

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</code> function	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

`\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 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:
स्कास्कास्काडडस्कांळीस्कास्काड
\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.

`\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>
```

```
3 \RequirePackage{fontspec}
4 \ProvidesExplPackage {akshar} {2020/05/17} {0.1}
5 {Support for syllables in the Devanagari script (JV)}
```

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 ~ string ~
37   #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~ Make ~ sure ~ that ~
38   you ~ use ~ a ~ number ~ between ~ and ~ not ~ including ~ 0 ~ and ~ #3, ~
39   so ~ that ~ I ~ can ~ return ~ a ~ good ~ output. ~ Proceed ~ and ~ I ~
40   will ~ return ~ \token_to_str:N \scan_stop:.
41 }

```

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.

```

42 \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`.

```

43 \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`

```

44 \cs_generate_variant:Nn \msg_error:nnn { nnx }
45 \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`

```

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

```

(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.

```

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

```

(End definition for `__akshar_int_append_ordinal:n`.)

3.4 The `\akshar_convert` function

`\akshar_convert:Nn` This converts #2 to a sequence of true Devanagari characters. The sequence
`\akshar_convert:cn` is set to #1, which should be a sequence variable. The assignment is local.
`\akshar_convert:Nx`
`\akshar_convert:cx`

```

106 \cs_new:Npn \akshar_convert:Nn #1 #2
107 {

```

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

```

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

```

Loop through every token of the input.

```

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

```

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.

```

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

```

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.

```

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

```

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

```

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

```

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

```

140     \__akshar_var_if_global:NTF #1
141     { \seq_gset_eq:NN #1 \l__akshar_char_seq }
142     { \seq_set_eq:NN #1 \l__akshar_char_seq }
143 }

```

Generate variants that might be helpful for some.

```

144 \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.

```

145 \cs_new:Npn \__akshar_seq_push_seq:NN #1 #2
146 { \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.

```

147 \cs_new:Npn \__akshar_replace_all:Nnnn #1 #2 #3 #4
148 {
149   \akshar_convert:Nn \l__akshar_tmpe_seq {#2}
150   \akshar_convert:Nn \l__akshar_tmpe_seq {#3}
151   \akshar_convert:Nn \l__akshar_tmpe_seq {#4}
152   \seq_clear:N \l__akshar_tmpe_seq
153   \seq_clear:N \l__akshar_tmpe_seq
154   \int_set:Nn \l__akshar_tmpe_int { 1 }
155   \seq_map_variable:Nnn \l__akshar_tmpe_seq \l__akshar_map_tl
156   {
157     \int_compare:nNnTF
158       { \l__akshar_tmpe_int } = { 1 + \seq_count:N \l__akshar_tmpe_seq }
159       {
160         \seq_clear:N \l__akshar_tmpe_seq
161         \__akshar_seq_push_seq:NN \l__akshar_tmpe_seq \l__akshar_tmpe_seq
162         \int_set:Nn \l__akshar_tmpe_int { 1 }
163         \tl_set:Nx \l__akshar_tmpe_tl
164           { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
165         \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
166           {
167             \int_incr:N \l__akshar_tmpe_int
168             \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
169           }
170           { \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl }
171       }
172     {
173       \tl_set:Nx \l__akshar_tmpe_tl
174         { \seq_item:Nn \l__akshar_tmpe_seq { \l__akshar_tmpe_int } }
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           \int_set:Nn \l__akshar_tmpe_int { 1 }
182           \__akshar_seq_push_seq:NN \l__akshar_tmpe_seq \l__akshar_tmpe_seq
183           \seq_clear:N \l__akshar_tmpe_seq
184           \tl_set:Nx \l__akshar_tmpe_tl
185             { \seq_item:Nn \l__akshar_tmpe_seq { 1 } }
186           \tl_if_eq:NNTF \l__akshar_map_tl \l__akshar_tmpe_tl
187             {
188               \int_incr:N \l__akshar_tmpe_int
189               \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl
190             }
191             { \seq_put_right:NV \l__akshar_tmpe_seq \l__akshar_map_tl }
192           }
193         }
194     }

```

```

195   \__akshar_seq_push_seq:NN \l__akshar_tmpa_seq \l__akshar_tmpp_seq
196   \__akshar_var_if_global:NTF #1
197   { \seq_gset_eq:NN #1 \l__akshar_tmpa_seq }
198   { \seq_set_eq:NN #1 \l__akshar_tmpa_seq }
199   }

```

(End definition for `__akshar_replace_all:Nnnn`.)

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

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

```

200 \NewExpandableDocumentCommand \aksharStrLen {m}
201 {
202   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
203   \seq_count:N \l__akshar_tmpa_seq
204 }

```

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

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

```

205 \NewExpandableDocumentCommand \aksharStrChar {mm}
206 {
207   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
208   \bool_if:nTF
209   {
210     \int_compare_p:nNn { #2 } > { 0 } &&
211     \int_compare_p:nNn { #2 } < { 1 + \seq_count:N \l__akshar_tmpa_seq }
212   }
213   { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
214   {
215     \msg_error:nnnxx { akshar } { err_character_out_of_bound }
216     { #1 } { \__akshar_int_append_ordinal:n { #2 } }
217     { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
218     \scan_stop:
219   }
220 }

```

(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.

```

221 \NewExpandableDocumentCommand \aksharStrReplace {smmm}
222 {
223   \IfBooleanTF {#1}
224   { \iow_log:n { Do ~ nothing } }
225   { \__akshar_replace_all:Nnnn \l__akshar_tmpa_seq {#2} {#3} {#4} }
226   \seq_use:Nn \l__akshar_tmpa_seq {}
227 }

```

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

```

228 </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.

A		.. 2, <u>106</u> , 149, 150, 151, 202, 207
akshar commands:		akshar internal commands:
\akshar_convert 1, 3, 5	\l__akshar_char_seq
\akshar_convert:Nn 6, 14, <u>108</u> , 114, 115,

121, 122, 129, 130, 135, 141, 142	\int_incr:N 167, 177, 188
\c__akshar_diacritics_tl . 3, 6, 112	\int_new:N 22
__akshar_int_append_ordinal:n 74, 216	\int_set:Nn 154, 162, 181
\c__akshar_joining_tl 3, 6, 119	low commands:
\l__akshar_map_tl 110, 112, 116, 119, 123, 131, 135, 155, 165, 168, 170, 175, 178, 186, 189, 191	\low_log:n 224
\l__akshar_prev_joining_bool 5, 13, 109, 124, 127, 132	M
__akshar_replace_all:Nnnn 147, 225	msg commands:
__akshar_seq_push_seq:NN 145, 161, 182, 195	\msg_error:nnn 44, 44, 53, 68
\c__akshar_str_g_tl 46	\msg_error:nnnn 44, 45, 215
\c__akshar_str_seq_tl 46	\msg_new:nnnn 23, 33
\l__akshar_tmpa_int 6, 7, 15, 154, 158, 162, 167, 174, 177, 181, 188	N
\l__akshar_tmpa_seq 15, 152, 161, 170, 182, 191, 195, 197, 198, 202, 203, 207, 211, 213, 217, 225, 226	\NewExpandableDocumentCommand 200, 205, 221
\l__akshar_tmpa_tl 15, 60, 64, 114, 116, 121, 123, 129, 131, 163, 165, 173, 175, 184, 186	P
\l__akshar_tmpb_seq 15, 58, 60, 61, 153, 160, 168, 178, 182, 183, 189, 195	prg commands:
\l__akshar_tmpb_tl 15, 61, 62	\prg_generate_conditional_ variant:Nnn 42
\l__akshar_tmpe_seq ... 15, 149, 155	\prg_new_conditional:Npnn 48
\l__akshar_tmpe_seq 15, 150, 158, 164, 174, 185	\prg_return_false: 55, 65, 70
\l__akshar_tmpe_seq ... 15, 151, 161	\prg_return_true: 65
__akshar_var_if_global 4	\ProvidesExplPackage 4
__akshar_var_if_global:NTF 46, 140, 196	R
\aksharStrChar 2, 3, 205	\RequirePackage 3
\aksharStrLen 1, 200	S
\aksharStrReplace 2, 221	scan commands:
\aksharStrReplace* 2	\scan_stop: 40, 218
B	seq commands:
bool commands:	\seq_clear:N 108, 152, 153, 160, 183
\bool_if:NTF 127	\seq_count:N ... 158, 203, 211, 217
\bool_if:nTF 50, 208	\seq_get_left:NN 60
\bool_new:N 13	\seq_get_right:NN 61
\bool_set_false:N 109, 132	\seq_gset_eq:NN 141, 197
\bool_set_true:N 124	\seq_item:Nn ... 164, 174, 185, 213
C	\seq_map_inline:Nn 146
cs commands:	\seq_map_variable:NNn 155
\cs_generate_variant:Nn 43, 44, 45, 144	\seq_new:N .. 14, 17, 18, 19, 20, 21
\cs_new:Npn 74, 106, 145, 147	\seq_pop_right:NN 114, 121, 129
\cs_split_function:N 51, 59	\seq_put_right:Nn 115, 122, 130, 135, 146, 168, 170, 178, 189, 191
E	\seq_set_eq:NN 142, 198
exp commands:	\seq_set_split:Nnn 43, 43, 58
\exp_last_unbraced:Nf 51, 59	\seq_use:Nn 226
I	T
\IfBooleanTF 223	tl commands:
int commands:	\tl_const:Nn 6, 7, 46, 47
\int_case:nnTF 77, 89, 97	\tl_if_eq:NNTF 62, 64, 119, 165, 175, 186
\int_compare:nNnTF 87, 157	\tl_if_in:Nn 42
\int_compare_p:nNn 210, 211	\tl_if_in:NnTF 42, 112
\int_eval:n 217	\tl_map_variable:NNn 110
	\tl_new:N 15, 16
	\tl_set:Nn 163, 173, 184
	\tl_to_str:n 46, 47
	token commands:
	\token_to_str:N 40, 54, 58, 69
	U
	use commands:
	\use_i:nnn 59
	\use_iii:nnn 51