\AlPh@GetNumberOfSymbols

2.7 Finding number of symbols

\AlPh@GetNumberOfSymbols

\AlPh@GetNumberOfSymbols#1: symbols provider
\ifAlPh@Unavailable
212 \let\ifAlPh@Unavailable\iffalse
213 \def\AlPh@Unavailabletrue{%
214 \global\let\ifAlPh@Unavailable\iftrue
215 }
216 \def\AlPh@Unavailablefalse{%
217 \global\let\ifAlPh@Unavailable\iffalse
218 }
\AlPh@TestNumber
\#1: number to be tested
\#2: symbols provider
219 \def\AlPh@TestNumber#1!#2{%
220 \AlPh@Unavailablefalse
221 \begingroup
222 \setbox0=\hbox{%
223 \let\@ctrerr\AlPh@Unavailabletrue
224 \let\AlPh@ctrerr\AlPh@Unavailabletrue
225 \let\AlPh@ctrerr\AlPh@Unavailabletrue
226 \#2{#1}%
227 \endgroup
228 }%
229 \ifdim\wd0=0pt %
230 \AlPh@Unavailabletrue
231 \fi
232 \endgroup
233 }
\AlPh@ExpSearch
\#1: number to be tested
\#2: symbols provider
234 \def\AlPh@ExpSearch#1!#2{%
235 \let\AlPh@Next\relax
236 \AlPh@TestNumber#1!{#2}%
237 \ifAlPh@Unavailable
238 \expandafter\AlPh@BinSearch\AlPh@Number!#1!{#2}%
239 \else
240 \def\AlPh@Number{#1}%
241 \ifnum#1>1073741823 %
242 \AlPh@TestNumber2147483647!{#2}%
243 \ifAlPh@Unavailable
244 \AlPh@BinSearch#1!2147483647!{#2}%
245 \else
246 \def\AlPh@Number{0}%
247 \AlPh@Error{%
248 Maximal symbol number not found%
249 }%
250 \fi
251 \else
252 \def\AlPh@Next{%
253 \expandafter\AlPh@ExpSearch\number\intcalcShl{#1}{#2}%
254 }%
255 \fi
256 \fi
257 \AlPh@Next
258 }
\AlPh@BinSearch
\#1: available number
\#2: unavailable number, \#2 \gt \#1
\#3: symbols provider
259 \def\AlPh@BinSearch#1!#2!#3{%
260 \expandafter\AlPh@ProcessBinSearch
261 \number\intcalcShr{\intcalcAdd{#1}{#2}}{#3}%
262 #1!#2!{#3}%
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}\fi \else \def\AlPh@Number{#2}\fi \else \def\AlPh@Number{#2}\fi \AlPh@Next}  

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

13
2.8.4 Method ‘mult’

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

\[ x := \text{number to be converted} \]
\[ n := \text{number of symbols} \]
\[ r := \text{repetition length} \]
\[ s := \text{symbol slot} \]
\[ r = \left(\frac{x - 1}{n}\right) + 1 \]
\[ s = ((x - 1) \mod n) + 1 \]
2.9 User interface

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}{%\AlPh@newalphalph{#1}%
  \if\AlPh@newalphalph{#1}{alph}{%}
  \if\AlPh@newalph alph{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
        \if\AlPh@GetNumberOfSymbols{#3}{%
          \ifcase\AlPh@Number
            \else
              \begingroup
              \escapechar=92 % backslash
              \@PackageInfo{alphalph}{%
                Number of symbols for \string#1 is \AlPh@Number%
              }
              \endgroup
              \AlPh@NewAlphAlph
              \csname AlPh@Method@#2\endcsname\AlPh@Number!{#1}{#3}{#4}{%
            \else
              \expandafter\AlPh@NewAlphAlph
              \csname AlPh@Method@#2\endcsname\number#4!% + \the\numexpr#4!%
              {#1}{#3}{#4}{%}
            \fi
          \fi}
        \else
          \AlPh@NewAlphAlph\csname AlPh@Method@#2\endcsname
        \fi}
      \fi}
  \else
    \expandafter\AlPh@NewAlphAlph
    \csname AlPh@Method@#2\endcsname\number#4!% + \the\numexpr#4!%
    {#1}{#3}{#4}{%}
  \fi
\fi\if\AlPh@newalph alph{1}[\#2]{#3}{#4}{%
\AlPh@newalph alph{1}{alph}%
```

```
\AlPh@newalph alph #1: method macro
#2: number of symbols
#3: cmd
#4: symbols provider
\AlPh@newalph alph #1[#2][#3][#4]{%
\AlPh@IfDefinable#3{%
  \ifnum#2>0 %
    \def#3{#1{#2}{#4}{%
```
\AlPh@Error{ %
   Definition of \string#3 failed,\MessageBreak
   because number of symbols (#2) is not positive% %
\fi %}
}%
}}%
\AlphAlph
\newalphalph\AlphAlph\alphalph@Alph{26}

\alphaalph
\newalphalph\alphaalph\alphalph@alpha{26}
\AlPh@AtEnd%
{/package}

3 Test

3.1 Catcode checks for loading
\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
\def\RangeCatcodeInvalid#1#2{\%
  \count@=#1\relax
  \loop
    \setbox0\hbox{
      \catcode\the\count@=15 \%
      \ifnum\count@<#2\relax
        \advance\count@ 1 \%
      \repeat
    }
  \ifnum\count@=#2\relax
    \errmessage{Character \the\count@ space with wrong \catcode \the\count@ space instead of \number#2\%}
  \else
    \ifnum\count@<#1\relax
      \advance\count@ 1 \%
    \repeat
  \fi
  \fi
\def\RangeCatcodeCheck#1#2#3{\%
  \count@=#1\relax
  \loop
    \ifnum#3=\catcode\count@
      \else \errmessage{Character \the\count@ space with wrong \catcode \the\count@ space instead of \number#3\%}
    \fi
    \ifnum\count@<#2\relax
      \advance\count@ 1 \%
    \repeat
  \fi
\def\space{ }
\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

\begin{verbatim}
\NeedsTeXFormat{LaTeX2e}
\nofiles
\documentclass{article}
\begin{document}
\newcommand*{\DisableNumexpr}{\let\numexpr\@undefined}
\newcommand*{\RestoreNumexpr}{\let\numexpr\saved@numexpr}
\DisableNumexpr
\usepackage{alphalph}[2016/05/16]
\usepackage{qstest}
\IncludeTests{*}
\LogTests{log}{*}{*}
\newcommand*{\TestCmd}[3]{\setbox0=\hbox{\DisableNumexpr\edef\TestString{#1{#2}}\expandafter\Expect\expandafter{\TestString}{#3}\edef\TestString{#1{#2} }\expandafter\Expect\expandafter{\TestString}{#3 }}}\Expect{\the\wd0}{0.0pt}
\end{document}
\end{verbatim}

\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}% 
\begin{qstest}{vocals}{vocals}% 
\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}% 
\end{qstest}
\textbf{TestVocals} (0)\%  
\textbf{TestVocals} (1)\{A\}\%  
\textbf{TestVocals} (2)\{E\}\%  
\textbf{TestVocals} (3)\{I\}\%  
\textbf{TestVocals} (4)\{O\}\%  
\textbf{TestVocals} (5)\{U\}\%  
\textbf{TestVocals} (6)\{AA\}\%  
\textbf{TestVocals} (7)\{AE\}\%  
\textbf{TestVocals} (8)\{AI\}\%  
\textbf{TestVocals} (9)\{AO\}\%  
\textbf{TestVocals} (10)\{AU\}\%  
\textbf{TestVocals} (11)\{EA\}\%  
\textbf{TestVocals} (24)\{OO\}\%  
\textbf{TestVocals} (25)\{OU\}\%  
\textbf{TestVocals} (26)\{UA\}\%  
\textbf{TestVocals} (29)\{UU\}\%  
\textbf{TestVocals} (30)\{UUU\}\%  
\textbf{TestVocals} (31)\{AAA\}\%  
\textbf{TestVocals} (155)\{UUU\}\%  
\textbf{TestVocals} (156)\{AAAA\}\%  
\textbf{TestVocals} (2147483647)\{AAAAIIIOIIOIIUOE\} \%
\end{qstest}

\texttt{\makeatletter}
\newalphalph\AlphMult{\alphalph@Alph}{26}
\newalphalph\alphmult{\alphalph@alph}{26}
\newalphalph\LaTeXAlphMult{\LaTeX@Alph}{26}
\newalphalph\LaTeXalphmult{\LaTeX@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}{aaaaaaa} %
\TestMult{262}{bbbbbbbbbbbb}%
\end{qstest}

\texttt{\def\myvocalsB#1{%  
\ifcase#1\or A\or E\or I\or O\or U\fi %\end{qstest}}
\begin{qstest}{symbolnum}{symbolnum}
\makeatletter
\def\Test#1#2{\let\TestCmd\relax
\newalphalph\TestCmd{#1}{}%\Expect*{\AlPh@Number}{#2}%}
\end{qstest}
\begin{document}

5 Installation

5.1 Download

Package. This package is available on CTAN:\footnote{http://ctan.org/pkg/alphalph}

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

\end{document}
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 \texttt{texmf} in their name are usually organized this way.

5.2 Bundle installation

**Unpacking.** Unpack the \texttt{oberdiek.tds.zip} in the TDS tree (also known as \texttt{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 \texttt{attachfile2} comes with the Perl script \texttt{pdfatfi.pl} that should be installed in such a way that it can be called as \texttt{pdfatfi}. Example (linux):

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

5.3 Package installation

**Unpacking.** The \texttt{.dtx} file is a self-extracting docstrip archive. The files are extracted by running the \texttt{.dtx} through plain \TeX:\n
```
tex alphalph.dtx
```

**TDS.** Now the different files must be moved into the different directories in your installation TDS tree (also known as \texttt{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 \texttt{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 (\texttt{te\TeX}, \texttt{mikTeX}, ...) relies on file name databases, you must refresh these. For example, \texttt{te\TeX} users run \texttt{texhash} or \texttt{mktexlar}.

5.5 Some details for the interested

**Unpacking with \LaTeX.** The \texttt{.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:

```
l latex \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 pdfL\TeX:\n
```
pdflatex alphalph.dtx
makeindex -s gind.ist alphalph.idx
pdflatex alphalph.dtx
makeindex -s gind.ist alphalph.idx
pdflatex alphalph.dtx
```

6 Catalogue

The following XML file can be used as source for the \TeX\ Catalogue. The elements caption and description are imported from the original XML file from the Catalogue. The name of the XML file in the Catalogue is alphalph.xml.

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE entry SYSTEM 'catalogue.dtd'>
<entry datestamp='$Date$' modifier='$Author$' id='alphalph'>
  <name>alphalph</name>
  <caption>Convert numbers to letters.</caption>
  <authorref id='auth:oberdiek'/>
  <license type='lppl1.3'/>
  <version number='2.5'/>
  <description>
    Provides commands \texttt{\alphalph} and \texttt{\AlphAlph}. They are like \texttt{\number} but the expansion consists of lowercase and uppercase letters respectively (1 to a, 26 to z, 27 to aa, 52 to zz, 53 to ba, 702 to zz, 703 to aaa, etc.). Can be used as a replacement for LaTeX's \texttt{\@alph} and \texttt{\@Alph} macros.
  </description>
  <documentation details='Package documentation' href='ctan:/macros/latex/contrib/oberdiek/alphalph.pdf'/>
  <ctan file='true' path='/macros/latex/contrib/oberdiek/alphalph.dtx'/>
  <miktex location='oberdiek'/>
  <texlive location='oberdiek'/>
  <install path='/macros/latex/contrib/oberdiek/oberdiek.tds.zip'/>
</entry>
```

7 History

[1999/03/19 v0.1]

- The first version was built as a response to a question\textsuperscript{2} of Will Douglas\textsuperscript{3} and the request\textsuperscript{4} of Donald Arsenau\textsuperscript{5}, published in the newsgroup comp.text.tex: “Re: alph counters > 26”\textsuperscript{6}

\textsuperscript{2}Url: http://groups.google.com/group/comp.text.tex/msg/17a74cd721641038
\textsuperscript{3}Will Douglas's email address: william.douglas@wolfson.ox.ac.uk
\textsuperscript{4}Url: http://groups.google.com/group/comp.text.tex/msg/8f9768825640315f
\textsuperscript{5}Donald Arsenau's email address: sand@reg.triumf.ca
\textsuperscript{6}Url: http://groups.google.com/group/comp.text.tex/msg/cee563ee8bf65d0
• Copyright: LPPL (CTAN:macros/latex/base/lppl.txt)

[1999/04/12 v1.0]
• Documentation added in dtx format.
• $\varepsilon$-\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]
• \newalphaphi added.

[2007/04/11 v1.5]
• Line ends sanitized.

[2007/09/09 v2.0]
• New implementation that uses package \intcalc. This removes the dependency on $\varepsilon$-\TeX.
• \newalphaphi 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).
• Documentation fixes (Jim Diamond) and using package hologo for the documentation.

• Catalogue file added.

[2016/05/16 v2.5]

• Documentation updates.

8 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.

Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>435</td>
</tr>
<tr>
<td>%</td>
<td>511</td>
</tr>
<tr>
<td>@</td>
<td>436, 509</td>
</tr>
<tr>
<td>@Alph</td>
<td>564, 568, 680, 718, 778</td>
</tr>
<tr>
<td>@LastSymbol</td>
<td>573, 576, 580</td>
</tr>
<tr>
<td>@PackageError</td>
<td>136</td>
</tr>
<tr>
<td>@PackageInfo</td>
<td>399</td>
</tr>
<tr>
<td>\ReturnAfterElseFi</td>
<td>153, 301, 329, 352</td>
</tr>
<tr>
<td>\ReturnAfterFi</td>
<td>513, 311, 335, 361, 369</td>
</tr>
<tr>
<td>\alpha</td>
<td>565, 569, 681, 717, 778</td>
</tr>
<tr>
<td>\ctrerr</td>
<td>224</td>
</tr>
<tr>
<td>\ecl</td>
<td>136</td>
</tr>
<tr>
<td>\firstofone</td>
<td>444, 447</td>
</tr>
<tr>
<td>\fnsymbol</td>
<td>719</td>
</tr>
<tr>
<td>\gobble</td>
<td>441, 449</td>
</tr>
<tr>
<td>\gobblefour</td>
<td>155, 288</td>
</tr>
<tr>
<td>\ifdefinable</td>
<td>151</td>
</tr>
<tr>
<td>\ifnextchar</td>
<td>176</td>
</tr>
<tr>
<td>\nil</td>
<td>573, 576, 580</td>
</tr>
<tr>
<td>\onelevel@sanitize</td>
<td>739, 744</td>
</tr>
<tr>
<td>\undefined</td>
<td>58, 142, 539</td>
</tr>
<tr>
<td>{</td>
<td>235</td>
</tr>
<tr>
<td>}</td>
<td>236</td>
</tr>
<tr>
<td>\advance</td>
<td>474, 482, 497</td>
</tr>
<tr>
<td>\aftergroup</td>
<td>29</td>
</tr>
<tr>
<td>\AlPh@AtEnd</td>
<td>95, 96, 116, 430</td>
</tr>
<tr>
<td>\AlPh@BinSearch</td>
<td>238, 244, 259, 271, 275</td>
</tr>
<tr>
<td>\AlPh@BracketLeft</td>
<td>167, 169</td>
</tr>
<tr>
<td>\AlPh@CheckPositive</td>
<td>573, 293, 321, 344</td>
</tr>
<tr>
<td>\AlPh@cterr</td>
<td>187, 197, 201, 225</td>
</tr>
<tr>
<td>\AlPh@Error</td>
<td>133, 145, 206, 247, 387, 421</td>
</tr>
<tr>
<td>\AlPh@ExpSearch</td>
<td>209, 234</td>
</tr>
<tr>
<td>\AlPh@GetNumberOfSymbols</td>
<td>202, 294</td>
</tr>
<tr>
<td>\AlPh@IfDefinable</td>
<td>139, 151, 375, 417</td>
</tr>
<tr>
<td>\AlPh@IfOptArg</td>
<td>158, 162, 176, 179, 377</td>
</tr>
<tr>
<td>\AlPh@IfOptArg@Next</td>
<td>165, 168</td>
</tr>
<tr>
<td>\AlPh@Method@alpha</td>
<td>292</td>
</tr>
<tr>
<td>\AlPh@Method@mult</td>
<td>343</td>
</tr>
<tr>
<td>\AlPh@Method@wrap</td>
<td>320</td>
</tr>
<tr>
<td>\AlPh@NewAlphAlph</td>
<td>403, 408, 416</td>
</tr>
</tbody>
</table>

A

\begin | 601, 629, 653, 689, 710, 725, 760 |
\body | 453, 457 |

B

\catcode | 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 69, 70, 72, 73, 74, 78, 79, 80, 81, 82, 83, 84, 87, 88, 90, 91, 92, 93, 97, 99, 127, 128, 130, 131, 433, 434, 435, 436, 471, 480, 488, 492, 509, 510, 511 |