



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 string of South Asian script. Currently supported scripts: Bengali, Gujarati, Gurmukhi, Kannada, Oriya, Malayalam, Sinhala, Tamil and Telugu.

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	12

1 Introduction

When dealing with processing strings in the South Asian scripts, 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 South Asian scripts, to a sequence of token lists. each of these token lists is a “true” South Asian language 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

\backslash aksharStrLen	\backslash aksharStrLen {<token list>}
---------------------------	--

Return the number of South Asian characters in the <token list>.

There are 7 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
```

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

Return the first character of the token list.

म 1 \aksharStrHead { मंलीमड }

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

Return the last character of the token list.

ं 1 \aksharStrTail { लीमडमं }

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

Return the n -th character of the token list.

3rd character of नमस्कार is स.
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
```

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

\backslash 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

\backslash aksharStrReplace output:

स्कास्कास्काडडस्कांलीस्कास्काड

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { ममडडडमंलीमड }
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 { ममडडडमंलीमड } { म } { स्का }
```

expl3 output:

स्कांमडडडमंलीमड

\backslash aksharStrReplace* output:

स्कांमडडडमंलीमड

```
1 \ExplSyntaxOn
2 \pkg{expl3} ~ output:\par
3 \tl_set:Nn \l_tmpa_tl { ममंमडडडमंलीमड }
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* { ममंमडडडमंलीमड } { मम } { स्का }
```

\aksharStrRemove	\aksharStrRemove {<tl 1>} {<tl 2>}
\aksharStrRemove*	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.
<div style="margin-left: 40px;">expl3 output:</div> <div style="margin-left: 40px;">डंडंकीड</div> <div style="margin-left: 40px;">\aksharStrRemove output:</div> <div style="margin-left: 40px;">डंडंकीड</div>	<pre> 1 \ExplSyntaxOn 2 \pkg{expl3} ~ output:\par 3 \tl_set:Nn \l_tmpa_tl { मममडडमंकीममड} 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 { मममडडमंकीममड} { म} </pre>
<div style="margin-left: 40px;">expl3 output:</div> <div style="margin-left: 40px;">ंममडडमंकीममड</div> <div style="margin-left: 40px;">\aksharStrRemove* output:</div> <div style="margin-left: 40px;">ंममडडमंकीममड</div>	<pre> 1 \ExplSyntaxOn 2 \pkg{expl3} ~ output:\par 3 \tl_set:Nn \l_tmpa_tl { मंममडडमंकीममड} 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* { मंममडडमंकीममड} { मम} </pre>

2.2 expl3 functions

This section assumes that you have a basic knowledge in L^AT_EX3 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.

<code>\akshar_convert:Nn</code>	<code>\akshar_convert:Nn <seq var> {<token list>}</code>
<code>\akshar_convert:(cn Nx cx)</code>	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.
<div> <div>न, म, स, ्र, क, ा, and र</div> <div> <pre> 1 \ExplSyntaxOn 2 \akshar_convert:Nn \l_tmpa_seq { नमस्कार} 3 \seq_use:Nnnn \l_tmpa_seq { ~and~ } { ,~ } { ,~and~ } 4 \ExplSyntaxOff </pre> </div> </div>	

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

[illegible]

[illegible]

```
\l akshar prev joining bool
```

```
23 \bool_new:N \l__akshar_prev_joining_bool
```

\l__akshar_char_seq

```
24 \seq_new:N \l__akshar_char_seq
```

```
\l__akshar_tmpa_tl  
\l__akshar_tmptb_tl  
\l__akshar_tmpa_seq  
\l__akshar_tmptb_seq  
\l__akshar_tmptc_seq  
\l__akshar_tmptd_seq  
\l__akshar_tmpte_seq  
\l__akshar_tmpa_int  
\l akshar tmptb int
```

```

25 \tl_new:N \l__akshar_tmpa_tl
26 \tl_new:N \l__akshar_tmpb_tl
27 \seq_new:N \l__akshar_tmpa_seq
28 \seq_new:N \l__akshar_tmpb_seq
29 \seq_new:N \l__akshar_tmpc_seq
30 \seq_new:N \l__akshar_tmpd_seq
31 \seq_new:N \l__akshar_tmpe_seq
32 \int_new:N \l__akshar_tmpp_int
33 \int_new:N \l__akshar_tmppb_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.

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.)

```

45 { Character ~ index ~ out ~ of ~ bound. }
46 {
47   You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~
48   string ~ #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~
49   Make ~ sure ~ that ~ you ~ use ~ a ~ number ~ between ~ and ~ not ~
50   including ~ 0 ~ and ~ #3, ~ so ~ that ~ I ~ can ~ return ~ a ~
51   good ~ output. ~ Proceed ~ and ~ I ~ will ~ return ~
52   \token_to_str:N \scan_stop:.
53 }

```

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

```

54 \msg_new:nnnn { akshar } { err_string_empty }
55 { The ~ input ~ string ~ is ~ empty. }
56 {
57   To ~ get ~ the ~ #1 ~ character ~ of ~ a ~ string, ~ that ~ string ~
58   must ~ not ~ be ~ empty, ~ but ~ the ~ input ~ string ~ is ~ empty.
59   Make ~ sure ~ the ~ string ~ contains ~ something, ~ or ~ proceed ~
60   and ~ I ~ will ~ use ~ \token_to_str:N \scan_stop:.
61 }

```

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.

```

62 \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.

```

63 \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`

```

64 \cs_generate_variant:Nn \msg_error:nnn { nnx }
65 \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`

```

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

```

```

85         { \prg_return_true: } { \prg_return_false: }
86     }
87     {
88         \msg_error:nnx { akshar } { err_not_a_sequence_variable }
89         { \token_to_str:N #1 }
90         \prg_return_false:
91     }
92 }
93 }

```

(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.

```

94 \cs_new:Npn \__akshar_int_append_ordinal:n #1
95 {
96     #1
97     \int_case:nnF { #1 }
98     {
99         { 11 } { th }
100        { 12 } { th }
101        { 13 } { th }
102        { -11 } { th }
103        { -12 } { th }
104        { -13 } { th }
105    }
106    {
107        \int_compare:nNnTF { #1 } > { -1 }
108        {
109            \int_case:nnF { #1 - 10 * ( #1 / 10 ) }
110            {
111                { 1 } { st }
112                { 2 } { nd }
113                { 3 } { rd }
114            } { th }
115        }
116        {
117            \int_case:nnF { ( - #1 ) - 10 * ( ( - #1 ) / 10 ) }
118            {
119                { 1 } { st }
120                { 2 } { nd }
121                { 3 } { rd }
122            } { th }
123        }
124    }
125 }

```

(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 South Asian 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
126 \cs_new:Npn \akshar_convert:Nn #1 #2
127 {

```

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

```

128     \seq_clear:N \l__akshar_char_seq
129     \bool_set_false:N \l__akshar_prev_joining_bool

```

Loop through every token of the input.

```

130     \tl_map_variable:NNn {#2} \l__akshar_map_tl
131     {
132         \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
133         {

```

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.

```

134         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
135         \seq_put_right:Nx \l__akshar_char_seq
136         { \l__akshar_tmpa_tl \l__akshar_map_tl }
137     }
138     {
139         \tl_if_in:NoTF \c__akshar_joining_tl {\l__akshar_map_tl}
140         {

```

In this case, the character is the joining character, ٠. 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.

```

141         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
142         \seq_put_right:Nx \l__akshar_char_seq
143         { \l__akshar_tmpa_tl \l__akshar_map_tl }
144         \bool_set_true:N \l__akshar_prev_joining_bool
145     }
146     {

```

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

```

147         \bool_if:NTF \l__akshar_prev_joining_bool
148         {
149             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
150             \seq_put_right:Nx \l__akshar_char_seq
151             { \l__akshar_tmpa_tl \l__akshar_map_tl }
152             \bool_set_false:N \l__akshar_prev_joining_bool
153         }
154         {
155             \seq_put_right:Nx
156             \l__akshar_char_seq { \l__akshar_map_tl }
157         }
158     }
159 }
160 }

```

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

```

161     \__akshar_var_if_global:NTF #1
162     { \seq_gset_eq:NN #1 \l__akshar_char_seq }
163     { \seq_set_eq:NN #1 \l__akshar_char_seq }
164 }

```

Generate variants that might be helpful for some.

```

165 \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.

```

166 \cs_new:Npn \__akshar_seq_push_seq:NN #1 #2
167 { \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 `\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.

```

168 \cs_new:Npn \__akshar_replace:NnnN #1 #2 #3 #4 #5
169 {
170   \akshar_convert:Nn \l__akshar_tmpe_seq {#2}
171   \akshar_convert:Nn \l__akshar_tmpe_seq {#3}
172   \akshar_convert:Nn \l__akshar_tmpe_seq {#4}
173   \seq_clear:N \l__akshar_tmpe_seq
174   \seq_clear:N \l__akshar_tmpe_seq
175   \int_set:Nn \l__akshar_tmpe_int { 1 }
176   \int_set:Nn \l__akshar_tmpe_int { 0 }
177   \seq_map_variable:Nnn \l__akshar_tmpe_seq \l__akshar_map_tl
178   {
179     \int_compare:nNnTF { \l__akshar_tmpe_int } > { 0 }
180     { \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl }
181     {
182       \int_compare:nNnTF
183       { \l__akshar_tmpe_int } = { 1 + \seq_count:N \l__akshar_tmpe_seq }
184       {
185         \bool_if:NT {#5}
186         { \int_incr:N \l__akshar_tmpe_int }
187         \seq_clear:N \l__akshar_tmpe_seq
188         \__akshar_seq_push_seq:NN
189         \l__akshar_tmpe_seq \l__akshar_tmpe_seq
190         \int_set:Nn \l__akshar_tmpe_int { 1 }
191         \tl_set:Nx \l__akshar_tmpe_tl
192         { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
193         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
194         {
195           \int_incr:N \l__akshar_tmpe_int
196           \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
197         }
198         {
199           \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
200         }
201       }

```



```

202 {
203   \tl_set:Nx \l__akshar_tmpa_tl
204   {
205     \seq_item:Nn \l__akshar_tmpd_seq { \l__akshar_tmpa_int }
206   }
207   \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
208   {
209     \int_incr:N \l__akshar_tmpa_int
210     \seq_put_right:NV \l__akshar_tmpb_seq \l__akshar_map_tl
211   }
212   {
213     \int_set:Nn \l__akshar_tmpa_int { 1 }
214     \__akshar_seq_push_seq:NN
215       \l__akshar_tmpa_seq \l__akshar_tmpb_seq
216     \seq_clear:N \l__akshar_tmpb_seq
217     \tl_set:Nx \l__akshar_tmpa_tl
218       { \seq_item:Nn \l__akshar_tmpd_seq { 1 } }
219     \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpa_tl
220     {
221       \int_incr:N \l__akshar_tmpa_int
222       \seq_put_right:NV
223         \l__akshar_tmpb_seq \l__akshar_map_tl
224     }
225     {
226       \seq_put_right:NV
227         \l__akshar_tmpa_seq \l__akshar_map_tl
228     }
229   }
230 }
231 }
232 }
233 \__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpb_seq
234 \__akshar_var_if_global:NTF #1
235 { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
236 { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
237 }

```

(End definition for `__akshar_replace:NnnnN`.)

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

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

```

238 \NewExpandableDocumentCommand \aksharStrLen {m}
239 {
240   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
241   \seq_count:N \l__akshar_tmpa_seq
242 }

```

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

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

```

243 \NewExpandableDocumentCommand \aksharStrChar {mm}
244 {
245   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
246   \bool_if:nTF
247   {
248     \int_compare_p:nNn {#2} > {0} &&
249     \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq}
250   }
251   { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
252   {
253     \msg_error:nnnxx { akshar } { err_character_out_of_bound }
254       { #1 } { \__akshar_int_append_ordinal:n { #2 } }
255     { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
256     \scan_stop:
257   }
258 }

```

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

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

```
259 \NewExpandableDocumentCommand \aksharStrHead {m}
260 {
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     { first }
266     \scan_stop:
267   }
268   { \seq_item:Nn \l__akshar_tmpa_seq { 1 } }
269 }
```

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

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

```
270 \NewExpandableDocumentCommand \aksharStrTail {m}
271 {
272   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
273   \int_compare:nNnTF { \seq_count:N \l__akshar_tmpa_seq } = {0}
274   {
275     \msg_error:nnn { akshar } { err_character_out_of_bound }
276     { last }
277     \scan_stop:
278   }
279   { \seq_item:Nn \l__akshar_tmpa_seq { \seq_count:N \l__akshar_tmpa_seq } }
280 }
```

(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*`

```
281 \NewExpandableDocumentCommand \aksharStrReplace {smmm}
282 {
283   \IfBooleanTF {#1}
284   {
285     \__akshar_replace:NnnnN \l__akshar_tmpa_seq
286     {#2} {#3} {#4} \c_true_bool
287   }
288   {
289     \__akshar_replace:NnnnN \l__akshar_tmpa_seq
290     {#2} {#3} {#4} \c_false_bool
291   }
292   \seq_use:Nn \l__akshar_tmpa_seq {}
293 }
```

(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*`

```
294 \NewExpandableDocumentCommand \aksharStrRemove {smm}
295 {
296   \IfBooleanTF {#1}
297   {
298     \__akshar_replace:NnnnN \l__akshar_tmpa_seq
299     {#2} {#3} {} \c_true_bool
300   }
301   {
302     \__akshar_replace:NnnnN \l__akshar_tmpa_seq
303     {#2} {#3} {} \c_false_bool
304   }
305   \seq_use:Nn \l__akshar_tmpa_seq {}
306 }
```

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

307 `\endpackage`

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>126</u> , <u>170</u> , <u>171</u> , <u>172</u> , <u>240</u> , <u>245</u> , <u>261</u> , <u>272</u>
akshar internal commands:	
\l__akshar_char_seq <u>7</u> , <u>24</u> , <u>128</u> , <u>134</u> , <u>135</u> , <u>141</u> , <u>142</u> , <u>149</u> , <u>150</u> , <u>156</u> , <u>162</u> , <u>163</u>
\c__akshar_diacritics_tl	. <u>3</u> , <u>6</u> , <u>132</u>
__akshar_int_append_ordinal:n <u>94</u> , <u>254</u>
\c__akshar_joining_tl <u>3</u> , <u>6</u> , <u>139</u>
\l__akshar_map_tl <u>130</u> , <u>132</u> , <u>136</u> , <u>139</u> , <u>143</u> , <u>151</u> , <u>156</u> , <u>177</u> , <u>180</u> , <u>193</u> , <u>196</u> , <u>199</u> , <u>207</u> , <u>210</u> , <u>219</u> , <u>223</u> , <u>227</u>
\l__akshar_prev_joining_bool <u>7</u> , <u>23</u> , <u>129</u> , <u>144</u> , <u>147</u> , <u>152</u>
__akshar_replace:NnnnN <u>168</u> , <u>285</u> , <u>289</u> , <u>298</u> , <u>302</u>
__akshar_seq_push_seq:NN <u>166</u> , <u>188</u> , <u>214</u> , <u>233</u>
\c__akshar_str_g_tl <u>66</u>
\c__akshar_str_seq_tl <u>66</u>
\l__akshar_tmpa_int	<u>7</u> , <u>8</u> , <u>25</u> , <u>175</u> , <u>183</u> , <u>190</u> , <u>195</u> , <u>205</u> , <u>209</u> , <u>213</u> , <u>221</u>
\l__akshar_tmpa_seq <u>25</u> , <u>173</u> , <u>189</u> , <u>199</u> , <u>215</u> , <u>227</u> , <u>233</u> , <u>235</u> , <u>236</u> , <u>240</u> , <u>241</u> , <u>245</u> , <u>249</u> , <u>251</u> , <u>255</u> , <u>261</u> , <u>262</u> , <u>268</u> , <u>272</u> , <u>273</u> , <u>279</u> , <u>285</u> , <u>289</u> , <u>292</u> , <u>298</u> , <u>302</u> , <u>305</u>
\l__akshar_tmpa_tl <u>25</u> , <u>80</u> , <u>84</u> , <u>134</u> , <u>136</u> , <u>141</u> , <u>143</u> , <u>149</u> , <u>151</u> , <u>191</u> , <u>193</u> , <u>203</u> , <u>207</u> , <u>217</u> , <u>219</u>
\l__akshar_tmpb_int <u>25</u> , <u>176</u> , <u>179</u> , <u>186</u>
\l__akshar_tmpb_seq <u>25</u> , <u>78</u> , <u>80</u> , <u>81</u> , <u>174</u> , <u>180</u> , <u>187</u> , <u>196</u> , <u>210</u> , <u>215</u> , <u>216</u> , <u>223</u> , <u>233</u>
\l__akshar_tmpb_tl <u>25</u> , <u>81</u> , <u>82</u>
\l__akshar_tmpe_seq	... <u>25</u> , <u>170</u> , <u>177</u>
\l__akshar_tmpe_seq <u>25</u> , <u>171</u> , <u>183</u> , <u>192</u> , <u>205</u> , <u>218</u>
__akshar_var_if_global <u>5</u>
__akshar_var_if_global:NTF <u>66</u> , <u>161</u> , <u>234</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>243</u>
\aksharStrHead <u>2</u> , <u>5</u> , <u>259</u>
\aksharStrLen <u>2</u> , <u>238</u>
\aksharStrRemove <u>3</u> , <u>294</u>
\aksharStrRemove* <u>3</u> , <u>294</u>
\aksharStrReplace <u>2</u> , <u>10</u> , <u>281</u>
\aksharStrReplace* <u>2</u> , <u>281</u>
\aksharStrTail <u>2</u> , <u>5</u> , <u>270</u>
B	
bool commands:	
\bool_if:NTF <u>147</u> , <u>185</u>
\bool_if:nTF <u>70</u> , <u>246</u>
\bool_new:N <u>23</u>
\bool_set_false:N <u>129</u> , <u>152</u>
\bool_set_true:N <u>144</u>
\c_false_bool <u>7</u> , <u>290</u> , <u>303</u>
\c_true_bool <u>7</u> , <u>286</u> , <u>299</u>
C	
cs commands:	
\cs_generate_variant:Nn <u>63</u> , <u>64</u> , <u>65</u> , <u>165</u>
\cs_new:Npn <u>94</u> , <u>126</u> , <u>166</u> , <u>168</u>
\cs_split_function:N <u>71</u> , <u>79</u>
E	
exp commands:	
\exp_last_unbraced:Nn <u>71</u> , <u>79</u>
I	
IfBooleanTF <u>283</u> , <u>296</u>
int commands:	
\int_case:nnTF <u>97</u> , <u>109</u> , <u>117</u>
\int_compare:nNnTF <u>107</u> , <u>179</u> , <u>182</u> , <u>262</u> , <u>273</u>
\int_compare_p:nNn <u>248</u> , <u>249</u>
\int_eval:n <u>255</u>
\int_incr:N <u>186</u> , <u>195</u> , <u>209</u> , <u>221</u>
\int_new:N <u>32</u> , <u>33</u>
\int_set:Nn <u>175</u> , <u>176</u> , <u>190</u> , <u>213</u>
M	
msg commands:	
\msg_error:nnn <u>64</u> , <u>64</u> , <u>73</u> , <u>88</u> , <u>264</u> , <u>275</u>
\msg_error:nnnnn <u>64</u> , <u>65</u> , <u>253</u>
\msg_new:nnnn <u>34</u> , <u>44</u> , <u>54</u>
N	
\NewExpandableDocumentCommand <u>238</u> , <u>243</u> , <u>259</u> , <u>270</u> , <u>281</u> , <u>294</u>
P	
prg commands:	
\prg_generate_conditional_	- variant:Nnn <u>62</u>
\prg_new_conditional:Npnn <u>68</u>
\prg_return_false: <u>75</u> , <u>85</u> , <u>90</u>
\prg_return_true: <u>85</u>
\ProvidesExplPackage <u>4</u>
R	
\RequirePackage <u>3</u>
S	
scan commands:	
\scan_stop:	.. <u>52</u> , <u>60</u> , <u>256</u> , <u>266</u> , <u>277</u>
seq commands:	
\seq_clear:N	<u>128</u> , <u>173</u> , <u>174</u> , <u>187</u> , <u>216</u>
\seq_count:N <u>183</u> , <u>241</u> , <u>249</u> , <u>255</u> , <u>262</u> , <u>273</u> , <u>279</u>
\seq_get_left:NN <u>80</u>
\seq_get_right:NN <u>81</u>
\seq_gset_eq:NN <u>162</u> , <u>235</u>
\seq_item:Nn <u>192</u> , <u>205</u> , <u>218</u> , <u>251</u> , <u>268</u> , <u>279</u>
\seq_map_inline:Nn <u>167</u>

<code>\seq_map_variable:NNn</code>	177	<code>\tl_if_in:Nn</code>	62
<code>\seq_new:N</code> ..	24, 27, 28, 29, 30, 31	<code>\tl_if_in:NnTF</code>	62, 132, 139
<code>\seq_pop_right:NN</code>	134, 141, 149	<code>\tl_map_variable:NNn</code>	130
<code>\seq_put_right:Nn</code>		<code>\tl_new:N</code>	25, 26
.....	135, 142, 150, 155,	<code>\tl_set:Nn</code>	191, 203, 217
.....	167, 180, 196, 199, 210, 222, 226	<code>\tl_to_str:n</code>	66, 67
<code>\seq_set_eq:NN</code>	163, 236	token commands:	
<code>\seq_set_split:Nnn</code>	63, 63, 78	<code>\token_to_str:N</code> .	52, 60, 74, 78, 89
<code>\seq_use:Nn</code>	292, 305		
T		U	
tl commands:		use commands:	
<code>\tl_const:Nn</code>	6, 7, 66, 67	<code>\use_i:nnn</code>	79
<code>\tl_if_eq:NNTF</code>	82, 84, 193, 207, 219	<code>\use_iii:nnn</code>	71