The akshar package
Vu Van Dung
Version 0.2 — 2020/09/06

Abstract
This package provides tools to deal with special characters in a Devanagari string.

Contents
1 Introduction 1
2 User manual 1
  2.1 \LaTeX{} macros 2
  2.2 expl3 functions 3
3 Implementation 3
  3.1 Variable declarations 3
  3.2 Messages 4
  3.3 Utilities 5
  3.4 The \texttt{\akshar\_convert:nn} function and its variants 6
  3.5 Other internal functions 7
  3.6 Front-end \LaTeX{} macros 9
Index 12

1 Introduction

When dealing with processing strings in the Devanagari script, normal \LaTeX{} commands usually find some difficulties in distinguishing “normal” characters, like क, and “special” characters, for example ओ or नी. Let’s consider this example code:

\begin{verbatim}
\ExplSyntaxOn
  \tl_set:Nn \l_tmpa_tl { क}
  \tl_count:N \l_tmpa_tl \c_space_token tokens.
\ExplSyntaxOff
\end{verbatim}

The output is 2, but the number of characters in it is only one! The reason is quite simple: the compiler treats नी as a normal character, and it shouldn’t do so.

To tackle that, this package provides expl3 functions to “convert” a given string, written in the Devanagari script, to a sequence of token lists. Each of these token lists is a “true” Devanagari character. You can now do anything you want with this sequence; and this package does provide some front-end macros for some simple actions on the input string.

2 User manual

Due to the current implementation, all of these macros and functions are not expandable.
2.1 \LaTeX 2 macros

\texttt{\textbackslash \text{aksharStrLen}} \{(token list)\}

Return the number of Devanagari characters in the \{(token list)\}.

\begin{itemize}
  \item There are \texttt{\textbackslash \text{aksharStrLen}} \{नमस्कार\} characters in नमस्कार, \texttt{expl3} returns 7, which is wrong.
  \item \texttt{\textbackslash \text{ExplSyntaxOn}}
  \item \texttt{\textbf{\pkg{expl3}}\textbackslash \text{\textbackslash tl\_count:n} \{ नमस्कार\}, which is wrong.
  \item \texttt{\textbackslash \text{ExplSyntaxOff}}
\end{itemize}

\texttt{\textbackslash \text{aksharStrHead}} \{(token list)\} \{\textbackslash \text{n}\}

Return the first character of the token list.

\begin{itemize}
  \item \texttt{\textbackslash \text{aksharStrHead}} \{ मं\}
\end{itemize}

\texttt{\textbackslash \text{aksharStrTail}} \{(token list)\} \{\textbackslash \text{n}\}

Return the last character of the token list.

\begin{itemize}
  \item \texttt{\textbackslash \text{aksharStrTail}} \{ मं\}
\end{itemize}

\texttt{\textbackslash \text{aksharStrChar}} \{(token list)\} \{\textbackslash \text{n}\}

Return the \textbackslash \text{n}-th character of the token list.

\begin{itemize}
  \item 3rd character of नमस्कार is स्का. It is not स.
\end{itemize}

\texttt{\textbackslash \text{aksharStrReplace}} \{(tl 1)\} \{(tl 2)\} \{(tl 3)\}

Replace all occurrences of \{(tl 2)\} in \{(tl 1)\} with \{(tl 3)\}, and leaves the modified \{(tl 1)\} in the input stream.

The starred variant will replace only the first occurrence of \{(tl 2)\}, all others are left intact.

\texttt{\textbackslash \text{ExplSyntaxOn}}
\texttt{\textbf{\pkg{expl3}}} ~ output:
\texttt{\textbackslash \text{\textbackslash tl\_set:Nn} \{ मममडडमंळममड\}}
\texttt{\textbackslash \text{\textbackslash tl\_replace:all:Nnn} \{ मममडडमंळममड\} \{ म\} \{ स्का\}}
\texttt{\textbackslash \text{\textbackslash tl\_use:N} \{ मममडडमंळममड\} \texttt{par}}
\texttt{\textbackslash \text{\textbackslash ExplSyntaxOff}}
\texttt{\textbf{\cs{aksharStrReplace}}} output:
\texttt{\textbackslash \text{aksharStrReplace} \{ मममडडमंळममड\} \{ म\} \{ स्का\}}

\texttt{\textbackslash \text{ExplSyntaxOn}}
\texttt{\textbf{\pkg{expl3}}} ~ output:
\texttt{\textbackslash \text{\textbackslash tl\_set:Nn} \{ ममंममडडमंळममड\}}
\texttt{\textbackslash \text{\textbackslash tl\_replace:once:Nnn} \{ ममंममडडमंळममड\} \{ मम\} \{ स्का\}}
\texttt{\textbackslash \text{\textbackslash tl\_use:N} \{ ममंममडडमंळममड\} \texttt{par}}
\texttt{\textbackslash \text{\textbackslash ExplSyntaxOff}}
\texttt{\textbf{\cs{aksharStrReplace*}}} output:
\texttt{\textbackslash \text{aksharStrReplace*} \{ ममंममडडमंळममड\} \{ मम\} \{ स्का\}}

\texttt{\textbackslash \text{aksharStrRemove}} \{(tl 1)\} \{(tl 2)\}

Remove all occurrences of \{(tl 2)\} in \{(tl 1)\}, and leaves the modified \{(tl 1)\} in the input stream.

The starred variant will remove only the first occurrence of \{(tl 2)\}, all others are left intact.
This section assumes that you have a basic knowledge in \LaTeX\ programming. All macros in 2.1 directly depend on the following function, so it is much more powerful than all features we have described above.

\texttt{\textbackslash \texttt{akshar\_convert:Nn}}  \\
\texttt{\textbackslash \texttt{akshar\_convert:(cn|Nx|cx)}}

This function converts \texttt{\{token list\}} to a sequence of characters, that sequence is stored in \texttt{\{seq var\}}.

3 Implementation

\texttt{\textbackslash \texttt{@\texttt{=}akshar}}  \\
\texttt{\texttt{@\texttt{package}}}  \\

Declare the package. By loading \texttt{fontspec}, \texttt{xparse}, and in turn, \texttt{expl3}, are also loaded.

3.1 Variable declarations

These variables store the special characters we need to take into account:

\begin{itemize}
  \item \texttt{\textbackslash \texttt{c\_akshar\_joining\_tl}} is the "connecting" character ◌्.
  \item \texttt{\textbackslash \texttt{c\_akshar\_diacritics\_tl}} is the list of all diacritics: ◌ा, ◌ी, ◌ू, ◌े, ◌ै, ◌ाे, ◌ाै, ◌ं, ◌ः, ◌ॢ, ◌ृ, ◌ॅ, ◌ँ, ◌़, ◌ॆ, ◌ाॆ, ◌ाॅ, ◌ै, ◌॑, ◌॒, ◌॓, ◌॔, ◌ॕ, ◌ॖ, ◌ॗ, ◌ॄ, ◌ऺ.
\end{itemize}

\texttt{\textbackslash tl\_const:Nn \textbackslash \texttt{c\_akshar\_joining\_tl} \{ \texttt{◌्} \}}  \\
\texttt{\textbackslash tl\_const:Nn \textbackslash \texttt{c\_akshar\_diacritics\_tl} \{}


When we get to a normal character, we need to know whether it is joined, i.e. whether the previous character is the joining character. This boolean variable takes care of that.

\bool_new:N \l__akshar_prev_joining_bool

This local sequence stores the output of the converter.

\seq_new:N \l__akshar_char_seq

Some self-descriptive constant variables.

\tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
\tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
\tl_const:Nx \c__akshar_str_comma_tl { \tl_to_str:n {,} }

Some temporary variables.

\tl_new:N \l__akshar_tmpa_tl
\tl_new:N \l__akshar_tmpb_tl
\seq_new:N \l__akshar_tmpa_seq
\seq_new:N \l__akshar_tmpb_seq
\seq_new:N \l__akshar_tmpc_seq
\seq_new:N \l__akshar_tmpd_seq
\seq_new:N \l__akshar_tmpe_seq
\int_new:N \l__akshar_tmpa_int
\int_new:N \l__akshar_tmpb_int

3.2 Messages

In \akshar_convert:Nn and friends, the argument needs to be a sequence variable. There will be an error if it isn’t.

\msg_new:nnnn { akshar } { err_not_a_sequence_variable }
\{ #1 ~ is ~ not ~ a ~ valid ~ LaTeX3 ~ sequence ~ variable. \}
\{
    You ~ have ~ requested ~ me ~ to ~ assign ~ some ~ value ~ to ~
    the ~ control ~ sequence ~ #1, ~ but ~ it ~ is ~ not ~ a ~ valid ~
    sequence ~ variable. ~ Read ~ the ~ documentation ~ of ~ expl3 ~
    for ~ more ~ information. ~ Proceed ~ and ~ I ~ will ~ pretend ~
    that ~ #1 ~ is ~ a ~ local ~ sequence ~ variable ~ (beware ~ that ~
    unexpected ~ behaviours ~ may ~ occur). ~
\}

In \aksharStrChar, we need to guard against accessing an ‘out-of-bound’ character (like trying to get the 8th character in a 5-character string.)

\msg_new:nnnn { akshar } { err_character_out_of_bound }
\{ Character ~ index ~ out ~ of ~ bound. \}
\{
    You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~
    string ~ #1. ~ However ~ that ~ character ~ doesn’t ~ exist. ~
    Make ~ sure ~ that ~ you ~ use ~ a ~ number ~ between ~ and ~ not ~
    including ~ 0 ~ and ~ #3, ~ so ~ that ~ I ~ can ~ return ~ a ~
    good ~ output. ~ Proceed ~ and ~ I ~ will ~ return ~
    \token_to_str:N \scan_stop:. ~
\}
In `\aksharStrHead` and `\aksharStrTail`, the string must not be blank.

```latex
\msg_new:nnnn { akshar } { err_string_empty } { The ~ input ~ string ~ is ~ empty. }

To ~ get ~ the ~ #1 ~ character ~ of ~ a ~ string, ~ that ~ string ~ must ~ not ~ be ~ empty, ~ but ~ the ~ input ~ string ~ is ~ empty. Make ~ sure ~ the ~ string ~ contains ~ something, ~ or ~ proceed ~ and ~ I ~ will ~ use ~ \token_to_str:N \scan_stop:.
```

### 3.3 Utilities

`\tl_if_in:No` When we get to a character which is not the joining one, we need to know if it is a diacritic. The current character is stored in a variable, so an expanded variant is needed. We only need it to expand only once.

```latex
\prg_generate_conditional_variant:Nnn \tl_if_in:Nn { No } { TF }
```

(End definition for `\tl_if_in:NoTF`.)

`\seq_set_split:Nxx` A variant we will need in `\__akshar_var_if_global`.

```latex
\cs_generate_variant:Nn \seq_set_split:Nnn { Nxx }
```

(End definition for `\seq_set_split:Nxx`.)

`\msg_error:nxx` Some variants of \l3msg functions that we will need when issuing error messages.

```latex
\cs_generate_variant:Nn \msg_error:nnn { nnx }
```

(End definition for `\msg_error:nxx` and `\msg_error:nnnxx`.)

`\__akshar_tl_if_in_ncomma:NNTF` This conditional is essentially `\tl_if_in:Nn`, but if #2 is a comma this conditional always return false.

```latex
\prg_new_conditional:Npnn \__akshar_tl_if_in_ncomma:N #1 #2 { T, F, TF }
```

(End definition for `\__akshar_tl_if_in_ncomma:NNTF`.)

`\__akshar_var_if_global:N` This conditional checks if #1 is a global sequence variable or not. In other words, it returns true iff #1 is a control sequence in the format `\g_{<name>}_seq`. If it is not a sequence variable, this function will (TODO) issue an error message.

```latex
\prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
```

(End definition for `\__akshar_var_if_global:N`.)
\__akshar_int_append_ordinal:n

Append st, nd, rd or th to integer #1. Will be needed in error messages.

\cs_new:Npn \__akshar_int_append_ordinal:n #1
{  \int_case:nnF { #1 }  
  { { 11 } } { th }  
  { { 12 } } { th }  
  { { 13 } } { th }  
  { { -11 } } { th }  
  { { -12 } } { th }  
  { { -13 } } { th }  
  { \int_compare:nNnTF { #1 } > { -1 }  
    { \int_case:nnF { #1 - 10 * (#1 / 10) }  
      { { 1 } } { st }  
      { { 2 } } { nd }  
      { { 3 } } { rd }  
      { th }  
    }  
    { \int_case:nnF { (- #1) - 10 * ((- #1) / 10) }  
      { { 1 } } { st }  
      { { 2 } } { nd }  
      { { 3 } } { rd }  
      { th }  
    }  
  }  
}

(End definition for \__akshar_int_append_ordinal:n.)

3.4 The \akshar_convert:Nn function and its variants

This converts #2 to a sequence of true Devanagari characters. The sequence is set to #1, which should be a sequence variable.

\cs_new:Npn \akshar_convert:Nn #1 #2
{ Clear anything stored in advance. We don’t want different calls of the function to conflict with each other.
  \seq_clear:N \l__akshar_char_seq \bool_set_false:N \l__akshar_prev_joining_bool  
  \tl_map_variable:NNn {#2} \__akshar_map_tl  
  \__akshar_tl_if_in_ncomma:NNTF \__akshar_diakritics_tl \__akshar_map_tl  
}

(End definition for \akshar_convert:Nn.)
It is a diacritic. We append the current diacritic to the last item of the sequence instead of pushing the diacritic to a new sequence item.

\seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
\seq_put_right:Nx \l__akshar_char_seq
{ \l__akshar_tmpa_tl \l__akshar_map_tl }

} 

\tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl 

In this case, the character is the joining character, ◌्. What we do is similar to the above case, but \l__akshar_prev_joining_bool is set to true so that the next character is also appended to this item.

\seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
\seq_put_right:Nx \l__akshar_char_seq
{ \l__akshar_tmpa_tl \l__akshar_map_tl }
\bool_set_true:N \l__akshar_prev_joining_bool

Now the character is normal. We see if we can push to a new item or not. It depends on the boolean variable.

\bool_if:NTF \l__akshar_prev_joining_bool 

\seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
\seq_put_right:Nx \l__akshar_char_seq
{ \l__akshar_tmpa_tl \l__akshar_map_tl }
\bool_set_false:N \l__akshar_prev_joining_bool

\seq_put_right:Nx \l__akshar_char_seq \l__akshar_map_tl 

Set #1 to \l__akshar_char_seq. The package automatically determines whether the variable is a global one or a local one.

\__akshar_var_if_global:NTF #1 

\seq_gset_eq:NN #1 \l__akshar_char_seq 
\seq_set_eq:NN #1 \l__akshar_char_seq

Generate variants that might be helpful for some.

\cs_generate_variant:Nn \akshar_convert:Nn { cn, Nx, cx }

(End definition for \akshar_convert:Nn. This function is documented on page 3.)

3.5 Other internal functions

\__akshar_seq_push_seq:NN
Append sequence #1 to the end of sequence #2. A simple loop will do.

\cs_new:Npn \__akshar_seq_push_seq:NN #1 #2
{ \seq_map_inline:Nn \seq_map_inline:Nn \seq_map_inline:Nn \seq_map_inline:Nn \seq_map_inline:Nn #1 #2 { \seq_put_right:Nn #1 { #1 } } }

(End definition for \__akshar_seq_push_seq:NN.)

\__akshar_replace:NnnnN
If #5 is \c_false_bool, this function replaces all occurrences of #3 in #2 by #4 and stores the output sequence to #1. If #5 is \c_true_bool, the replacement only happens once.

The algorithm used in this function: We will use \l__akshar_tmpa_int to store the “current position” in the sequence of #3. At first it is set to 1.
We will store any subsequence of #2 that may match #3 to a temporary sequence. If it doesn’t match, we push this temporary sequence to the output, but if it matches, #4 is pushed instead.

We loop over #2. For each of these loops, we need to make sure the \_\_akshar_tmpa_int-th item must indeed appear in #3. So we need to compare that with the length of #3.

- If now \_\_akshar_tmpa_int is greater than the length of #3, the whole of #3 has been matched somewhere, so we reinitialize the integer to 1 and push #4 to the output.

Note that it is possible that the current character might be the start of another match, so we have to compare it to the first character of #3. If they are not the same, we may now push the current mapping character to the output and proceed; otherwise the current character is pushed to the temporary variable.

- Otherwise, we compare the current loop character of #2 with the \_\_akshar_tmpa_int-th character of #3.
  - If they are the same, we still have a chance that it will match, so we increase the “iterator” \_\_akshar_tmpa_int by 1 and push the current mapping character to the temporary sequence.
  - If they are the same, the temporary sequence won’t match. Let’s push that sequence to the output and set the iterator back to 1.

Note that now the iterator has changed. Who knows whether the current character may start a match? Let’s compare it to the first character of #3, and do as in the case of \_\_akshar_tmpa_int is greater than the length of #3.

The complexity of this algorithm is \(O(m \max(n, p))\), where \(m, n, p\) are the lengths of the sequences created from #2, #3 and #4. As #3 and #4 are generally short strings, this is (almost) linear to the length of the original sequence #2.

```latex
\cs_new:Npn \_\_akshar_replace:NnnnN #1 #2 #3 #4 #5
{\akshar_convert:Nn \l__akshar_tmpc_seq {#2}}
{\akshar_convert:Nn \l__akshar_tmpd_seq {#3}}
{\akshar_convert:Nn \l__akshar_tmpe_seq {#4}}
{\seq_clear:N \l__akshar_tmpa_seq}
{\seq_clear:N \l__akshar_tmpb_seq}
{\int_set:Nn \l__akshar_tmpa_int { 1 }}
{\int_set:Nn \l__akshar_tmpb_int { 0 }}
{\seq_map_variable:NNn \l__akshar_tmpc_seq \l__akshar_map_tl}
{\int_compare:nNnTF {\l__akshar_tmpb_int} > {0}}
{\seq_put_right:NV \l__akshar_tmpb_seq \l__akshar_map_tl}
{\int_compare:nNnTF \{\l__akshar_tmpa_int\} = \{1 + \seq_count:N \l__akshar_tmpd_seq\}}
{\bool_if:NT \{#5\}}
{\int_incr:N \l__akshar_tmpb_int}
{\seq_clear:N \l__akshar_tmpb_seq}
{\seq_put_right:NV \l__akshar_tmpb_seq \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ \n\l__akshar TMPSEQ}
{\seq_put_right:NV \l__akshar_map_tl \l__akshar_map_tl}
{\tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_map_tl}
{\int_incr:N \l__akshar_tmpa_int}
{\seq_put_right:NV \l__akshar_tmpb_int \l__akshar_map_tl}
{\seq_put_right:NV \l__akshar_map_tl \l__akshar_map_tl}
```

8
\begin{verbatim}
{ \tl_set:Nx \l__akshar_tmpa_tl
  { \seq_item:Nn \l__akshar_tmpd_seq { \l__akshar_tmpa_int } }
\tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
  { \int_incr:N \l__akshar_tmpa_int
    \seq_put_right:NV \l__akshar_tmpb_seq \l__akshar_map_tl }
  { \int_set:Nn \l__akshar_tmpa_int { 1 }
    \__akshar_seq_push_seq:NN
    \l__akshar_tmpa_seq \l__akshar_tmpb_seq
    \seq_clear:N \l__akshar_tmpb_seq
    \tl_set:Nx \l__akshar_tmpa_tl
    { \seq_item:Nn \l__akshar_tmpd_seq { 1 } }
    \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
      { \int_incr:N \l__akshar_tmpa_int
        \seq_put_right:NV
        \l__akshar_tmpb_seq \l__akshar_map_tl }
      { \seq_put_right:NV
        \l__akshar_tmpa_seq \l__akshar_map_tl } }
  }
\__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpb_seq
\__akshar_var_if_global:NTF #1
  { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
  { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
}
\end{verbatim}

(End definition for \__akshar_replace:NnnnN.)

\section{Front-end \LaTeX macros}

**\aksharStrLen** Expands to the length of the string.

\begin{verbatim}
\NewDocumentCommand \aksharStrLen {m}
  { \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
    \seq_count:N \l__akshar_tmpa_seq }
\end{verbatim}

(End definition for \aksharStrLen. This function is documented on page \pageref{aksharStrLen}.)

**\aksharStrChar** Returns the $n$-th character of the string.

\begin{verbatim}
\NewDocumentCommand \aksharStrChar {mm}
  { \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
    \bool_if:nTF
      { \int_compare_p:nNn {#2} > {0} &&
        \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq} }
      { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
      { \msg_error:nnnxx { akshar } { err_character_out_of_bound }
        { #1 } { \__akshar_int_append_ordinal:n { #2 } }
      }
    \{}\scan_stop:
      { 1 + \seq_count:N \l__akshar_tmpa_seq }
  }
\end{verbatim}
Return the first character of the string.

\NewDocumentCommand \aksharStrHead {m}
{\akshar_convert:Nn \l__akshar_tmpa_seq {#1}}\int_compare:nNnTF {\seq_count:N \l__akshar_tmpa_seq } = {0}
{\msg_error:nnn {akshar} {err_character_out_of_bound}
{first}
\scan_stop:
\seq_item:Nn \l__akshar_tmpa_seq {1}}

Return the last character of the string.

\NewDocumentCommand \aksharStrTail {m}
{\akshar_convert:Nn \l__akshar_tmpa_seq {#1}}\int_compare:nNnTF {\seq_count:N \l__akshar_tmpa_seq } = {0}
{\msg_error:nnn {akshar} {err_character_out_of_bound}
{last}
\scan_stop:
\seq_item:Nn \l__akshar_tmpa_seq {\seq_count:N \l__akshar_tmpa_seq}}

Replace occurrences of #3 of a string #2 with another string #4.

\NewDocumentCommand \aksharStrReplace {smmm}
{\IfBooleanTF {#1}
 {\__akshar_replace:NnnnN \l__akshar_tmpa_seq {#2} {#3} {#4} \c_true_bool}
 {\__akshar_replace:NnnnN \l__akshar_tmpa_seq {#2} {#3} {#4} \c_false_bool}
 \seq_use:Nn \l__akshar_tmpa_seq {}}

Remove occurrences of #3 in #2. This is just a special case of \aksharStrReplace.

\NewDocumentCommand \aksharStrRemove {smm}
{\IfBooleanTF {#1}
 {\__akshar_replace:NnnnN \l__akshar_tmpa_seq {#2} {#3} {#4} \c_true_bool}
 {\__akshar_replace:NnnnN \l__akshar_tmpa_seq {#2} {#3} {#4} \c_false_bool}
 \seq_use:Nn \l__akshar_tmpa_seq {}}
(End definition for \texttt{\textbackslash aksharStrRemove} and \texttt{\textbackslash aksharStrRemove*}. These functions are documented on page 2.)

```bash
</package>
```
Index

Underlined page numbers point to the definition, all others indicate the places where it is used or described.

A

akshar commands:
\akshar_convert:Nn 1, 3, 4, 6, 123, 168, 169, 170, 236, 243, 259, 270

akshar internal commands:
\__akshar_char_seq ................. 7, 15, 125, 132, 133, 139, 140, 147, 148, 154, 160, 161
\c__akshar_diacritics_tl 3, 6, 130
\__akshar_int_append_ordinal:n .................................................. 91, 252
\c__akshar_joining_tl 3, 6, 137
\__akshar_map_tl .................... 127, 130, 134, 137, 141, 149, 154, 175, 178, 191, 194, 197, 205, 208, 217, 221, 225
\__akshar_prev_joining_bool .. .... 7, 14, 126, 142, 145, 150
\__akshar_replace:NnNNN ........... 166, 283, 287, 296, 300
\__akshar_seq_push_seq:NN ........... 164, 186, 212, 231
\c__akshar_str_comma_tl ............ 16, 62
\c__akshar_str_g_tl ................. 16, 81
\c__akshar_str_seq_tl ................ 16, 79
\__akshar_tl_if_in_ncomma:NNTF ................. 60, 129
\__akshar_tmpa_int .................. 7, 8, 19, 173, 181, 188, 193, 203, 207, 211, 219
\__akshar_tmpa_tl ................... 19, 77, 81, 132, 134, 139, 141, 147, 149, 189, 191, 201, 205, 215, 217
\__akshar_tmpb_int .................. 19, 174, 177, 184
\__akshar_tmpb_seq ................. 19, 75, 77, 78, 172, 178, 185, 194, 208, 213, 214, 221, 231
\__akshar_tmpb_tl ................... 19, 78, 79
\__akshar_tmpc_seq ................. 19, 168, 175
\__akshar_tmpd_seq .................. 19, 169, 181, 190, 203, 216
\__akshar_tmpm_seq ................. 19, 170, 187
\__akshar_var_if_global ............ 5
\__akshar_var_if_global:NTF ......... 65, 159, 232
\aksharPackageDate ................. 5
\aksharPackageDescription ........... 5
\aksharPackageName ................. 4
\aksharPackageVersion .............. 5
\aksharStrChar ..................... 2, 4, 241
\aksharStrHead ..................... 2, 5, 257
\aksharStrLen ...................... 2, 236
\aksharStrRemove ................... 2, 3, 292
\aksharStrRemove* ................... 2, 3, 292
\aksharStrReplace .................. 2, 10, 279
\aksharStrReplace* .................. 2, 279
\aksharStrTail ...................... 2, 5, 268

B

bool commands:
\bool_if:N 145, 183
\bool_if:nTF 67, 244
\bool_new:N 14
\bool_set_false:N 126, 150
\bool_set_true:N 142
\c__false Bool 7, 288, 301
\c__true Bool 7, 284, 297

C

cs commands:
\cs_generate_variant:Nn .................................................. 57, 58, 59, 163
\cs_new:Npn 91, 123, 164, 166
\cs_split_function:N .................................................. 68, 76

E

exp commands:
\exp_last_unbraced:Nf 68, 76

I

int commands:
\int_case:nnTF 94, 106, 114
\int_compare:NnTF 104, 177, 180, 260, 271
\int_compare_p:nNn 246, 247
\int_eval:n 253
\int_incr:N 184, 193, 207, 219
\int_new:N 26, 27
\int_set:N 173, 174, 188, 211

M

msg commands:
\msg_error:nnnn ................. 58, 58, 70, 85, 262, 273
\msg_error:nnnnnn .............. 58, 59, 251
\msg_new:nnnn .................. 28, 38, 48

N

\NewDocumentCommand 236, 241, 257, 268, 279, 292

P

prg commands:
\prg_generate_conditional:_variant:Nnn .................................. 56
\prg_new_conditional:Nnnnn .. 60, 65
\prg_return_false: 62, 63, 72, 82, 87
\prg_return_true: 63, 82
\ProvidesExplPackage ............ 4

R

\RequirePackage .................... 3

S

scan commands:
\scan_stop: 46, 54, 254, 264, 275

seq commands:
\seq_clear:N 125, 171, 172, 185, 214
\seq_count:N 181, 239, 247, 253, 260, 271, 277
\seq_get_left:NN ................. 77
\seq_get_right:NN ................ 78
\seq_gset_eq:NN \ldots 160, 233
\seq_item:Nn \ldots \ldots 190, 203, 216, 249, 266, 277
\seq_map_inline:Nn \ldots 165
\seq_map_variable:NNn \ldots 175
\seq_new:N \ldots 15, 21, 22, 23, 24, 25
\seq_pop_right:NN \ldots 132, 139, 147
\seq_put_right:Nn \ldots \ldots 133, 140, 148, 153, 165, 178, 194, 197, 208, 220, 224
\seq_set_eq:NN \ldots 161, 234
\seq_set_split:Nnn \ldots 57, 57, 75
\seq_use:Nn \ldots 290, 303
\tl_commands: \tl_const:Nn \ldots 6, 7, 16, 17, 18

\tl_commands:
\tl_const:Nn \ldots 6, 7, 16, 17, 18

\tl如果您想使用T

token commands:
\token_to_str:N \ldots 46, 54, 71, 75, 86

use commands:
\use_i:nnn \ldots 76
\use_iii:nnn \ldots 68