

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	3
3	Implementation	3
3.1	Variable declarations	3
3.2	Messages	4
3.3	Utilities	4
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	8
	Index	9

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 L^AT_EX 2_ε 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
```

`\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 नमस्कारis \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>}`

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 { मममडडमंलीममड}
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 { मममडडमंलीममड} { म } { स्का}
8 \ExplSyntaxOff
```

expl3 output:
स्कांममडडमंलीममड
\aksharStrReplace* output:
ममंस्काडडमंलीममड

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

`\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:
डडलीड
\aksharStrRemove output:
डडमंलीड

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

```

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

```

(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` Some temporary variables.

```
\l__akshar_tmpl_tl
\l__akshar_tmplb_tl
\l__akshar_tmpla_seq 15 \tl_new:N \l__akshar_tmpl_a_tl
\l__akshar_tmplb_seq 16 \tl_new:N \l__akshar_tmpl_b_tl
\l__akshar_tmplc_seq 17 \seq_new:N \l__akshar_tmpl_a_seq
\l__akshar_tmpld_seq 18 \seq_new:N \l__akshar_tmpl_b_seq
\l__akshar_tmpe_seq 19 \seq_new:N \l__akshar_tmpl_c_seq
\l__akshar_tmpla_int 20 \seq_new:N \l__akshar_tmpl_d_seq
\l__akshar_tmplb_int 21 \seq_new:N \l__akshar_tmpl_e_seq
22 \int_new:N \l__akshar_tmpl_a_int
23 \int_new:N \l__akshar_tmpl_b_int
```

(End definition for `\l__akshar_tmpl_a_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.

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

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

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.

```
44 \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`.

```
45 \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

```
46 \cs_generate_variant:Nn \msg_error:nnn { nnx }
47 \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. If it is not a sequence variable, this function will (TODO) issue an error message.

```
48 \tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
49 \tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
50 \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
51 {
52   \bool_if:nTF
53     { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
54     {
55       \msg_error:nnx { akshar } { err_not_a_sequence_variable }
56       { \token_to_str:N #1 }
57       \prg_return_false:
58     }
59     {
60       \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
61       { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
62       \seq_get_left:NN \l__akshar_tmpb_seq \l__akshar_tmpa_tl
63       \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
64       \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
65       {
66         \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
67         { \prg_return_true: } { \prg_return_false: }
68       }
69       {
70         \msg_error:nnx { akshar } { err_not_a_sequence_variable }
71         { \token_to_str:N #1 }
72         \prg_return_false:
73       }
74     }
75 }
```

(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.

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

```

99         \int_case:nnF { (- #1) - 10 * ((- #1) / 10) }
100         {
101             { 1 } { st }
102             { 2 } { nd }
103             { 3 } { rd }
104             } { th }
105         }
106     }
107 }

```

(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
108 \cs_new:Npn \akshar_convert:Nn #1 #2
109 {

```

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

```

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

```

Loop through every token of the input.

```

112     \tl_map_variable:NNn {#2} \l__akshar_map_tl
113     {
114         \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
115         {

```

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.

```

116             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
117             \seq_put_right:Nx \l__akshar_char_seq
118             { \l__akshar_tmpa_tl \l__akshar_map_tl }
119         }
120     {
121         \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
122         {

```

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

```

123             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
124             \seq_put_right:Nx \l__akshar_char_seq
125             { \l__akshar_tmpa_tl \l__akshar_map_tl }
126             \bool_set_true:N \l__akshar_prev_joining_bool
127         }
128     }

```

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

```

129         \bool_if:NNTF \l__akshar_prev_joining_bool
130         {
131             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
132             \seq_put_right:Nx \l__akshar_char_seq
133             { \l__akshar_tmpa_tl \l__akshar_map_tl }
134             \bool_set_false:N \l__akshar_prev_joining_bool
135         }
136         {
137             \seq_put_right:Nx
138             \l__akshar_char_seq { \l__akshar_map_tl }
139         }
140     }
141 }
142 }

```

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

```

143 \__akshar_var_if_global:NTF #1
144 { \seq_gset_eq:NN #1 \l__akshar_char_seq }
145 { \seq_set_eq:NN #1 \l__akshar_char_seq }
146 }

```

Generate variants that might be helpful for some.

```

147 \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.

```

148 \cs_new:Npn \__akshar_seq_push_seq:NN #1 #2
149 { \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.

```

150 \cs_new:Npn \__akshar_replace:NnnnN #1 #2 #3 #4 #5
151 {
152   \akshar_convert:Nn \l__akshar_tmpe_seq {#2}
153   \akshar_convert:Nn \l__akshar_tmpe_seq {#3}

```

```

154 \akshar_convert:Nn \l__akshar_tmpe_seq {#4}
155 \seq_clear:N \l__akshar_tmpe_seq
156 \seq_clear:N \l__akshar_tmpe_seq
157 \int_set:Nn \l__akshar_tmpe_int { 1 }
158 \int_set:Nn \l__akshar_tmpe_int { 0 }
159 \seq_map_variable:Nn \l__akshar_tmpe_seq \l__akshar_map_tl
160 {
161   \int_compare:nNnTF { \l__akshar_tmpe_int } > { 0 }
162   { \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl }
163   {
164     \int_compare:nNnTF
165     { \l__akshar_tmpe_int } = { 1 + \seq_count:N \l__akshar_tmpe_seq }
166     {
167       \bool_if:NT {#5}
168       { \int_incr:N \l__akshar_tmpe_int }
169       \seq_clear:N \l__akshar_tmpe_seq
170       \__akshar_seq_push_seq:NN
171       \l__akshar_tmpe_seq \l__akshar_tmpe_seq
172       \int_set:Nn \l__akshar_tmpe_int { 1 }
173       \tl_set:Nx \l__akshar_tmpe_tl
174       { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
175       \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
176       {
177         \int_incr:N \l__akshar_tmpe_int
178         \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
179       }
180       {
181         \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
182       }
183     }
184   }
185   \tl_set:Nx \l__akshar_tmpe_tl
186   {
187     \seq_item:Nn \l__akshar_tmpe_seq { \l__akshar_tmpe_int }
188   }
189   \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
190   {
191     \int_incr:N \l__akshar_tmpe_int
192     \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
193   }
194   {
195     \int_set:Nn \l__akshar_tmpe_int { 1 }
196     \__akshar_seq_push_seq:NN
197     \l__akshar_tmpe_seq \l__akshar_tmpe_seq
198     \seq_clear:N \l__akshar_tmpe_seq
199     \tl_set:Nx \l__akshar_tmpe_tl
200     { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
201     \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
202     {
203       \int_incr:N \l__akshar_tmpe_int
204       \seq_put_right:NV
205       \l__akshar_tmpe_seq \l__akshar_map_tl
206     }
207     {
208       \seq_put_right:NV
209       \l__akshar_tmpe_seq \l__akshar_map_tl
210     }
211   }
212 }
213 }
214 }
215 \__akshar_seq_push_seq:NN \l__akshar_tmpe_seq \l__akshar_tmpe_seq
216 \__akshar_var_if_global:NNTF #1
217 { \seq_gset_eq:NN #1 \l__akshar_tmpe_seq }
218 { \seq_set_eq:NN #1 \l__akshar_tmpe_seq }
219 }

```

(End definition for __akshar_replace:NnnnN.)

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

\aksharStrLen Expands to the length of the string.

```

220 \NewExpandableDocumentCommand \aksharStrLen {m}
221 {
222   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
223   \seq_count:N \l__akshar_tmpa_seq
224 }
```

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

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

```

225 \NewExpandableDocumentCommand \aksharStrChar {mm}
226 {
227   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
228   \bool_if:nTF
229     {
230       \int_compare_p:nNn {#2} > {0} &&
231       \int_compare_p:nNn {#2} < {1 + \seq_count:N \l__akshar_tmpa_seq}
232     }
233     { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
234     {
235       \msg_error:nnnxx { akshar } { err_character_out_of_bound }
236       { #1 } { \__akshar_int_append_ordinal:n { #2 } }
237       { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
238       \scan_stop:
239     }
240 }
```

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

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

```

241 \NewExpandableDocumentCommand \aksharStrReplace {smmm}
242 {
243   \IfBooleanTF {#1}
244     {
245       \__akshar_replace:NnnnN \l__akshar_tmpa_seq
246       {#2} {#3} {#4} \c_true_bool
247     }
248     {
249       \__akshar_replace:NnnnN \l__akshar_tmpa_seq
250       {#2} {#3} {#4} \c_false_bool
251     }
252     \seq_use:Nn \l__akshar_tmpa_seq {}
253 }
```

(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*

```

254 \NewExpandableDocumentCommand \aksharStrRemove {smm}
255 {
256   \IfBooleanTF {#1}
257     {
258       \__akshar_replace:NnnnN \l__akshar_tmpa_seq
259       {#2} {#3} {} \c_true_bool
260     }
261     {
262       \__akshar_replace:NnnnN \l__akshar_tmpa_seq
263       {#2} {#3} {} \c_false_bool
264     }
265     \seq_use:Nn \l__akshar_tmpa_seq {}
266 }
```

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

```

267 \endpackage
```

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.

A		C	
akshar commands:		cs commands:	
\akshar_convert:Nn 1, 3, 4, 6, <u>108</u> , 152, 153, 154, 222, 227		\cs_generate_variant:Nn 45, 46, 47, 147	
akshar internal commands:		\cs_new:Npn 76, 108, 148, 150	
\l__akshar_char_seq 6, <u>14</u> , 110, 116, 117, 123, 124, 131, 132, 138, 144, 145		\cs_split_function:N 53, 61	
\c__akshar_diacritics_tl . 3, <u>6</u> , 114		E	
__akshar_int_append_ordinal:n <u>76</u> , 236		exp commands:	
\c__akshar_joining_tl 3, <u>6</u> , 121		\exp_last_unbraced:Nf 53, 61	
\l__akshar_map_tl 112, 114, 118, 121, 125, 133, 138, 159, 162, 175, 178, 181, 189, 192, 201, 205, 209		I	
\l__akshar_prev_joining_bool 6, <u>13</u> , 111, 126, 129, 134		\IfBooleanTF 243, 256	
__akshar_replace:NnnN <u>150</u> , 245, 249, 258, 262		int commands:	
__akshar_seq_push_seq:NN <u>148</u> , 170, 196, 215		\int_case:nnTF 79, 91, 99	
\c__akshar_str_g_tl <u>48</u>		\int_compare:nNnTF 89, 161, 164	
\c__akshar_str_seq_tl <u>48</u>		\int_compare_p:nNn 230, 231	
\l__akshar_tmpa_int . 7, <u>15</u> , 157, 165, 172, 177, 187, 191, 195, 203		\int_eval:n 237	
\l__akshar_tmpa_seq <u>15</u> , 155, 171, 181, 197, 209, 215, 217, 218, 222, 223, 227, 231, 233, 237, 245, 249, 252, 258, 262, 265		\int_incr:N 168, 177, 191, 203	
\l__akshar_tmpa_tl <u>15</u> , 62, 66, 116, 118, 123, 125, 131, 133, 173, 175, 185, 189, 199, 201		\int_new:N 22, 23	
\l__akshar_tmpe_int <u>15</u> , 158, 161, 168		\int_set:Nn 157, 158, 172, 195	
\l__akshar_tmpe_seq <u>15</u> , 60, 62, 63, 156, 162, 169, 178, 192, 197, 198, 205, 215		M	
\l__akshar_tmpe_tl <u>15</u> , 63, 64		msg commands:	
\l__akshar_tmpe_seq ... <u>15</u> , 152, 159		\msg_error:nnn <u>46</u> , 46, 55, 70	
\l__akshar_tmpe_seq <u>15</u> , 153, 165, 174, 187, 200		\msg_error:nnnn <u>46</u> , 47, 235	
\l__akshar_tmpe_seq ... <u>15</u> , 154, 171		\msg_new:nnnn 24, 34	
__akshar_var_if_global 4		N	
__akshar_var_if_global:Nf <u>48</u> , 143, 216		\NewExpandableDocumentCommand 220, 225, 241, 254	
\aksharStrChar 2, 4, <u>225</u>		P	
\aksharStrLen 1, <u>220</u>		prg commands:	
\aksharStrRemove 2, <u>254</u>		\prg_generate_conditional_ variant:Nnn 44	
\aksharStrRemove* 2, <u>254</u>		\prg_new_conditional:Npnn 50	
\aksharStrReplace 2, 9, <u>241</u>		\prg_return_false: 57, 67, 72	
\aksharStrReplace* 2, <u>241</u>		\prg_return_true: 67	
B		\ProvidesExplPackage 4	
bool commands:		R	
\bool_if:Nf 129, 167		\RequirePackage 3	
\bool_if:nf 52, 228		S	
\bool_new:N 13		scan commands:	
\bool_set_false:N 111, 134		\scan_stop: 42, 238	
\bool_set_true:N 126		seq commands:	
\c_false_bool 7, 250, 263		\seq_clear:N 110, 155, 156, 169, 198	
\c_true_bool 7, 246, 259		\seq_count:N ... 165, 223, 231, 237	
C		\seq_get_left:NN 62	
cs commands:		\seq_get_right:NN 63	
\cs_generate_variant:Nn 45, 46, 47, 147		\seq_gset_eq:NN 144, 217	
\cs_new:Npn 76, 108, 148, 150		\seq_item:Nn ... 174, 187, 200, 233	
\cs_split_function:N 53, 61		\seq_map_inline:Nn 149	
E		\seq_map_variable:Nnn 159	
exp commands:		\seq_new:N .. 14, 17, 18, 19, 20, 21	
\exp_last_unbraced:Nf 53, 61		\seq_pop_right:NN 116, 123, 131	
I		\seq_put_right:Nn 117, 124, 132, 137, 149, 162, 178, 181, 192, 204, 208	
\IfBooleanTF 243, 256		\seq_set_eq:NN 145, 218	
int commands:		\seq_set_split:Nnn <u>45</u> , 45, 60	
\int_case:nnTF 79, 91, 99		\seq_use:Nn 252, 265	
\int_compare:nNnTF 89, 161, 164			
\int_compare_p:nNn 230, 231			
\int_eval:n 237			
\int_incr:N 168, 177, 191, 203			
\int_new:N 22, 23			
\int_set:Nn 157, 158, 172, 195			
M			
msg commands:			
\msg_error:nnn <u>46</u> , 46, 55, 70			
\msg_error:nnnn <u>46</u> , 47, 235			
\msg_new:nnnn 24, 34			
N			
\NewExpandableDocumentCommand 220, 225, 241, 254			
P			
prg commands:			
\prg_generate_conditional_ variant:Nnn 44			
\prg_new_conditional:Npnn 50			
\prg_return_false: 57, 67, 72			
\prg_return_true: 67			
\ProvidesExplPackage 4			
R			
\RequirePackage 3			
S			
scan commands:			
\scan_stop: 42, 238			
seq commands:			
\seq_clear:N 110, 155, 156, 169, 198			
\seq_count:N ... 165, 223, 231, 237			
\seq_get_left:NN 62			
\seq_get_right:NN 63			
\seq_gset_eq:NN 144, 217			
\seq_item:Nn ... 174, 187, 200, 233			
\seq_map_inline:Nn 149			
\seq_map_variable:Nnn 159			
\seq_new:N .. 14, 17, 18, 19, 20, 21			
\seq_pop_right:NN 116, 123, 131			
\seq_put_right:Nn 117, 124, 132, 137, 149, 162, 178, 181, 192, 204, 208			
\seq_set_eq:NN 145, 218			
\seq_set_split:Nnn <u>45</u> , 45, 60			
\seq_use:Nn 252, 265			

	T		\tl_set:Nn 173, 185, 199
tl commands:			\tl_to_str:n 48, 49
\tl_const:Nn 6, 7, 48, 49		token commands:	
\tl_if_eq:NNTF 64, 66, 121, 175, 189, 201		\token_to_str:N 42, 56, 60, 71	
\tl_if_in:Nn 44			U
\tl_if_in:NnTF 44, 114		use commands:	
\tl_map_variable:NNn 112		\use_i:nnn 61	
\tl_new:N 15, 16		\use_iii:nnn 53	