



The akshar package

Vu Van Dung

Version 0.1 — 2020/05/23

 <https://ctan.org/pkg/akshar>
 <https://github.com/jouleev/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	$\LaTeX 2_{\epsilon}$ 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 <code>\akshar_convert:Nn</code> function and its variants	6
3.5	Other internal functions	7
3.6	Front-end $\LaTeX 2_{\epsilon}$ macros	9
	Index	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:

```
1 \ExplSyntaxOn
2 \tl_set:Nn \l_tmpa_tl { की}
3 \tl_count:N \l_tmpa_tl \c_space_token tokens.
4 \ExplSyntaxOff
```

2 tokens.

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 $\text{\LaTeX} 2_{\epsilon}$ macros

\aksharStrLen \aksharStrLen {<token list>}

Return the number of Devanagari characters in the <token list>.

There are 4 characters in नमस्कार.
expl3 returns 7, which is wrong.

```
1 There are \aksharStrLen{ नमस्कार} characters in नमस्कार.\par
2 \ExplSyntaxOn
3 \pkg{expl3}~returns~\tl_count:n { नमस्कार},~which~is~wrong.
4 \ExplSyntaxOff
```

\aksharStrHead \aksharStrHead {<token list>} {<n>}

Return the first character of the token list.

मं 1 \aksharStrHead { मंलीमडमं }

\aksharStrTail \aksharStrTail {<token list>} {<n>}

Return the last character of the token list.

मं 1 \aksharStrTail { लीमडमं }

\aksharStrChar \aksharStrChar {<token list>} {<n>}

Return the n -th character of the token list.

3rd character of नमस्कार is स्का.
It is not स.

```
1 3rd character of नमस्कार is \aksharStrChar{ नमस्कार}{3}.\par
2 \ExplSyntaxOn
3 It~is~not~\tl_item:nn { नमस्कार} {3}.
4 \ExplSyntaxOff
```

\aksharStrReplace \aksharStrReplace {<tl 1>} {<tl 2>} {<tl 3>}

\aksharStrReplace*

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.

expl3 output:

स्कास्कास्काडडस्कांलीस्कास्काड

\aksharStrReplace output:

स्कास्कास्काडडमंलीस्कास्काड

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { ममडडमंलीमड }
4 \tl_replace_all:Nnn \l_tmpa_tl { म } { स्का }
5 \tl_use:N \l_tmpa_tl\par
6 \ExplSyntaxOff
7 \cs{aksharStrReplace} output:\par
8 \aksharStrReplace { ममडडमंलीमड } { म } { स्का }
```

expl3 output:

स्कांमडडमंलीमड

\aksharStrReplace* output:

ममस्काडडमंलीमड

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { ममडडमंलीमड }
4 \tl_replace_once:Nnn \l_tmpa_tl { मम } { स्का }
5 \tl_use:N \l_tmpa_tl\par
6 \ExplSyntaxOff
7 \cs{aksharStrReplace*} output:\par
8 \aksharStrReplace* { ममडडमंलीमड } { मम } { स्का }
```

```
\aksharStrRemove
\aksharStrRemove*
```

```
\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.

```
expl3 output:
  डडंलीड
\aksharStrRemove output:
  डडंलीड
```

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { ममडडडंलीमडड }
4 \tl_remove_all:Nn \l_tmpa_tl { म }
5 \tl_use:N \l_tmpa_tl\par
6 \ExplSyntaxOff
7 \cs{aksharStrRemove} output:\par
8 \aksharStrRemove { ममडडडंलीमडड } { म }
```

```
expl3 output:
  ममडडडंलीमडड
\aksharStrRemove* output:
  मडंडडंलीमडड
```

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { मममडडडंलीमडड }
4 \tl_remove_once:Nn \l_tmpa_tl { मम }
5 \tl_use:N \l_tmpa_tl\par
6 \ExplSyntaxOff
7 \cs{aksharStrRemove*} output:\par
8 \aksharStrRemove* { मममडडडंलीमडड } { मम }
```

2.2 expl3 functions

This section assumes that you have a basic knowledge in \LaTeX 3 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 <seq var> <{token list}>
```

This function converts <token list> to a sequence of characters, that sequence is stored in <seq var>. The assignment to <seq var> is local to the current \TeX group.

```
न, म, स्का, and र
```

```
1 \ExplSyntaxOn
2 \akshar_convert:Nn \l_tmpa_seq { नमस्कार }
3 \seq_use:Nnnn \l_tmpa_seq { ~and~ } { ,~ } { ,~and~ }
4 \ExplSyntaxOff
```

3 Implementation

```
1 <@@=akshar>
2 <*package>
```

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

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

3.1 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 ँ.
- `\c__akshar_diacritics_tl` is the list of all diacritics: ा, ि, ी, ु, ू, े, ै, ो, औ, ं, ः, ऌ, ॡ, ॢ, ॣ, ।, ॥, ०, १, २, ३, ४, ५, ६, ७, ८, ९, अ, उ, क, न, प, र, त्रि, ङ, ँ, ०, १, २, ३, ४, ५, ६, ७, ८, ९, अ, उ, क, न, प, र, त्रि, ङ.

```

6 \tl_const:Nn \c__akshar_joining_tl { ञ }
7 \tl_const:Nn \c__akshar_diacritics_tl
8 {
9   ा, ि, ी, ु, ू, े, ै, ो, ौ, ं, ः, ॄ, ॅ,
10  ॅ, ॊ, ो, ौ, ्, ॎ, ॏ, ॐ, ॐ, ॑, ॒, ॒, ॒, ॒,
11  ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒,
12  ॒, ॒, ॒, ॒, ॒, ॒, ॒, ॒
13 }

```

(End definition for `\c__akshar_joining_tl` and `\c__akshar_diacritics_tl`.)

`\l__akshar_prev_joining_bool`

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.

```
14 \bool_new:N \l__akshar_prev_joining_bool
```

(End definition for `\l__akshar_prev_joining_bool`.)

`\l__akshar_char_seq`

This local sequence stores the output of the converter.

```
15 \seq_new:N \l__akshar_char_seq
```

(End definition for `\l__akshar_char_seq`.)

`\l__akshar_tmpa_tl`
`\l__akshar_tmppb_tl`
`\l__akshar_tmppa_seq`
`\l__akshar_tmppb_seq`
`\l__akshar_tmppc_seq`
`\l__akshar_tmppd_seq`
`\l__akshar_tmpe_seq`
`\l__akshar_tmppa_int`
`\l__akshar_tmppb_int`

Some temporary variables.

```

16 \tl_new:N \l__akshar_tmpa_tl
17 \tl_new:N \l__akshar_tmppb_tl
18 \seq_new:N \l__akshar_tmppa_seq
19 \seq_new:N \l__akshar_tmppb_seq
20 \seq_new:N \l__akshar_tmppc_seq
21 \seq_new:N \l__akshar_tmppd_seq
22 \seq_new:N \l__akshar_tmpe_seq
23 \int_new:N \l__akshar_tmppa_int
24 \int_new:N \l__akshar_tmppb_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.

```

25 \msg_new:nnnn { akshar } { err_not_a_sequence_variable }
26 { #1 ~ is ~ not ~ a ~ valid ~ LaTeX3 ~ sequence ~ variable. }
27 {
28   You ~ have ~ requested ~ me ~ to ~ assign ~ some ~ value ~ to ~
29   the ~ control ~ sequence ~ #1, ~ but ~ it ~ is ~ not ~ a ~ valid ~
30   sequence ~ variable. ~ Read ~ the ~ documentation ~ of ~ expl3 ~
31   for ~ more ~ information. ~ Proceed ~ and ~ I ~ will ~ pretend ~
32   that ~ #1 ~ is ~ a ~ local ~ sequence ~ variable ~ (beware ~ that ~
33   unexpected ~ behaviours ~ may ~ occur).
34 }

```

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

```

35 \msg_new:nnnn { akshar } { err_character_out_of_bound }
36 { Character ~ index ~ out ~ of ~ bound. }
37 {
38   You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~
39   string ~ #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~
40   Make ~ sure ~ that ~ you ~ use ~ a ~ number ~ between ~ and ~ not ~
41   including ~ 0 ~ and ~ #3, ~ so ~ that ~ I ~ can ~ return ~ a ~
42   good ~ output. ~ Proceed ~ and ~ I ~ will ~ return ~
43   \token_to_str:N \scan_stop:.
44 }

```

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

```

45 \msg_new:nnnn { akshar } { err_string_empty }
46 { The ~ input ~ string ~ is ~ empty. }
47 {
48   To ~ get ~ the ~ #1 ~ character ~ of ~ a ~ string, ~ that ~ string ~
49   must ~ not ~ be ~ empty, ~ but ~ the ~ input ~ string ~ is ~ empty.
50   Make ~ sure ~ the ~ string ~ contains ~ something, ~ or ~ proceed ~
51   and ~ I ~ will ~ use ~ \token_to_str:N \scan_stop:.
52 }

```

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.

```

53 \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`.

```

54 \cs_generate_variant:Nn \seq_set_split:Nnn { Nxx }

```

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

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

```

55 \cs_generate_variant:Nn \msg_error:nnn { nnx }
56 \cs_generate_variant:Nn \msg_error:nnnnn { nnnxx }

```

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

`__akshar_var_if_global:NTF` 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`.
`\c__akshar_str_g_tl` If it is not a sequence variable, this function will (TODO) issue an error message.
`\c__akshar_str_seq_tl`

```

57 \tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
58 \tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
59 \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
60 {
61   \bool_if:nTF
62     { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
63     {
64       \msg_error:nnx { akshar } { err_not_a_sequence_variable }
65       { \token_to_str:N #1 }
66       \prg_return_false:
67     }
68     {
69       \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
70       { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
71       \seq_get_left:NN \l__akshar_tmpb_seq \l__akshar_tmpa_tl
72       \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
73       \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
74       {
75         \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
76         { \prg_return_true: } { \prg_return_false: }
77       }
78       {
79         \msg_error:nnx { akshar } { err_not_a_sequence_variable }
80         { \token_to_str:N #1 }
81         \prg_return_false:
82       }
83     }
84 }

```

(End definition for `__akshar_var_if_global:NTF`, `\c__akshar_str_g_tl`, and `\c__akshar_str_seq_tl`.)

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

```

85 \cs_new:Npn \__akshar_int_append_ordinal:n #1
86 {
87   #1
88   \int_case:nnF { #1 }
89   {
90     { 11 } { th }
91     { 12 } { th }
92     { 13 } { th }
93     { -11 } { th }
94     { -12 } { th }
95     { -13 } { th }
96   }
97   {
98     \int_compare:nNnTF { #1 } > { -1 }
99     {
100       \int_case:nnF { #1 - 10 * ( #1 / 10 ) }
101       {
102         { 1 } { st }
103         { 2 } { nd }
104         { 3 } { rd }
105       } { th }
106     }
107     {
108       \int_case:nnF { ( - #1 ) - 10 * ( ( - #1 ) / 10 ) }
109       {
110         { 1 } { st }
111         { 2 } { nd }
112         { 3 } { rd }
113       } { th }
114     }
115   }
116 }

```

(End definition for `__akshar_int_append_ordinal:n`.)

3.4 The `\akshar_convert:Nn` function and its variants

`\akshar_convert:Nn` 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.
`\akshar_convert:cn`
`\akshar_convert:Nx`
`\akshar_convert:cx`

```

117 \cs_new:Npn \akshar_convert:Nn #1 #2
118 {

```

Clear anything stored in advance. We don't want different calls of the function to conflict with each other.

```

119   \seq_clear:N \l__akshar_char_seq
120   \bool_set_false:N \l__akshar_prev_joining_bool

```

Loop through every token of the input.

```

121   \tl_map_variable:NNn {#2} \l__akshar_map_tl
122   {
123     \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
124     {

```

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.

```

125       \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
126       \seq_put_right:Nx \l__akshar_char_seq
127       { \l__akshar_tmpa_tl \l__akshar_map_tl }
128     }
129   {
130     \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
131     {

```

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.

```

132         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
133         \seq_put_right:Nx \l__akshar_char_seq
134         { \l__akshar_tmpa_tl \l__akshar_map_tl }
135         \bool_set_true:N \l__akshar_prev_joining_bool
136     }
137 {

```

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

```

138         \bool_if:NTF \l__akshar_prev_joining_bool
139         {
140             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
141             \seq_put_right:Nx \l__akshar_char_seq
142             { \l__akshar_tmpa_tl \l__akshar_map_tl }
143             \bool_set_false:N \l__akshar_prev_joining_bool
144         }
145         {
146             \seq_put_right:Nx
147             \l__akshar_char_seq { \l__akshar_map_tl }
148         }
149     }
150 }
151 }

```

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

```

152     \__akshar_var_if_global:NTF #1
153     { \seq_gset_eq:NN #1 \l__akshar_char_seq }
154     { \seq_set_eq:NN #1 \l__akshar_char_seq }
155 }

```

Generate variants that might be helpful for some.

```

156 \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.

```

157 \cs_new:Npn \__akshar_seq_push_seq:NN #1 #2
158 { \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_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 `\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 $\backslash_akshar_tmpa_int$ -th character of #3.
 - If they are the same, we still have a chance that it will match, so we increase the “iterator” $\backslash_akshar_tmpa_int$ 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 $\backslash_akshar_tmpa_int$ 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.

```

159 \cs_new:Npn \__akshar_replace:NnnN #1 #2 #3 #4 #5
160 {
161   \akshar_convert:Nn \__akshar_tmpe_seq {#2}
162   \akshar_convert:Nn \__akshar_tmpe_seq {#3}
163   \akshar_convert:Nn \__akshar_tmpe_seq {#4}
164   \seq_clear:N \__akshar_tmpe_seq
165   \seq_clear:N \__akshar_tmpe_seq
166   \int_set:Nn \__akshar_tmpe_int { 1 }
167   \int_set:Nn \__akshar_tmpe_int { 0 }
168   \seq_map_variable:Nnn \__akshar_tmpe_seq \__akshar_map_tl
169   {
170     \int_compare:nNnTF { \__akshar_tmpe_int } > { 0 }
171     { \seq_put_right:NV \__akshar_tmpe_seq \__akshar_map_tl }
172     {
173       \int_compare:nNnTF
174       { \__akshar_tmpe_int } = { 1 + \seq_count:N \__akshar_tmpe_seq }
175       {
176         \bool_if:NT {#5}
177         { \int_incr:N \__akshar_tmpe_int }
178         \seq_clear:N \__akshar_tmpe_seq
179         \__akshar_seq_push_seq:NN
180         \__akshar_tmpe_seq \__akshar_tmpe_seq
181         \int_set:Nn \__akshar_tmpe_int { 1 }
182         \tl_set:Nx \__akshar_tmpe_tl
183         { \seq_item:Nn \__akshar_tmpe_seq { 1 } }
184         \tl_if_eq:NNTF \__akshar_map_tl \__akshar_tmpe_tl
185         {
186           \int_incr:N \__akshar_tmpe_int
187           \seq_put_right:NV \__akshar_tmpe_seq \__akshar_map_tl
188         }
189         {
190           \seq_put_right:NV \__akshar_tmpe_seq \__akshar_map_tl
191         }
192       }
193     }
194     \tl_set:Nx \__akshar_tmpe_tl
195     {
196       \seq_item:Nn \__akshar_tmpe_seq { \__akshar_tmpe_int }
197     }
198     \tl_if_eq:NNTF \__akshar_map_tl \__akshar_tmpe_tl
199     {
200       \int_incr:N \__akshar_tmpe_int
201       \seq_put_right:NV \__akshar_tmpe_seq \__akshar_map_tl
202     }
203     {
204       \int_set:Nn \__akshar_tmpe_int { 1 }

```



```

205         \__akshar_seq_push_seq:NN
206         \l__akshar_tmpa_seq \l__akshar_tmpb_seq
207         \seq_clear:N \l__akshar_tmpb_seq
208         \tl_set:Nx \l__akshar_tmpa_tl
209         { \seq_item:Nn \l__akshar_tmpd_seq { 1 } }
210         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
211         {
212             \int_incr:N \l__akshar_tmpa_int
213             \seq_put_right:NV
214                 \l__akshar_tmpb_seq \l__akshar_map_tl
215         }
216         {
217             \seq_put_right:NV
218                 \l__akshar_tmpa_seq \l__akshar_map_tl
219         }
220     }
221 }
222 }
223 }
224 \__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpb_seq
225 \__akshar_var_if_global:NTF #1
226 { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
227 { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
228 }

```

(End definition for `__akshar_replace:NnnnN`.)

3.6 Front-end $\text{\LaTeX}2_{\epsilon}$ macros

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

```

229 \NewExpandableDocumentCommand \aksharStrLen {m}
230 {
231     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
232     \seq_count:N \l__akshar_tmpa_seq
233 }

```

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

`\aksharStrChar` Returns the n -th character of the string.

```

234 \NewExpandableDocumentCommand \aksharStrChar {mm}
235 {
236     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
237     \bool_if:nTF
238     {
239         \int_compare_p:nNn {#2} > {0} &&
240         \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq}
241     }
242     { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
243     {
244         \msg_error:nnnxx { akshar } { err_character_out_of_bound }
245         { #1 } { \__akshar_int_append_ordinal:n { #2 } }
246         { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
247         \scan_stop:
248     }
249 }

```

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

`\aksharStrHead` Return the first character of the string.

```

250 \NewExpandableDocumentCommand \aksharStrHead {m}
251 {
252     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
253     \int_compare:nNnTF { \seq_count:N \l__akshar_tmpa_seq } = {0}
254     {
255         \msg_error:nnn { akshar } { err_character_out_of_bound }
256         { first }

```

```

257     \scan_stop:
258   }
259   { \seq_item:Nn \l__akshar_tmpa_seq { 1 } }
260 }

```

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

`\aksharStrTail` Return the last character of the string.

```

261 \NewExpandableDocumentCommand \aksharStrTail {m}
262 {
263   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
264   \int_compare:nNnTF { \seq_count:N \l__akshar_tmpa_seq } = {0}
265   {
266     \msg_error:nnn { akshar } { err_character_out_of_bound }
267     { last }
268     \scan_stop:
269   }
270   { \seq_item:Nn \l__akshar_tmpa_seq { \seq_count:N \l__akshar_tmpa_seq } }
271 }

```

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

`\aksharStrReplace` Replace occurrences of #3 of a string #2 with another string #4.
`\aksharStrReplace*`

```

272 \NewExpandableDocumentCommand \aksharStrReplace {smmm}
273 {
274   \IfBooleanTF {#1}
275   {
276     \__akshar_replace:NnnN \l__akshar_tmpa_seq
277     {#2} {#3} {#4} \c_true_bool
278   }
279   {
280     \__akshar_replace:NnnN \l__akshar_tmpa_seq
281     {#2} {#3} {#4} \c_false_bool
282   }
283   \seq_use:Nn \l__akshar_tmpa_seq {}
284 }

```

(End definition for `\aksharStrReplace` and `\aksharStrReplace*`. These functions are documented on page 2.)

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

```

285 \NewExpandableDocumentCommand \aksharStrRemove {smm}
286 {
287   \IfBooleanTF {#1}
288   {
289     \__akshar_replace:NnnN \l__akshar_tmpa_seq
290     {#2} {#3} {} \c_true_bool
291   }
292   {
293     \__akshar_replace:NnnN \l__akshar_tmpa_seq
294     {#2} {#3} {} \c_false_bool
295   }
296   \seq_use:Nn \l__akshar_tmpa_seq {}
297 }

```

(End definition for `\aksharStrRemove` and `\aksharStrRemove*`. These functions are documented on page 3.)

```

298 </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	<u>1</u> , <u>3</u> , <u>4</u> , <u>6</u> , <u>117</u> , <u>161</u> , <u>162</u> , <u>163</u> , <u>231</u> , <u>236</u> , <u>252</u> , <u>263</u>
akshar internal commands:	
\l__akshar_char_seq <u>7</u> , <u>15</u> , <u>119</u> , <u>125</u> , <u>126</u> , <u>132</u> , <u>133</u> , <u>140</u> , <u>141</u> , <u>147</u> , <u>153</u> , <u>154</u>
\c__akshar_diacritics_tl	<u>3</u> , <u>6</u> , <u>123</u>
__akshar_int_append_ordinal:n <u>85</u> , <u>245</u>
\c__akshar_joining_tl <u>3</u> , <u>6</u> , <u>130</u>
\l__akshar_map_tl <u>121</u> , <u>123</u> , <u>127</u> , <u>130</u> , <u>134</u> , <u>142</u> , <u>147</u> , <u>168</u> , <u>171</u> , <u>184</u> , <u>187</u> , <u>190</u> , <u>198</u> , <u>201</u> , <u>210</u> , <u>214</u> , <u>218</u>
\l__akshar_prev_joining_bool <u>7</u> , <u>14</u> , <u>120</u> , <u>135</u> , <u>138</u> , <u>143</u>
__akshar_replace:NnnnN <u>159</u> , <u>276</u> , <u>280</u> , <u>289</u> , <u>293</u>
__akshar_seq_push_seq:NN <u>157</u> , <u>179</u> , <u>205</u> , <u>224</u>
\c__akshar_str_g_tl <u>57</u>
\c__akshar_str_seq_tl <u>57</u>
\l__akshar_tmpa_int	<u>7</u> , <u>8</u> , <u>16</u> , <u>166</u> , <u>174</u> , <u>181</u> , <u>186</u> , <u>196</u> , <u>200</u> , <u>204</u> , <u>212</u>
\l__akshar_tmpa_seq <u>16</u> , <u>164</u> , <u>180</u> , <u>190</u> , <u>206</u> , <u>218</u> , <u>224</u> , <u>226</u> , <u>227</u> , <u>231</u> , <u>232</u> , <u>236</u> , <u>240</u> , <u>242</u> , <u>246</u> , <u>252</u> , <u>253</u> , <u>259</u> , <u>263</u> , <u>264</u> , <u>270</u> , <u>276</u> , <u>280</u> , <u>283</u> , <u>289</u> , <u>293</u> , <u>296</u>
\l__akshar_tmpa_tl <u>16</u> , <u>71</u> , <u>75</u> , <u>125</u> , <u>127</u> , <u>132</u> , <u>134</u> , <u>140</u> , <u>142</u> , <u>182</u> , <u>184</u> , <u>194</u> , <u>198</u> , <u>208</u> , <u>210</u>
\l__akshar_tmpb_int <u>16</u> , <u>167</u> , <u>170</u> , <u>177</u>
\l__akshar_tmpb_seq <u>16</u> , <u>69</u> , <u>71</u> , <u>72</u> , <u>165</u> , <u>171</u> , <u>178</u> , <u>187</u> , <u>201</u> , <u>206</u> , <u>207</u> , <u>214</u> , <u>224</u>
\l__akshar_tmpb_tl <u>16</u> , <u>72</u> , <u>73</u>
\l__akshar_tmpe_seq	... <u>16</u> , <u>161</u> , <u>168</u>
\l__akshar_tmpe_seq <u>16</u> , <u>162</u> , <u>174</u> , <u>183</u> , <u>196</u> , <u>209</u>
\l__akshar_tmpe_seq	... <u>16</u> , <u>163</u> , <u>180</u>
__akshar_var_if_global <u>5</u>
__akshar_var_if_global:Ntf <u>57</u> , <u>152</u> , <u>225</u>
\aksharPackageDate <u>5</u>
\aksharPackageDescription <u>5</u>
\aksharPackageName <u>4</u>
\aksharPackageVersion <u>5</u>
\aksharStrChar <u>2</u> , <u>4</u> , <u>234</u>
\aksharStrHead <u>2</u> , <u>5</u> , <u>250</u>
\aksharStrLen <u>2</u> , <u>229</u>
\aksharStrRemove <u>3</u> , <u>285</u>
\aksharStrRemove* <u>3</u> , <u>285</u>
\aksharStrReplace <u>2</u> , <u>10</u> , <u>272</u>
\aksharStrReplace* <u>2</u> , <u>272</u>
\aksharStrTail <u>2</u> , <u>5</u> , <u>261</u>
B	
bool commands:	
\bool_if:Ntf <u>138</u> , <u>176</u>
\bool_if:nTF <u>61</u> , <u>237</u>
\bool_new:N <u>14</u>
\bool_set_false:N <u>120</u> , <u>143</u>
\bool_set_true:N <u>135</u>
\c_false_bool <u>7</u> , <u>281</u> , <u>294</u>
\c_true_bool <u>7</u> , <u>277</u> , <u>290</u>
C	
cs commands:	
\cs_generate_variant:Nn <u>54</u> , <u>55</u> , <u>56</u> , <u>156</u>
\cs_new:Npn <u>85</u> , <u>117</u> , <u>157</u> , <u>159</u>
\cs_split_function:N <u>62</u> , <u>70</u>
E	
exp commands:	
\exp_last_unbraced:Nf <u>62</u> , <u>70</u>
I	
IfBooleanTF <u>274</u> , <u>287</u>
int commands:	
\int_case:nnTF <u>88</u> , <u>100</u> , <u>108</u>
\int_compare:nNnTF <u>98</u> , <u>170</u> , <u>173</u> , <u>253</u> , <u>264</u>
\int_compare_p:nNn <u>239</u> , <u>240</u>
\int_eval:n <u>246</u>
\int_incr:N <u>177</u> , <u>186</u> , <u>200</u> , <u>212</u>
\int_new:N <u>23</u> , <u>24</u>
\int_set:Nn <u>166</u> , <u>167</u> , <u>181</u> , <u>204</u>
M	
msg commands:	
\msg_error:nnn <u>55</u> , <u>55</u> , <u>64</u> , <u>79</u> , <u>255</u> , <u>266</u>
\msg_error:nnnnn <u>55</u> , <u>56</u> , <u>244</u>
\msg_new:nnnn <u>25</u> , <u>35</u> , <u>45</u>
N	
\NewExpandableDocumentCommand <u>229</u> , <u>234</u> , <u>250</u> , <u>261</u> , <u>272</u> , <u>285</u>
P	
prg commands:	
\prg_generate_conditional_	- variant:Nnn <u>53</u>
\prg_new_conditional:Npnn <u>59</u>
\prg_return_false: <u>66</u> , <u>76</u> , <u>81</u>
\prg_return_true: <u>76</u>
\ProvidesExplPackage <u>4</u>
R	
\RequirePackage <u>3</u>
S	
scan commands:	
\scan_stop:	.. <u>43</u> , <u>51</u> , <u>247</u> , <u>257</u> , <u>268</u>
seq commands:	
\seq_clear:N	<u>119</u> , <u>164</u> , <u>165</u> , <u>178</u> , <u>207</u>
\seq_count:N <u>174</u> , <u>232</u> , <u>240</u> , <u>246</u> , <u>253</u> , <u>264</u> , <u>270</u>
\seq_get_left:NN <u>71</u>
\seq_get_right:NN <u>72</u>
\seq_gset_eq:NN <u>153</u> , <u>226</u>
\seq_item:Nn <u>183</u> , <u>196</u> , <u>209</u> , <u>242</u> , <u>259</u> , <u>270</u>
\seq_map_inline:Nn <u>158</u>

<code>\seq_map_variable:NNn</code>	168	<code>\tl_if_in:Nn</code>	53
<code>\seq_new:N</code> ..	15, 18, 19, 20, 21, 22	<code>\tl_if_in:NnTF</code>	53, 123
<code>\seq_pop_right:NN</code>	125, 132, 140	<code>\tl_map_variable:NNn</code>	121
<code>\seq_put_right:Nn</code>		<code>\tl_new:N</code>	16, 17
.....	126, 133, 141, 146,	<code>\tl_set:Nn</code>	182, 194, 208
.....	158, 171, 187, 190, 201, 213, 217	<code>\tl_to_str:n</code>	57, 58
<code>\seq_set_eq:NN</code>	154, 227	token commands:	
<code>\seq_set_split:Nnn</code>	54, 54, 69	<code>\token_to_str:N</code> .	43, 51, 65, 69, 80
<code>\seq_use:Nn</code>	283, 296		
T		U	
tl commands:		use commands:	
<code>\tl_const:Nn</code>	6, 7, 57, 58	<code>\use_i:nnn</code>	70
<code>\tl_if_eq:NNTF</code>		<code>\use_iii:nnn</code>	62
.....	73, 75, 130, 184, 198, 210		