

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 ϵ macros	1
2.2	expl3 functions	2
3	Implementation	2
3.1	Variable declarations	2
3.2	Messages	3
3.3	Utilities	3
3.4	The <code>\akshar_convert</code> function	5
3.5	Front-end \LaTeX 2 ϵ macros	6
	Index	6

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 \LaTeX 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.

\aksharStrChar

Return the n -th character of the token list.

```

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

```

2.2 expl3 functions

\akshar_convert:Nn

```
\akshar_convert:(cn|Nx|cx)
```

न, म, स्का, and र

3 Implementation

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

3.1 Variable declarations

These variables store the special characters we need to take into account:

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

(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` Some temporary variables.

`\l__akshar_tmpb_tl`

`\l__akshar_tmpa_seq`

`\l__akshar_tmpb_seq`

```
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
```

(End definition for `\l__akshar_tmpa_tl` and others.)

3.2 Messages

In `\akshar_convert`, the argument needs to be a sequence variable. There will be an error if it isn't.

```
19 \msg_new:nnnn { akshar } { err_not_a_sequence_variable }
20 { #1 ~ is ~ not ~ a ~ valid ~ LaTeX3 ~ sequence ~ variable. }
21 {
22   You ~ have ~ requested ~ me ~ to ~ assign ~ some ~ value ~ to ~ the ~
23   control ~ sequence ~ #1, ~ but ~ it ~ is ~ not ~ a ~ valid ~ sequence ~
24   variable. ~ Read ~ the ~ documentation ~ of ~ expl3 ~ for ~ more ~
25   information. ~ Proceed ~ and ~ I ~ will ~ pretend ~ that ~ #1 ~ is ~ a ~
26   local ~ sequence ~ variable ~ (beware ~ that ~ unexpected ~ behaviours ~
27   may ~ occur).
28 }
```

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.)

```
29 \msg_new:nnnn { akshar } { err_character_out_of_bound }
30 { Character ~ index ~ out ~ of ~ bound }
31 {
32   You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~ string ~
33   #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~ Make ~ sure ~ that ~
34   you ~ use ~ a ~ number ~ between ~ and ~ not ~ including ~ 0 ~ and ~ #3, ~
35   so ~ that ~ I ~ can ~ return ~ a ~ good ~ output. ~ Proceed ~ and ~ I ~
36   will ~ return ~ \token_to_str:N \scan_stop:.
37 }
```

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.

```
38 \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`.

```
39 \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

```

40 \cs_generate_variant:Nn \msg_error:nnn { nnx }
41 \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.

```

42 \tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
43 \tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
44 \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
45 {
46   \bool_if:nTF
47     { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
48     {
49       \msg_error:nnx { akshar } { err_not_a_sequence_variable }
50       { \token_to_str:N #1 }
51       \prg_return_false:
52     }
53     {
54       \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
55       { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
56       \seq_get_left:NN \l__akshar_tmpb_seq \l__akshar_tmpa_tl
57       \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
58       \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
59       {
60         \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
61         { \prg_return_true: } { \prg_return_false: }
62       }
63       {
64         \msg_error:nnx { akshar } { err_not_a_sequence_variable }
65         { \token_to_str:N #1 }
66         \prg_return_false:
67       }
68     }
69 }

```

(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.

```

70 \cs_new:Npn \__akshar_int_append_ordinal:n #1
71 {
72   #1
73   \int_case:nnF { #1 }
74   {
75     { 11 } { th }
76     { 12 } { th }
77     { 13 } { th }
78     { -11 } { th }
79     { -12 } { th }
80     { -13 } { th }
81   }
82   {
83     \int_compare:nNnTF { #1 } > { -1 }
84     {
85       \int_case:nnF { #1 - 10 * ( #1 / 10 ) }
86       {
87         { 1 } { st }
88         { 2 } { nd }
89         { 3 } { rd }
90       } { th }
91     }
92   }

```

```

93         \int_case:nnF { (- #1) - 10 * ((- #1) / 10) }
94         {
95             { 1 } { st }
96             { 2 } { nd }
97             { 3 } { rd }
98         } { th }
99     }
100 }
101 }

```

(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 is set to #1, which should be a sequence variable. The assignment is local.

```

\akshar_convert:cn
\akshar_convert:Nx
\akshar_convert:cx
102 \cs_new:Npn \akshar_convert:Nn #1 #2
103 {

```

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

```

104     \seq_clear:N \l__akshar_char_seq
105     \bool_set_false:N \l__akshar_prev_joining_bool

```

Loop through every token of the input.

```

106     \tl_map_variable:NNn {#2} \l__akshar_map_tl
107     {
108         \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
109         {

```

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.

```

110         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
111         \seq_put_right:Nx \l__akshar_char_seq
112         { \l__akshar_tmpa_tl \l__akshar_map_tl }
113     }
114     {
115         \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
116         {

```

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.

```

117         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
118         \seq_put_right:Nx \l__akshar_char_seq
119         { \l__akshar_tmpa_tl \l__akshar_map_tl }
120         \bool_set_true:N \l__akshar_prev_joining_bool
121     }
122     {

```

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

```

123         \bool_if:NNTF \l__akshar_prev_joining_bool
124         {
125             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
126             \seq_put_right:Nx \l__akshar_char_seq
127             { \l__akshar_tmpa_tl \l__akshar_map_tl }
128             \bool_set_false:N \l__akshar_prev_joining_bool
129         }
130         {
131             \seq_put_right:Nx \l__akshar_char_seq { \l__akshar_map_tl }
132         }
133     }
134 }
135 }

```

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

```

136   \__akshar_var_if_global:NTF #1
137   { \seq_gset_eq:NN #1 \l__akshar_char_seq }
138   { \seq_set_eq:NN #1 \l__akshar_char_seq }
139   }

```

Generate variants that might be helpful for some.

```

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

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

```

141 \NewExpandableDocumentCommand \aksharStrLen {m}
142 {
143   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
144   \seq_count:N \l__akshar_tmpa_seq
145 }

```

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

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

```

146 \NewExpandableDocumentCommand \aksharStrChar {mm}
147 {
148   \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
149   \bool_if:nTF
150   {
151     \int_compare_p:nNn { #2 } > { 0 } &&
152     \int_compare_p:nNn { #2 } < { 1 + \seq_count:N \l__akshar_tmpa_seq }
153   }
154   { \seq_item:Nn \l__akshar_tmpa_seq { #2 } }
155   {
156     \msg_error:nnnx { akshar } { err_character_out_of_bound }
157     { #1 } { \__akshar_int_append_ordinal:n { #2 } }
158     { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
159     \scan_stop:
160   }
161 }

```

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

```

162 </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	
akshar commands:	
<code>\akshar_convert</code>	1, 3, 5
<code>\akshar_convert:Nn</code>	2, <u>102</u> , 143, 148
akshar internal commands:	
<code>\l__akshar_char_seq</code>	<u>6</u> , <u>14</u> , 104, 110, 111, 117, 118, 125, 126, 131, 137, 138
<code>\c__akshar_diacritics_tl</code>	2, <u>6</u> , 108
<code>__akshar_int_append_ordinal:n</code>	<u>70</u> , 157
<code>\c__akshar_joining_tl</code>	2, <u>6</u> , 115
<code>\l__akshar_map_tl</code>	106, 108, 112, 115, 119, 127, 131
<code>\l__akshar_prev_joining_bool</code> ..	5, <u>13</u> , 105, 120, 123, 128
<code>\c__akshar_str_g_tl</code>	<u>42</u>
<code>\c__akshar_str_seq_tl</code>	<u>42</u>
<code>\l__akshar_tmpa_seq</code>	<u>15</u> , 143, 144, 148, 152, 154, 158
<code>\l__akshar_tmpa_tl</code>	<u>15</u> , 56, 60, 110, 112, 117, 119, 125, 127

\l__akshar_tmpb_seq .	15, 54, 56, 57	P	
\l__akshar_tmpb_tl	15, 57, 58	prg commands:	
__akshar_var_if_global	3	\prg_generate_conditional_-	
__akshar_var_if_global:NTF	42, 136	variant:Nnn	38
\aksharStrChar	2, 3, 146	\prg_new_conditional:Npnn	44
\aksharStrLen	1, 141	\prg_return_false:	51, 61, 66
		\prg_return_true:	61
		\ProvidesExplPackage	4
B			
bool commands:		R	
\bool_if:NTF	123	\RequirePackage	3
\bool_if:nTF	46, 149		
\bool_new:N	13	S	
\bool_set_false:N	105, 128	scan commands:	
\bool_set_true:N	120	\scan_stop:	36, 159
		seq commands:	
C		\seq_clear:N	104
cs commands:		\seq_count:N	144, 152, 158
\cs_generate_variant:Nn	39, 40, 41, 140	\seq_get_left:NN	56
\cs_new:Npn	70, 102	\seq_get_right:NN	57
\cs_split_function:N	47, 55	\seq_gset_eq:NN	137
		\seq_item:Nn	154
E		\seq_new:N	14, 17, 18
exp commands:		\seq_pop_right:NN	110, 117, 125
\exp_last_unbraced:Nf	47, 55	\seq_put_right:Nn	111, 118, 126, 131
		\seq_set_eq:NN	138
		\seq_set_split:Nnn	39, 39, 54
I		T	
int commands:		tl commands:	
\int_case:nnTF	73, 85, 93	\tl_const:Nn	6, 7, 42, 43
\int_compare:nNnTF	83	\tl_if_eq:NNTF	58, 60, 115
\int_compare_p:nNn	151, 152	\tl_if_in:Nn	38
\int_eval:n	158	\tl_if_in:NnTF	38, 108
		\tl_map_variable:NNn	106
M		\tl_new:N	15, 16
msg commands:		\tl_to_str:n	42, 43
\msg_error:nnn	40, 40, 49, 64	token commands:	
\msg_error:nnnnn	40, 41, 156	\token_to_str:N	36, 50, 54, 65
\msg_new:nnnn	19, 29		
		U	
N		use commands:	
\NewExpandableDocumentCommand	141, 146	\use_i:nnn	55
		\use_iii:nnn	47