

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	Utilities	3
3.3	The <code>\akshar_convert</code> function	4
3.4	Front-end $\LaTeX 2_{\epsilon}$ macros	5
	Index	5

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 की 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 $\LaTeX 2_{\epsilon}$ macros

<code>\aksharStrLen</code>	<code>\aksharStrLen {(token list)}</code>
	Return the number of Devanagari characters in the <code>{(token list)}</code> .

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

```

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.

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

`\akshar_convert:Nn (seq var) {(token list)}`

This function converts \langle token list \rangle to a sequence of characters, that sequence is stored in \langle seq var \rangle . The assignment to \langle seq var \rangle is local to the current T_EX 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: ा, ि, ी, ु, ू, े, ै, ो, ौ, ं, ः, ्र, ृ, ॄ, ॅ, ॱ, ॲ, ॳ, ॴ, ॵ, ॶ, ॷ, ॸ, ॹ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ.

```

6 \tl_const:Nn \c__akshar_joining_tl { ँ}
7 \tl_const:Nn \c__akshar_diacritics_tl
8 {
9   ा, ी, ु, ू, े, ै, ो, ौ, ॄ, ॱ, ॲ, ॳ, ॴ, ॵ, ॶ, ॷ, ॸ, ॹ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ,
10  ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ,
11  ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ, ॺ, ॻ, ॼ, ॽ, ॾ, ॿ
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_tmpl_tl`

`\l__akshar_tmpa_seq`

`\l__akshar_tmpl_seq`

```
15 \tl_new:N \l__akshar_tmpa_tl
```

```
16 \tl_new:N \l__akshar_tmpl_tl
```

```
17 \seq_new:N \l__akshar_tmpa_seq
```

```
18 \seq_new:N \l__akshar_tmpl_seq
```

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

3.2 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.

```
19 \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`.

```
20 \cs_generate_variant:Nn \seq_set_split:Nnn { Nxx }
```

(End definition for `\tl_if_in:NoTF` and `\seq_set_split:Nxx`.)

`__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`

```
21 \tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
22 \tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
23 \prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
24 {
25   \bool_if:nTF
26     { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
27     {
28       \iow_term:n { It ~ is ~ a ~ function! }
29       \prg_return_false:
30     }
31     {
32       \seq_set_split:Nxx \l__akshar_tmpl_seq { \token_to_str:N _ }
33       { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
34       \seq_get_left:NN \l__akshar_tmpl_seq \l__akshar_tmpa_tl
35       \seq_get_right:NN \l__akshar_tmpl_seq \l__akshar_tmpl_tl
36       \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpl_tl
37         {
38           \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
39             {
40               \iow_term:n { It ~ is ~ a ~ global ~ variable }
41               \prg_return_true:
42             }
43             {
44               \iow_term:n { It ~ is ~ a ~ local ~ variable }
45               \prg_return_false:

```

```

46         }
47     }
48     {
49         \iow_term:n { It ~ is ~ not ~ a ~ sequence ~ variable }
50         \prg_return_false:
51     }
52 }
53 }

```

(End definition for `\tl_if_in:NoTF` and others.)

3.3 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
54 \cs_new:Npn \akshar_convert:Nn #1 #2
55 {

```

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

```

56     \seq_clear:N \l__akshar_char_seq
57     \bool_set_false:N \l__akshar_prev_joining_bool

```

Loop through every token of the input.

```

58     \tl_map_variable:NNn {#2} \l__akshar_map_tl
59     {
60         \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}
61         {

```

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.

```

62         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
63         \seq_put_right:Nx \l__akshar_char_seq
64         { \l__akshar_tmpa_tl \l__akshar_map_tl }
65     }
66     {
67         \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
68         {

```

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.

```

69         \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
70         \seq_put_right:Nx \l__akshar_char_seq
71         { \l__akshar_tmpa_tl \l__akshar_map_tl }
72         \bool_set_true:N \l__akshar_prev_joining_bool
73     }
74     {

```

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

```

75         \bool_if:NNTF \l__akshar_prev_joining_bool
76         {
77             \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmpa_tl
78             \seq_put_right:Nx \l__akshar_char_seq
79             { \l__akshar_tmpa_tl \l__akshar_map_tl }
80             \bool_set_false:N \l__akshar_prev_joining_bool
81         }
82         {
83             \seq_put_right:Nx \l__akshar_char_seq { \l__akshar_map_tl }
84         }
85     }
86 }
87 }

```

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

```

88     \__akshar_var_if_global:NTF #1
89     { \seq_gset_eq:NN #1 \l__akshar_char_seq }
90     { \seq_set_eq:NN #1 \l__akshar_char_seq }
91 }

```

Generate variants that might be helpful for some.

```

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

```

(End definition for `\tl_if_in:NoTF` and others. These functions are documented on page ??.)

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

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

```

93 \NewExpandableDocumentCommand \aksharStrLen {m}
94 {
95     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
96     \seq_count:N \l__akshar_tmpa_seq
97 }

```

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

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

```

98 \NewExpandableDocumentCommand \aksharStrChar {mm}
99 {
100     \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
101     \seq_item:Nn \l__akshar_tmpa_seq {#2}
102 }

```

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

```

103 </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		<code>__akshar_var_if_global:NTF</code> 21 , 88
akshar commands:		<code>\aksharStrChar</code> 2 , 98
<code>\akshar_convert</code>	1 , 4	<code>\aksharStrLen</code> 1 , 93
<code>\akshar_convert:Nn</code> ..	2 , 54 , 95 , 100	
akshar internal commands:		B
<code>\l__akshar_char_seq</code> ..	5 , 14 , 56 , 62 , 63 , 69 , 70 , 77 , 78 , 83 , 89 , 90	bool commands:
<code>\c__akshar_diacritics_tl</code> ..	2 , 6 , 60	<code>\bool_if:NTF</code> 75
<code>\c__akshar_joining_tl</code>	2 , 6 , 67	<code>\bool_if:nTF</code> 25
<code>\l__akshar_map_tl</code> 58 , 60 , 64 , 67 , 71 , 79 , 83	<code>\bool_new:N</code> 13
<code>\l__akshar_prev_joining_bool</code> 4 , 13 , 57 , 72 , 75 , 80	<code>\bool_set_false:N</code> 57 , 80
<code>\c__akshar_str_g_tl</code>	21	<code>\bool_set_true:N</code> 72
<code>\c__akshar_str_seq_tl</code>	21	C
<code>\l__akshar_tmpa_seq</code> 15 , 95 , 96 , 100 , 101	cs commands:
<code>\l__akshar_tmpa_tl</code>	15 , 34 , 38 , 62 , 64 , 69 , 71 , 77 , 79	<code>\cs_generate_variant:Nn</code> 20 , 92
<code>\l__akshar_tmpb_seq</code> .	15 , 32 , 34 , 35	<code>\cs_new:Npn</code> 54
<code>\l__akshar_tmpb_tl</code>	15 , 35 , 36	<code>\cs_split_function:N</code> 26 , 33
<code>__akshar_var_if_global</code>	3	E
		exp commands:
		<code>\exp_last_unbraced:Nf</code> 26 , 33

I	\seq_gset_eq:NN	89
iow commands:	\seq_item:Nn	101
\iow_term:n	\seq_new:N	14, 17, 18
N	\seq_pop_right:NN	62, 69, 77
\NewExpandableDocumentCommand	\seq_put_right:Nn	63, 70, 78, 83
P	\seq_set_eq:NN	90
prg commands:	\seq_set_split:Nnn	20, 20, 32
\prg_generate_conditional_		
variant:Nnn	tl commands:	
\prg_new_conditional:Npnn	\tl_const:Nn	6, 7, 21, 22
\prg_return_false:	\tl_if_eq:NNTF	36, 38, 67
\prg_return_true:	\tl_if_in:Nn	19
\ProvidesExplPackage	\tl_if_in:NnTF	19, 60
R	\tl_map_variable:NNn	58
\RequirePackage	\tl_new:N	15, 16
S	\tl_to_str:n	21, 22
seq commands:	token commands:	
\seq_clear:N	\token_to_str:N	32
\seq_count:N	U	
\seq_get_left:NN	use commands:	
\seq_get_right:NN	\use_i:nnn	33
	\use_iii:nnn	26