

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	Utilities	3
3.3	The \akshar_convert function	3
3.4	Front-end LaTeX 2 _ε macros	4
	Index	4

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

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

(End definition for \l__akshar_char_seq.)

```
\l__akshar_tmp_tl  
\l__akshar_tmp_seq
```

Some temporary variables.

```
10 \tl_new:N \l__akshar_tmp_tl  
11 \seq_new:N \l__akshar_tmp_seq
```

(End definition for \l__akshar_tmp_tl and \l__akshar_tmp_seq.)

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**.

```
12 \prg_generate_conditional_variant:Nnn \tl_if_in:Nn { No } { TF }
```

(End definition for \tl_if_in:NoTF.)

3.3 The \akshar_convert function

```
\akshar_convert:Nn  
\akshar_convert:cn  
\akshar_convert:Nx  
\akshar_convert:cx
```

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.

```
13 \cs_new:Npn \akshar_convert:Nn #1 #2  
14 {
```

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

```
15 \seq_clear:N \l__akshar_char_seq  
16 \bool_set_false:N \l__akshar_prev_joining_bool
```

Loop through every token of the input.

```
17 \tl_map_variable:NNn {#2} \l__akshar_map_tl  
18 {  
19 \tl_if_in:NoTF \c__akshar_diacritics_tl {\l__akshar_map_tl}  
20 {
```

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.

```
21 \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl  
22 \seq_put_right:Nx \l__akshar_char_seq  
23 { \l__akshar_tmp_tl \l__akshar_map_tl }  
24 }  
25 {  
26 \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl  
27 {
```

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.

```
28 \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl  
29 \seq_put_right:Nx \l__akshar_char_seq  
30 { \l__akshar_tmp_tl \l__akshar_map_tl }  
31 \bool_set_true:N \l__akshar_prev_joining_bool  
32 }  
33 {
```

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

```
34 \bool_if:NTF \l__akshar_prev_joining_bool  
35 {  
36 \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl  
37 \seq_put_right:Nx \l__akshar_char_seq  
38 { \l__akshar_tmp_tl \l__akshar_map_tl }  
39 \bool_set_false:N \l__akshar_prev_joining_bool
```

```

40         }
41         {
42             \seq_put_right:Nx \l__akshar_char_seq { \l__akshar_map_tl }
43         }
44     }
45 }
46 }

```

Set #1 to `\l__akshar_char_seq`. The assignment is local, and I have not found a way to automatically pick `\seq_set_eq` or `\seq_gset_eq` based on the name of the sequence variable.

```

47     \seq_set_eq:NN #1 \l__akshar_char_seq
48 }

```

Generate variants that might be helpful for some.

```

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

```

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

3.4 Front-end \LaTeX macros

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

```

50 \NewExpandableDocumentCommand \aksharStrLen {m}
51 {
52     \akshar_convert:Nn \l__akshar_tmp_seq {#1}
53     \seq_count:N \l__akshar_tmp_seq
54 }

```

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

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

```

55 \NewExpandableDocumentCommand \aksharStrChar {mm}
56 {
57     \akshar_convert:Nn \l__akshar_tmp_seq {#1}
58     \seq_item:Nn \l__akshar_tmp_seq {#2}
59 }

```

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

```

60 \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	 10, 21, 23, 28, 30, 36, 38
akshar commands:		\aksharStrChar 2, 55
\akshar_convert 1, 3	\aksharStrLen 1, 50
\akshar_convert:Nn 2, 13, 52, 57	
akshar internal commands:		B
\l__akshar_char_seq 3, 9, 15, 21, 22, 28, 29, 36, 37, 42, 47		bool commands:
\c__akshar_diacritics_tl	... 2, 6, 19	\bool_if:NTF 34
\c__akshar_joining_tl 2, 6, 26	\bool_new:N 8
\l__akshar_map_tl 17, 19, 23, 26, 30, 38, 42	\bool_set_false:N 16, 39
\l__akshar_prev_joining_bool 3, 8, 16, 31, 34, 39	\bool_set_true:N 31
\l__akshar_tmp_seq	. 10, 52, 53, 57, 58	C
\l__akshar_tmp_tl	cs commands:
		\cs_generate_variant:Nn 49
		\cs_new:Npn 13

N		\seq_gset_eq	3
\NewExpandableDocumentCommand ..	50, 55	\seq_item:Nn	58
P		\seq_new:N	9, 11
prg commands:		\seq_pop_right:NN	21, 28, 36
\prg_generate_conditional_		\seq_put_right:Nn	22, 29, 37, 42
variant:Nnn	12	\seq_set_eq	3
\ProvidesExplPackage	4	\seq_set_eq:NN	47
R		T	
\RequirePackage	3	tl commands:	
S		\tl_const:Nn	6, 7
seq commands:		\tl_if_eq:NNTF	26
\seq_clear:N	15	\tl_if_in:Nn	12
\seq_count:N	53	\tl_if_in:NnTF	12, 19
		\tl_map_variable:NNn	17
		\tl_new:N	10