

# The akshar package

Vu Van Dung

Version 0.1 — 2020/05/17

## Abstract

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

## Contents

1	Introduction	1
2	User manual	1
2.1	$\LaTeX 2_{\epsilon}$ macros	1
2.2	expl3 functions	2
3	Implementation	2
3.1	Variable declarations	2
3.2	Messages	3
3.3	Utilities	4
3.4	The <code>\akshar_convert:Nn</code> function and its variants	5
3.5	Other internal functions	6
3.6	Front-end $\LaTeX 2_{\epsilon}$ macros	8
	Index	8

## 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

---

$\backslash\text{aksharStrLen}$   $\backslash\text{aksharStrLen} \{ \langle \text{token list} \rangle \}$

---

Return the number of Devanagari characters in the  $\langle \text{token list} \rangle$ .

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
```

---

$\backslash\text{aksharStrChar}$   $\backslash\text{aksharStrChar} \{ \langle \text{token list} \rangle \} \{ \langle n \rangle \}$

---

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
```

---

$\backslash\text{aksharStrReplace}$   $\backslash\text{aksharStrReplace} \{ \langle \text{tl 1} \rangle \} \{ \langle \text{tl 2} \rangle \} \{ \langle \text{tl 3} \rangle \}$

---

Replace all occurrences of  $\langle \text{tl 2} \rangle$  in  $\langle \text{tl 1} \rangle$  with  $\langle \text{tl 3} \rangle$ , and leaves the modified  $\langle \text{tl 1} \rangle$  in the input stream.

The starred variant will replace only the first occurrence of  $\langle \text{tl 2} \rangle$ , all others are left intact.

expl3 output:  
स्कास्कास्काडडस्कांळीस्कास्काड  
 $\backslash\text{aksharStrReplace}$  output:  
स्कास्कास्काडडमंळीस्कास्काड

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

### 2.2 expl3 functions

This section assumes that you have a basic knowledge in  $\text{\LaTeX}3$  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.

---

$\backslash\text{akshar\_convert:Nn}$

---

$\backslash\text{akshar\_convert:}(cn|Nx|cx)$

---

$\backslash\text{akshar\_convert:Nn} \langle \text{seq var} \rangle \{ \langle \text{token list} \rangle \}$

This function converts  $\langle \text{token list} \rangle$  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  $\text{\TeX}$  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 {akshar} {2020/05/17} {0.1}
5   {Support for syllables in the Devanagari script (JV)}
```

### 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: ा, ि, ી, ુ, ૂ, ૃ, ૄ, ૅ, ૆, ે, ૈ, ૉ, ૊, ો, ૌ, ્, ૎, ૏, ૐ, ૑, ૒, ૓, ૔, ૕, ૖, ૗, ૘, ૙, ૚, ૛, ૜, ૝, ૞, ૟, ૠ, ૡ, ૢ, ૣ, ૤, ૥, ૦, ૧, ૨, ૩, ૪, ૫, ૬, ૭, ૮, ૯, ૰, ૱, ૲, ૳, ૴, ૵, ૶, ૷, ૸, ૹ, ૺ, ૻ, ૼ, ૽, ૾, ૿, ૻ, ૼ, ૽, ૾, ૿.

[illegible]

(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.

```
13 \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.

```
14 \seq_new:N \l_akshar_char_seq
```

(End definition for \l akshar char seq.)

```
\l__akshar_tmpa_tl  
\l__akshar_tmptb_tl  
\l__akshar_tmtpa_seq  
\l__akshar_tmtpb_seq  
\l__akshar_tmtpc_seq  
\l__akshar_tmtpd_seq  
\l__akshar_tmtp_e_seq  
\l__akshar_tmtpa_int
```

Some temporary variables.

```

15 \tl_new:N \l__akshar_tmpa_tl
16 \tl_new:N \l__akshar_tmpb_tl
17 \seq_new:N \l__akshar_tmpa_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

```

(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.

```

23 \msg_new:nnnn { akshar } { err_not_a_sequence_variable }
24 { #1 ~ is ~ not ~ a ~ valid ~ LaTeX3 ~ sequence ~ variable. }
25 {
26   You ~ have ~ requested ~ me ~ to ~ assign ~ some ~ value ~ to ~
27   the ~ control ~ sequence ~ #1, ~ but ~ it ~ is ~ not ~ a ~ valid ~
28   sequence ~ variable. ~ Read ~ the ~ documentation ~ of ~ expl3 ~
29   for ~ more ~ information. ~ Proceed ~ and ~ I ~ will ~ pretend ~
30   that ~ #1 ~ is ~ a ~ local ~ sequence ~ variable ~ (beware ~ that ~
31   unexpected ~ behaviours ~ may ~ occur).
32 }

```

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.)

```

33 \msg_new:nnnn { akshar } { err_character_out_of_bound }
34 { 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 ~ 0 ~ and ~ #3, ~ so ~ that ~ I ~ can ~ return ~ a ~
40   good ~ output. ~ Proceed ~ and ~ I ~ will ~ return ~
41   \token_to_str:N \scan_stop:.
42 }

```

### 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.

```

43 \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`.

```

44 \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.  
`\msg_error:nnnxx`

```

45 \cs_generate_variant:Nn \msg_error:nnn { nxx }
46 \cs_generate_variant:Nn \msg_error:nnnnn { nnnxx }

```

(End definition for `\msg_error:nxx` 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`. If it is not a sequence variable, this function will (TODO) issue an error message.

```

\c__akshar_str_g_tl
\c__akshar_str_seq_tl
47 \tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
48 \tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
49 \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
50 {
51   \bool_if:nTF
52   { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
53   {
54     \msg_error:nxx { akshar } { err_not_a_sequence_variable }
55     { \token_to_str:N #1 }
56     \prg_return_false:
57   }
58   {
59     \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
60     { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
61     \seq_get_left:NN \l__akshar_tmpb_seq \l__akshar_tmpa_tl
62     \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
63     \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
64     {
65       \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
66       { \prg_return_true: } { \prg_return_false: }
67     }
68     {
69       \msg_error:nxx { akshar } { err_not_a_sequence_variable }
70       { \token_to_str:N #1 }
71       \prg_return_false:
72     }

```

```

73     }
74 }

```

(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.

```

75 \cs_new:Npn \__akshar_int_append_ordinal:n #1
76 {
77   #1
78   \int_case:nnF { #1 }
79   {
80     { 11 } { th }
81     { 12 } { th }
82     { 13 } { th }
83     { -11 } { th }
84     { -12 } { th }
85     { -13 } { th }
86   }
87   {
88     \int_compare:nNnTF { #1 } > { -1 }
89     {
90       \int_case:nnF { #1 - 10 * ( #1 / 10 ) }
91       {
92         { 1 } { st }
93         { 2 } { nd }
94         { 3 } { rd }
95       } { th }
96     }
97     {
98       \int_case:nnF { ( - #1 ) - 10 * ( ( - #1 ) / 10 ) }
99       {
100         { 1 } { st }
101         { 2 } { nd }
102         { 3 } { rd }
103       } { th }
104     }
105   }
106 }

```

(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
107 \cs_new:Npn \akshar_convert:Nn #1 #2
108 {

```

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

```

109   \seq_clear:N \l__akshar_char_seq
110   \bool_set_false:N \l__akshar_prev_joining_bool

```

Loop through every token of the input.

```

111   \tl_map_variable:NNn {#2} \l__akshar_map_tl
112   {
113     \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
114     {

```

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.

```

115       \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
116       \seq_put_right:Nx \l__akshar_char_seq
117       { \l__akshar_tmpa_tl \l__akshar_map_tl }
118     }

```

```

119         {
120             \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
121             {

```

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.

```

122                 \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
123                 \seq_put_right:Nx \l__akshar_char_seq
124                 { \l__akshar_tmpa_tl \l__akshar_map_tl }
125                 \bool_set_true:N \l__akshar_prev_joining_bool
126             }
127         {

```

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

```

128             \bool_if:NNTF \l__akshar_prev_joining_bool
129             {
130                 \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
131                 \seq_put_right:Nx \l__akshar_char_seq
132                 { \l__akshar_tmpa_tl \l__akshar_map_tl }
133                 \bool_set_false:N \l__akshar_prev_joining_bool
134             }
135             {
136                 \seq_put_right:Nx
137                 \l__akshar_char_seq { \l__akshar_map_tl }
138             }
139         }
140     }
141 }

```

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

```

142     \__akshar_var_if_global:NNTF #1
143     { \seq_gset_eq:NN #1 \l__akshar_char_seq }
144     { \seq_set_eq:NN #1 \l__akshar_char_seq }
145 }

```

Generate variants that might be helpful for some.

```

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

```

(End definition for `\akshar_convert:Nn`. This function is documented on page 2.)

### 3.5 Other internal functions

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

```

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

```

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

`\__akshar_replace_all:Nnnn` This function replaces all occurrences of #3 in #2 by #4. The result is stored in a sequence variable #1.

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.

```

149 \cs_new:Npn \__akshar_replace_all:Nnnn #1 #2 #3 #4
150 {
151   \akshar_convert:Nn \l__akshar_tmpe_seq {#2}
152   \akshar_convert:Nn \l__akshar_tmpe_seq {#3}
153   \akshar_convert:Nn \l__akshar_tmpe_seq {#4}
154   \seq_clear:N \l__akshar_tmpe_seq
155   \seq_clear:N \l__akshar_tmpe_seq
156   \int_set:Nn \l__akshar_tmpe_int { 1 }
157   \seq_map_variable:Nnn \l__akshar_tmpe_seq \l__akshar_map_tl
158   {
159     \int_compare:nNnTF
160       { \l__akshar_tmpe_int } = { 1 + \seq_count:N \l__akshar_tmpe_seq }
161       {
162         \seq_clear:N \l__akshar_tmpe_seq
163         \__akshar_seq_push_seq:NN
164           \l__akshar_tmpe_seq \l__akshar_tmpe_seq
165         \int_set:Nn \l__akshar_tmpe_int { 1 }
166         \tl_set:Nx \l__akshar_tmpe_tl
167           { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
168         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
169           {
170             \int_incr:N \l__akshar_tmpe_int
171             \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
172           }
173         { \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl }
174       }
175     {
176       \tl_set:Nx \l__akshar_tmpe_tl
177         { \seq_item:Nn \l__akshar_tmpe_seq { \l__akshar_tmpe_int } }
178       \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
179         {
180           \int_incr:N \l__akshar_tmpe_int
181           \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
182         }
183       {
184         \int_set:Nn \l__akshar_tmpe_int { 1 }
185         \__akshar_seq_push_seq:NN
186           \l__akshar_tmpe_seq \l__akshar_tmpe_seq
187         \seq_clear:N \l__akshar_tmpe_seq
188         \tl_set:Nx \l__akshar_tmpe_tl
189           { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
190         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
191           {
192             \int_incr:N \l__akshar_tmpe_int
193             \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
194           }
195       }
196     }
197   }

```

```

195         {
196         \seq_put_right:NV \l__akshar_tmpa_seq \l__akshar_map_tl
197         }
198     }
199 }
200 }
201 \__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpb_seq
202 \__akshar_var_if_global:NTF #1
203 { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
204 { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
205 }

```

(End definition for `\__akshar_replace_all:Nnnn`.)

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

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

```

206 \NewExpandableDocumentCommand \aksharStrLen {m}
207 {
208     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
209     \seq_count:N \l__akshar_tmpa_seq
210 }

```

(End definition for `\aksharStrLen`. This function is documented on page 1.)

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

```

211 \NewExpandableDocumentCommand \aksharStrChar {mm}
212 {
213     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
214     \bool_if:nTF
215     {
216         \int_compare_p:nNn {#2} > {0} &&
217         \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq}
218     }
219     { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
220     {
221         \msg_error:nnnxx { akshar } { err_character_out_of_bound }
222         { #1 } { \__akshar_int_append_ordinal:n { #2 } }
223         { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
224         \scan_stop:
225     }
226 }

```

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

`\aksharStrReplace` Replace any apparence of #3 of a string #2 with another string #4.

```

227 \NewExpandableDocumentCommand \aksharStrReplace {smmm}
228 {
229     \IfBooleanTF {#1}
230     { \iow_log:n { Do ~ nothing } }
231     { \__akshar_replace_all:Nnnn \l__akshar_tmpa_seq {#2} {#3} {#4} }
232     \seq_use:Nn \l__akshar_tmpa_seq {}
233 }

```

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

```

234 </package>

```

## Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.



3, 5, [107](#), [151](#), [152](#), [153](#), [208](#), [213](#)  
 akshar internal commands:  
 \l\_\_akshar\_char\_seq .....  
         ..... [6](#), [14](#), [109](#), [115](#), [116](#),  
                 [122](#), [123](#), [130](#), [131](#), [137](#), [143](#), [144](#)  
 \c\_\_akshar\_diacritics\_tl . [3](#), [6](#), [113](#)  
 \\_\_akshar\_int\_append\_ordinal:n  
         ..... [75](#), [222](#)  
 \c\_\_akshar\_joining\_tl .... [3](#), [6](#), [120](#)  
 \l\_\_akshar\_map\_tl [111](#), [113](#), [117](#),  
         [120](#), [124](#), [132](#), [137](#), [157](#), [168](#),  
         [171](#), [173](#), [178](#), [181](#), [190](#), [193](#), [196](#)  
 \l\_\_akshar\_prev\_joining\_bool ..  
         ..... [5](#), [13](#), [110](#), [125](#), [128](#), [133](#)  
 \\_\_akshar\_replace\_all:Nnnn [149](#), [231](#)  
 \\_\_akshar\_seq\_push\_seq:NN .....  
         ..... [147](#), [163](#), [185](#), [201](#)  
 \c\_\_akshar\_str\_g\_tl ..... [47](#)  
 \c\_\_akshar\_str\_seq\_tl ..... [47](#)  
 \l\_\_akshar\_tmpa\_int [6](#), [7](#), [15](#), [156](#),  
         [160](#), [165](#), [170](#), [177](#), [180](#), [184](#), [192](#)  
 \l\_\_akshar\_tmpa\_seq .....  
         ..... [15](#), [154](#), [164](#), [173](#),  
                 [186](#), [196](#), [201](#), [203](#), [204](#), [208](#),  
                 [209](#), [213](#), [217](#), [219](#), [223](#), [231](#), [232](#)  
 \l\_\_akshar\_tmpa\_tl ..... [15](#),  
         [61](#), [65](#), [115](#), [117](#), [122](#), [124](#), [130](#),  
         [132](#), [166](#), [168](#), [176](#), [178](#), [188](#), [190](#)  
 \l\_\_akshar\_tmpb\_seq .....  
         ..... [15](#), [59](#), [61](#), [62](#), [155](#),  
                 [162](#), [171](#), [181](#), [186](#), [187](#), [193](#), [201](#)  
 \l\_\_akshar\_tmpb\_tl ..... [15](#), [62](#), [63](#)  
 \l\_\_akshar\_tmpe\_seq ... [15](#), [151](#), [157](#)  
 \l\_\_akshar\_tmpe\_seq .....  
         ..... [15](#), [152](#), [160](#), [167](#), [177](#), [189](#)  
 \l\_\_akshar\_tmpe\_seq ... [15](#), [153](#), [164](#)  
 \\_\_akshar\_var\_if\_global ..... [4](#)  
 \\_\_akshar\_var\_if\_global:NTF ...  
         ..... [47](#), [142](#), [202](#)  
 \aksharStrChar ..... [2](#), [3](#), [211](#)  
 \aksharStrLen ..... [1](#), [206](#)  
 \aksharStrReplace ..... [2](#), [227](#)  
 \aksharStrReplace\* ..... [2](#)

## B

bool commands:  
 \bool\_if:NTF ..... [128](#)  
 \bool\_if:nTF ..... [51](#), [214](#)  
 \bool\_new:N ..... [13](#)  
 \bool\_set\_false:N ..... [110](#), [133](#)  
 \bool\_set\_true:N ..... [125](#)

## C

cs commands:  
 \cs\_generate\_variant:Nn .....  
         ..... [44](#), [45](#), [46](#), [146](#)  
 \cs\_new:Npn ..... [75](#), [107](#), [147](#), [149](#)  
 \cs\_split\_function:N ..... [52](#), [60](#)

## E

exp commands:  
 \exp\_last\_unbraced:Nf ..... [52](#), [60](#)

## I

\IfBooleanTF ..... [229](#)  
 int commands:  
 \int\_case:nnTF ..... [78](#), [90](#), [98](#)  
 \int\_compare:nNnTF ..... [88](#), [159](#)

\int\_compare\_p:nNn ..... [216](#), [217](#)  
 \int\_eval:n ..... [223](#)  
 \int\_incr:N ..... [170](#), [180](#), [192](#)  
 \int\_new:N ..... [22](#)  
 \int\_set:Nn ..... [156](#), [165](#), [184](#)  
 iow commands:  
 \iow\_log:n ..... [230](#)

## M

msg commands:  
 \msg\_error:nnn ..... [45](#), [45](#), [54](#), [69](#)  
 \msg\_error:nnnn ..... [45](#), [46](#), [221](#)  
 \msg\_new:nnnn ..... [23](#), [33](#)

## N

\NewExpandableDocumentCommand ....  
         ..... [206](#), [211](#), [227](#)

## P

prg commands:  
 \prg\_generate\_conditional\_-  
         variant:Nnn ..... [43](#)  
 \prg\_new\_conditional:Npnn ..... [49](#)  
 \prg\_return\_false: ..... [56](#), [66](#), [71](#)  
 \prg\_return\_true: ..... [66](#)  
 \ProvidesExplPackage ..... [4](#)

## R

\RequirePackage ..... [3](#)

## S

scan commands:  
 \scan\_stop: ..... [41](#), [224](#)  
 seq commands:  
 \seq\_clear:N [109](#), [154](#), [155](#), [162](#), [187](#)  
 \seq\_count:N ... [160](#), [209](#), [217](#), [223](#)  
 \seq\_get\_left:NN ..... [61](#)  
 \seq\_get\_right:NN ..... [62](#)  
 \seq\_gset\_eq:NN ..... [143](#), [203](#)  
 \seq\_item:Nn ... [167](#), [177](#), [189](#), [219](#)  
 \seq\_map\_inline:Nn ..... [148](#)  
 \seq\_map\_variable:Nnn ..... [157](#)  
 \seq\_new:N .. [14](#), [17](#), [18](#), [19](#), [20](#), [21](#)  
 \seq\_pop\_right:NN .... [115](#), [122](#), [130](#)  
 \seq\_put\_right:Nn [116](#), [123](#), [131](#),  
         [136](#), [148](#), [171](#), [173](#), [181](#), [193](#), [196](#)  
 \seq\_set\_eq:NN ..... [144](#), [204](#)  
 \seq\_set\_split:Nnn ..... [44](#), [44](#), [59](#)  
 \seq\_use:Nn ..... [232](#)

## T

tl commands:  
 \tl\_const:Nn ..... [6](#), [7](#), [47](#), [48](#)  
 \tl\_if\_eq:NNTF .....  
         ..... [63](#), [65](#), [120](#), [168](#), [178](#), [190](#)  
 \tl\_if\_in:Nn ..... [43](#)  
 \tl\_if\_in:NnTF ..... [43](#), [113](#)  
 \tl\_map\_variable:Nnn ..... [111](#)  
 \tl\_new:N ..... [15](#), [16](#)  
 \tl\_set:Nn ..... [166](#), [176](#), [188](#)  
 \tl\_to\_str:n ..... [47](#), [48](#)  
 token commands:  
 \token\_to\_str:N .... [41](#), [55](#), [59](#), [70](#)

## U

use commands:  
 \use\_i:nnn ..... [60](#)  
 \use\_iii:nnn ..... [52](#)