The akshar package

Vu Van Dung

Version 0.1 - 2020/05/17

https://github.com/joulev/akshar

Abstract

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

Contents

1	Introduction	1
2	User manual	2
	2.1 $AT_EX 2_E$ 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 \akshar_convert:Nn function and its variants	6
	3.5 Other internal functions	7
	3.6 Front-end \LaTeX $2_{\mathcal{E}}$ macros	9
Inde	eX	11

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 ् or ी. Let's consider this example code:

- 2 \tl_set:Nn \l_tmpa_tl { की}

2 tokens.

- $_{\mbox{\scriptsize 3}}$ \tl_count:N \l_tmpa_tl \c_space_token tokens.
- 4 \ExplSyntaxOff

The output is 2, but the number of characters in it is only one! The reason is quite simple: the compiler treats 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

2.1 $\text{LAT}_{EX} 2_{\varepsilon}$ macros

\aksharStrLen $\arrowvert aksharStrLen {\langle token list \rangle}$ Return the number of Devanagari characters in the \taken list\. There are \aksharStrLen{ नमस्कार} characters in नमस्कार.\par There are 4 characters in नमस्कार. 2 \ExplSyntaxOn expl3 returns 7, which is wrong. ³ \pkg{expl3}~returns~\tl_count:n { नमस्कार},~which~is~wrong. 4 \ExplSyntaxOff \aksharStrHead $\arstr Head {\langle token list \rangle} {\langle n \rangle}$ Return the first character of the token list. ा \aksharStrHead { मंळीममड} मं \aksharStrTail $\arstriam{\langle token list \rangle} {\langle n \rangle}$ Return the last character of the token list. ा \aksharStrTail { ळीममडमं} मं \aksharStrChar $\arrowvert aksharStrChar {\langle token list \rangle} {\langle n \rangle}$ Return the *n*-th character of the token list. ा 3rd character of नमस्कारांs \aksharStrChar{ नमस्कार}{3}.\par 3rd character of नमस्कार is स्का. 2 \ExplSyntaxOn 3 It~is~not~\tl item:nn { नमस्कार} {3}. It is not स. 4 \ExplSyntaxOff \aksharStrReplace $\{\langle tl \ 1 \rangle\}$ $\{\langle tl \ 2 \rangle\}$ $\{\langle tl \ 3 \rangle\}$ \aksharStrReplace \aksharStrReplace* Replace all occurences of $\langle tl 2 \rangle$ in $\langle tl 1 \rangle$ with $\langle tl 3 \rangle$, and leaves the modified $\langle tl 1 \rangle$ in the input stream. The starred variant will replace only the first occurence of (tl 2), all others are left intact. 1 \ExplSyntaxOn 2 \pkg{expl3} ~ output:\par 3 \tl set:Nn \l tmpa tl { मममडडमंळीममड} expl3 output: 4 \tl replace all:Nnn \l tmpa_tl { म} { स्का} स्कास्काडडस्कांळीस्कास्काड 5 \tl use:N \l tmpa tl\par \aksharStrReplace output: 6 \cs{aksharStrReplace} ~ output:\par स्कास्कारकाडडमंळीस्कास्काड 7 \aksharStrReplace { मममडडमंळीममड} { म} { स्का} 8 \ExplSyntaxOff 1 \ExplSyntaxOn 2 \pkg{expl3} ~ output:\par expl3 output: ₃ \tl set:Nn \l tmpa tl { ममंममडडमंळीममड} स्कांममडडमंळीममड 4 \tl_replace_once:Nnn \l_tmpa_tl { मम} { स्का} 5 \tl use:N \l tmpa tl\par \aksharStrReplace* output: 6 \cs{aksharStrReplace*} ~ output:\par ममंस्काडडमंळीममड र \aksharStrReplace* { ममंममडडमंळीममड} { मम} { स्का} 8 \ExplSyntaxOff

\aksharStrRemove \aksharStrRemove* \aksharStrRemove $\{\langle tl \ 1 \rangle\}$ $\{\langle tl \ 2 \rangle\}$

Remove all occurrences of $\langle tl 2 \rangle$ in $\langle tl 1 \rangle$, and leaves the modified $\langle tl 1 \rangle$ in the input stream.

The starred variant will remove only the first occurence of $\langle tl \ 2 \rangle$, all others are left intact.

expl3 output:

डडंळीड

\aksharStrRemove output: डडमंळीड 1 \ExplSyntaxOn

2 \pkg{expl3} ~ output:\par

3 \tl set:Nn \l tmpa tl { मममडडमंळीममड}

4 \tl remove all:Nn \l tmpa tl { ਸ}

5 \tl use:N \l tmpa tl\par

6 \cs{aksharStrRemove} ~ output:\par

7 \aksharStrRemove { मममडडमंळीममड} { म}

8 \ExplSyntaxOff

1 \ExplSyntaxOn

expl3 output: ंममडडमंळीममड

\aksharStrRemove* output: ममंडडमंळीममड 2 \pkg{expl3} ~ output:\par

₃ \tl_set:Nn \l_tmpa_tl { ममंममडडमंळीममड}

4 \tl remove once:Nn \l tmpa tl { मम}

5 \tl use:N \l tmpa tl\par

6 \cs{aksharStrRemove*} ~ output:\par

र \aksharStrRemove∗ { ममंममडडमंळीममड} { मम}

8 \ExplSyntaxOff

expl3 functions 2.2

This section assumes that you have a basic knowledge in LATEX3 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.

\akshar_convert:Nn \akshar_convert:(cn|Nx|cx) \akshar_convert:Nn \langle seq var \rangle \langle \taken list \rangle \rangle

This function converts (token list) to a sequence of characters, that sequence is stored in $\langle \text{seq var} \rangle$. The assignment to $\langle \text{seq var} \rangle$ is local to the current T_FX group.

न, म, स्का, and र

- 1 \ExplSyntaxOn
- 2 \akshar convert:Nn \l tmpa seq { नमस्कार}
- $_{\text{3}} \geq ... \\ \label{eq:seq_use:Nnnn l_tmpa_seq { ~and~ } { ,~ } { ,~and~ }$
- 4 \ExplSyntaxOff

Implementation

- ₁ ⟨@@=akshar⟩
- 2 (*package)

Declare the package. By loading fontspec, xparse, and in turn, expl3, are also

- 3 \RequirePackage{fontspec}
- 4 \ProvidesExplPackage {\aksharPackageName}
- {\aksharPackageDate} {\aksharPackageVersion} {\aksharPackageDescription}

Variable declarations

\c__akshar_joining_tl \c__akshar_diacritics_tl

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

- \c_akshar_joining_tl is the "connecting" character \(\).

\l__akshar_prev_joining_bool

\l__akshar_tmpd_seq

\l__akshar_tmpe_seq

\l__akshar_tmpa_int

\l__akshar_tmpb_int

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.

```
13 \bool_new:N \l__akshar_prev_joining_bool
                      (End definition for \l__akshar_prev_joining_bool.)
                     This local sequence stores the output of the converter.
\l__akshar_char_seq
                       14 \seq_new:N \l__akshar_char_seq
                      (End definition for \l__akshar_char_seq.)
                     Some temporary variables.
 \l__akshar_tmpa_tl
 \l__akshar_tmpb_tl
                       15 \tl_new:N \l__akshar_tmpa_tl
\l__akshar_tmpa_seq
                       16 \tl_new:N \l__akshar_tmpb_tl
\l__akshar_tmpb_seq
                       17 \seq_new:N \l__akshar_tmpa_seq
\l__akshar_tmpc_seq
                       ^{18} \seq_new:N \l__akshar_tmpb_seq
```

19 \seq_new:N \l__akshar_tmpc_seq

20 \seq_new:N \l__akshar_tmpd_seq

21 \seq_new:N \l__akshar_tmpe_seq

22 \int_new:N \l__akshar_tmpa_int
23 \int_new:N \l__akshar_tmpb_int

(End definition for $\l_akshar_tmpa_tl$ and others.)

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.

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.)

```
34 \msg_new:nnnn { akshar } { err_character_out_of_bound }
   { Character ~ index ~ out ~ of ~ bound. }
35
    {
36
      You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~
37
      string ~ #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~
38
      Make ~ sure ~ that ~ you ~ use ~ a ~ number ~ between ~ and ~ not ~
39
      including \sim 0 \sim and \sim #3, \sim so \sim that \sim I \sim can \sim return \sim a \sim
      good ~ output. ~ Proceed ~ and ~ I ~ will ~ return ~
      \token_to_str:N \scan_stop:.
42
    }
43
```

In \aksharStrHead and \aksharStrTail, the string must not be blank.

```
44 \msg_new:nnnn { akshar } { err_string_empty }
    { The ~ input ~ string ~ is ~ empty. }
45
    {
46
      To ~ get ~ the ~ #1 ~ character ~ of ~ a ~ string, ~ that ~ string ~
47
      must ~ not ~ be ~ empty, ~ but ~ the ~ input ~ string ~ is ~ empty.
48
      Make ~ sure ~ the ~ string ~ contains ~ something, ~ or ~ proceed ~
      and ~ I ~ will ~ use ~ \token_to_str:N \scan_stop:.
51
```

3.3 Utilities

\tl_if_in:NoTF

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.

```
52 \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.
                      53 \cs_generate_variant:Nn \seq_set_split:Nnn { Nxx }
                     (End definition for \seq_set_split:Nxx.)
```

Some variants of l3msg functions that we will need when issuing error mes-\msg_error:nnx \msg_error:nnnxx sages.

```
54 \cs_generate_variant:Nn \msg_error:nnn { nnx }
55 \cs_generate_variant:Nn \msg_error:nnnnn { nnnxx }
(End definition for \msg_error:nnx and \msg_error:nnnxx.)
```

__akshar_var_if_global:N*TF* \c__akshar_str_g_tl

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_{α} (name)_seq. \c_akshar_str_seq_tl If it is not a sequence variable, this function will (TODO) issue an error message.

```
56 \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} }
  \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
59
    {
      \bool_if:nTF
        { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
61
62
          \msg_error:nnx { akshar } { err_not_a_sequence_variable }
63
64
            { \token_to_str:N #1 }
          \prg_return_false:
65
        }
66
        {
67
          \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
68
            { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
69
          \seq_get_left:NN \l__akshar_tmpb_seq \l__akshar_tmpa_tl
          \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
          \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
            {
73
              \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
74
                { \prg_return_true: } { \prg_return_false: }
            }
            {
              \msg_error:nnx { akshar } { err_not_a_sequence_variable }
78
                { \token_to_str:N #1 }
              \prg_return_false:
81
        }
82
    }
83
```

(End definition for $_$ akshar_var_if_global:NTF, $_$ akshar_str_g_tl, and $_$ akshar_str_seq_tl.)

__akshar_int_append_ordinal:n Append st, nd, rd or th to interger #1. Will be needed in error messages.

```
84 \cs_new:Npn \__akshar_int_append_ordinal:n #1
  85
                       {
  86
                                       #1
                                        \int_case:nnF { #1 }
  87
  88
                                                  {
                                                               { 11 } { th }
  89
                                                               { 12 } { th }
  90
                                                              { 13 } { th }
  91
                                                               { -11 } { th }
  92
                                                               { -12 } { th }
  93
                                                               { -13 } { th }
  94
  95
                                                  }
  96
                                                   {
                                                               \int \int d^2 r 
  97
                                                                          {
  98
                                                                                       \int_case:nnF { #1 - 10 * (#1 / 10) }
  99
                                                                                                  {
100
                                                                                                             { 1 } { st }
101
                                                                                                             { 2 } { nd }
102
                                                                                                              { 3 } { rd }
103
                                                                                                   } { th }
                                                                          }
                                                                          {
                                                                                      \int_case:nnF { (- #1) - 10 * ((- #1) / 10) }
107
108
                                                                                                             { 1 } { st }
109
                                                                                                             { 2 } { nd }
110
                                                                                                             { 3 } { rd }
                                                                                                   } { th }
                                                                          }
113
                                                  }
114
                           }
```

(End definition for $_$ akshar_int_append_ordinal:n.)

The \akshar_convert:Nn function and its variants

\akshar_convert:Nn \akshar_convert:cn \akshar_convert:Nx \akshar_convert:cx

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

```
\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
119
```

Loop through every token of the input.

```
\tl_map_variable:NNn {#2} \l__akshar_map_tl
121
        {
           \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
```

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_put_right:Nx \l__akshar_char_seq
            { \l_akshar_tmpa_tl \l_akshar_map_tl }
126
         }
         {
128
           \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
129
            {
130
```

In this case, the character is the joining character, Q. 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\langle \l\langle akshar_map_tl \rangle
\langle 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
138
                        \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
139
                        \seq_put_right:Nx \l__akshar_char_seq
                          { \l_akshar_tmpa_tl \l_akshar_map_tl }
141
                        \bool_set_false:N \l__akshar_prev_joining_bool
                      }
                        \seq_put_right:Nx
145
                          \l__akshar_char_seq { \l__akshar_map_tl }
                      }
147
                 }
148
             }
149
         }
150
```

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

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.

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

(End definition for \__akshar_seq_push_seq:NN.)
```

__akshar_replace:NnnnN

If #5 is $c_{\text{false_bool}}$, this function replaces all occurences of #3 in #2 by #4 and stores the output sequence to #1. If #5 is $c_{\text{true_bool}}$, 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 \l_- -akshar_tmpa_int-th item must indeed appear in #3. So we need to compare that with the length of #3.

If now \l__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 \l_-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" \l_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 \l_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.

```
158 \cs_new:Npn \__akshar_replace:NnnnN #1 #2 #3 #4 #5
159
       \akshar_convert:Nn \l__akshar_tmpc_seq {#2}
160
       \akshar_convert:Nn \l__akshar_tmpd_seq {#3}
161
       \akshar_convert:Nn \l__akshar_tmpe_seq {#4}
162
       \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
167
         {
168
           \int_compare:nNnTF { \l__akshar_tmpb_int } > { 0 }
169
             { \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 }
176
                    \seq_clear:N \l__akshar_tmpb_seq
178
                    \__akshar_seq_push_seq:NN
                      \verb|\l_akshar_tmpa_seq \l_akshar_tmpe_seq| \\
179
                    \int_set:Nn \l__akshar_tmpa_int { 1 }
180
                    \tl_set:Nx \l__akshar_tmpa_tl
181
                      { \seq_item:Nn \l__akshar_tmpd_seq { 1 } }
182
                    \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
183
                      {
                        \int_incr:N \l__akshar_tmpa_int
                        \seq_put_right:NV \l__akshar_tmpb_seq \l__akshar_map_tl
                      }
188
                        \seq_put_right:NV \l__akshar_tmpa_seq \l__akshar_map_tl
189
190
191
192
                    \tl_set:Nx \l__akshar_tmpa_tl
193
                        \seq_item:Nn \l__akshar_tmpd_seq { \l__akshar_tmpa_int }
                    \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
197
                     {
198
                        \int_incr:N \l__akshar_tmpa_int
199
                        \seq_put_right:NV \l__akshar_tmpb_seq \l__akshar_map_tl
200
                      }
201
202
                        \int_set:Nn \l__akshar_tmpa_int { 1 }
203
```

```
\__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
                 207
                                           { \seq_item:Nn \l__akshar_tmpd_seq { 1 } }
                 208
                                         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
                 209
                                           {
                 210
                                              \int_incr:N \l__akshar_tmpa_int
                                              \seq_put_right:NV
                                                \l__akshar_tmpb_seq \l__akshar_map_tl
                 214
                                              \seq_put_right:NV
                                                \l__akshar_tmpa_seq \l__akshar_map_tl
                 218
                                       }
                 219
                                  }
                 220
                          }
                        \__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpb_seq
                 224
                        \__akshar_var_if_global:NTF #1
                          { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
                          { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
                 226
                      }
                 (End definition for \__akshar_replace:NnnnN.)
                 3.6 Front-end \text{LAT}_{FX}2_{\varepsilon} macros
                Expands to the length of the string.
\aksharStrLen
                 228 \NewExpandableDocumentCommand \aksharStrLen {m}
                        \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
                 230
                        \seq_count:N \l__akshar_tmpa_seq
                      }
                 (End definition for \aksharStrLen. This function is documented on page 2.)
\aksharStrChar
                Returns the n-th character of the string.
                 233 \NewExpandableDocumentCommand \aksharStrChar {mm}
                 234
                        \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
                        \bool_if:nTF
                 236
                          {
                            \int \int d^2 x dx dx = 0
                 238
                            \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq}</pre>
                 239
                          }
                          { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
                            \msg_error:nnnxx { akshar } { err_character_out_of_bound }
                 243
                              { #1 } { \__akshar_int_append_ordinal:n { #2 } }
                 244
                               { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
                 245
                            \scan_stop:
                 246
                          }
                 247
                (End definition for \aksharStrChar. This function is documented on page 2.)
                Return the first character of the string.
\aksharStrHead
                    \NewExpandableDocumentCommand \aksharStrHead {m}
                        \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
                 251
                        \int_compare:nNnTF { \seq_count:N \l__akshar_tmpa_seq } = {0}
                 253
                            \msg_error:nnn { akshar } { err_character_out_of_bound }
                 254
                              { first }
```

(End definition for \aksharStrHead. This function is documented on page 2.)

\aksharStrTail Return the last character of the string.

```
_{260} \NewExpandableDocumentCommand \aksharStrTail {m}
261
       \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
262
       \int_compare:nNnTF { \seq_count:N \l__akshar_tmpa_seq } = {0}
263
264
         {
           \msg_error:nnn { akshar } { err_character_out_of_bound }
265
             { last }
266
           \scan_stop:
         }
         { \seq_item:Nn \l__akshar_tmpa_seq {\seq_count:N \l__akshar_tmpa_seq} }
269
     }
```

(End definition for \aksharStrTail. This function is documented on page 2.)

\aksharStrReplace \aksharStrReplace*

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

```
\NewExpandableDocumentCommand \aksharStrReplace {smmm}
       \IfBooleanTF {#1}
         {
274
              _akshar_replace:NnnnN \l__akshar_tmpa_seq
             {#2} {#3} {#4} \c_true_bool
276
         }
277
         {
              _akshar_replace:NnnnN \l__akshar_tmpa_seq
280
             {#2} {#3} {#4} \c_false_bool
281
282
       \seq_use:Nn \l__akshar_tmpa_seq {}
283
```

(End definition for $\arrange and \arrange and \arrange$

\aksharStrRemove \aksharStrRemove*

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

```
\NewExpandableDocumentCommand \aksharStrRemove {smm}
285
       \IfBooleanTF {#1}
286
287
              akshar_replace:NnnnN \l_akshar_tmpa_seq
288
             {#2} {#3} {} \c_true_bool
289
         }
290
         {
291
           \__akshar_replace:NnnnN \l__akshar_tmpa_seq
292
             {#2} {#3} {} \c_false_bool
295
       \seq_use:Nn \l__akshar_tmpa_seq {}
     }
296
```

(End definition for \arrangle and \arrangle and \arrangle . These functions are documented on page 3.)

```
297 (/package)
```

Index

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

Α	\bool new:N
akshar commands:	\bool_set_false:N 119, 142
\akshar_convert:Nn 1, 3, 4, 6, 116,	\bool_set_true:N
160, 161, 162, 230, 235, 251, 262	\c_false_bool 7, 280, 293
akshar internal commands:	\c_true_bool 7, 276, 289
\lakshar_char_seq	
7, <u>14</u> , 118, 124, 125,	C
131, 132, 139, 140, 146, 152, 153	<pre>cs commands: \cs_generate_variant:Nn</pre>
\cakshar_diacritics_tl . 3, 6, 122 \akshar_int_append_ordinal:n	
	\cs_new:Npn 84, 116, 156, 158
\cakshar_joining_tl 3, 6, 129	\cs_split_function:N 61, 69
\lakshar_map_tl	
120, 122, 126, 129,	E
133, 141, 146, 167, 170, 183,	exp commands:
186, 189, 197, 200, 209, 213, 217	\exp_last_unbraced:Nf 61, 69
\l_akshar_prev_joining_bool	Ī
7, <u>13</u> , 119, 134, 137, 142	\IfBooleanTF 273, 286
\akshar_replace:NnnnN 158, 275, 279, 288, 292	int commands:
_akshar_seq_push_seq:NN	\int_case:nnTF 87, 99, 107
	\int_compare:nNnTF
\cakshar_str_g_tl <u>56</u>	97, 169, 172, 252, 263
$c_akshar_str_seq_tl \dots 56$	\int_compare_p:nNn 238, 239
$local_loc$	\int_eval:n 245 \int_incr:N 176, 185, 199, 211
173, 180, 185, 195, 199, 203, 211	\int_new:N 176, 163, 133, 211
\l_akshar_tmpa_seq <u>15</u> , 163,	\int_set:Nn 165, 166, 180, 203
179, 189, 205, 217, 223, 225, 226, 230, 231, 235, 239, 241,	, , , ,
245, 251, 252, 258, 262, 263,	M
269, 275, 279, 282, 288, 292, 295	msg commands:
\lakshar_tmpa_tl 15,	\msg_error:nnn
70, 74, 124, 126, 131, 133, 1 39 ,	
141, 181, 183, 193, 197, 207, 209	\msg_error:nnnn <u>54</u> , 55, 243 \msg_new:nnnn 24, 34, 44
\lakshar_tmpb_int	(m3g_new.mim 21, 51, 11
	N
\l_akshar_tmpb_seq 15, 68, 70, 71, 164, 170,	\NewExpandableDocumentCommand
177, 186, 200, 205, 206, 213, 223	228, 233, 249, 260, 271, 284
\lakshar_tmpb_tl 15, 71, 72	Р
\lakshar_tmpc_seq <u>15</u> , 160, 167	prg commands:
<pre>\lakshar_tmpd_seq</pre>	\prg_generate_conditional
<u>15</u> , 161, 173, 182, 195, 208	variant:Nnn 52
\lakshar_tmpe_seq <u>15</u> , 162, 179	\prg_new_conditional:Npnn 58
<pre>\akshar_var_if_global 5 \akshar_var_if_global:NTF</pre>	\prg_return_false: 65, 75, 80
	\prg_return_true:
\aksharPackageDate5	\ProvidesExplPackage 4
\aksharPackageDescription 5	R
$\arrowvert \arrowvert \arrowver$	\RequirePackage 3
\aksharPackageVersion 5	
\aksharStrChar	S
\aksharStrHead	scan commands:
\aksharStrLen	\scan_stop: 42, 50, 246, 256, 267
\aksharStrRemove	seq commands: \seq_clear:N 118, 163, 164, 177, 206
\aksharStrReplace	\seq_cunt:N\
\aksharStrReplace* 2, 271	173, 231, 239, 245, 252, 263, 269
\aksharStrTail 2, 5, <u>260</u>	\seq_get_left:NN 70
	\seq_get_right:NN 71
В	\seq_gset_eq:NN 152, 225
bool commands:	\seq_item:Nn
\bool_if:NTF	182, 195, 208, 241, 258, 269 \seq_map_inline:Nn 157
\b00t_ti;iiif 00, 230	\seq_map_thtthe.NH 15/

\seq_map_variable:NNn 167	\tl_if_in:Nn 52
\seq_new:N 14, 17, 18, 19, 20, 21	\tl_if_in:NnTF <u>52</u> , 122
\seq_pop_right:NN 124, 131, 139	\tl_map_variable:NNn 120
\seq_put_right:Nn	\tl_new:N 15, 10
157, 170, 186, 189, 200, 212, 216	\tl_set:Nn 181, 193, 20
\seq_set_eq:NN 153, 226	\tl_to_str:n 56, 5
\seq_set_split:Nnn 53, 53, 68	token commands:
\seq_use:Nn 282, 295	\token_to_str:N . 42, 50, 64, 68, 79
Т	7.7
tl commands:	U
\tl_const:Nn 6, 7, 56, 57	use commands:
\tl_if_eq:NNTF	\use_i:nnn 69
72, 74, 129, 183, 197, 209	\use_iii:nnn 6