

# 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>User manual</b>	<b>1</b>
2.1	$\LaTeX$ 2 <sub>ε</sub> macros	1
2.2	expl3 functions	2
<b>3</b>	<b>Implementation</b>	<b>2</b>
	<b>Index</b>	<b>4</b>

## 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, which it isn’t.

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<sub>ε</sub> macros

---

`\aksharStrLen` `{<token list>}`

---

Return the number of Devanagari characters in the `<token list>`.

There are 8 characters in घघाघिघीघुघूघेघै.  
expl3 returns 15, which is wrong.

```
1 There are \aksharStrLen{ घघाघिघीघुघूघेघै} characters in घघाघिघीघुघूघेघै.\par
2 \ExplSyntaxOn
3 \pkg{expl3}-returns-\tl_count:n { घघाघिघीघुघूघेघै},-which-is-wrong.
4 \ExplSyntaxOff
```

\aksharStrChar

```
\aksharStrChar {⟨token list⟩} {⟨n⟩}
```

Return the  $n$ -th character of the token list.

4th character of घघाघिघीघुघूघेघै is घी.  
It is not घ.

```

1 4th character of घघाघिघीघुघूघैघै is \aksharStrChar{ घघाघिघीघुघूघै}{4}.\par
2 \ExplSyntaxOn
3 It-is-not-\tl_item:nn { घघाघिघीघुघूघैघै } {4}.
4 \ExplSyntaxOff

```

## 2.2 expl3 functions

This section assumes that you have a basic knowledge in  $\text{\LaTeX}$  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  $\text{\TeX}$  group.

घ, घा, घि, घी, घु, घू, घे, and घै

```

1 \ExplSyntaxOn
2 \akshar_convert:Nn \l_tmpa_seq { घघाघिघीघुघूघैघौ }
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)}
```

```
\c__akshar_joining_tl
\c__akshar_diacritics_tl
```

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

- \\_\_akshar\_joining\_tl is the “connecting” character ँ.
- \\_\_akshar\_diacritics\_tl is a list of all diacritics: िँिंः्र्ळ (they are ा, ि, ી, ੂ, ੄, ੇ, ੈ, ੋ, ੌ, ੍, ெ, ொ, ோ, ௌ, ௎, ௏, ௐ without the commas).

```
6 \tl_const:Nn \c__akshar_joining_tl { [] }
7 \tl_const:Nn \c__akshar_diacritics_tl { [ ] }
```

(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.

```
8 \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.

```
9 \seq_new:N \l__akshar_char_seq
```

(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.)

`\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`. This function is documented on page ??.)

`\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
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:NNTF \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 `\akshar_convert:Nn`. This function is documented on page 2.)

Time for some front-end macros that can be used directly in the  $\text{\LaTeX}2_{\epsilon}$  context.

**\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 1.)

```
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		P	
akshar commands:		prg commands:	
\akshar_convert:Nn	2, <u>13</u> , 52, 57	\prg_generate_conditional_	
akshar internal commands:		variant:Nnn	12
\l__akshar_char_seq	3,	\ProvidesExplPackage	4
9, 15, 21, 22, 28, 29, 36, 37, 42, 47			
\c__akshar_diacritics_tl	2, <u>6</u> , 19		
\c__akshar_joining_tl	2, <u>6</u> , 26		
\l__akshar_map_tl			
	17, 19, 23, 26, 30, 38, 42		
\l__akshar_prev_joining_bool			
	3, 8, 16, 31, 34, 39		
\l__akshar_tmp_seq	10, 52, 53, 57, 58		
\l__akshar_tmp_tl			
	10, 21, 23, 28, 30, 36, 38		
\aksharStrChar	1, <u>55</u>		
\aksharStrLen	1, <u>50</u>		
B		R	
bool commands:		\RequirePackage	3
\bool_if:NTF	34		
\bool_new:N	8		
\bool_set_false:N	16, 39		
\bool_set_true:N	31		
C		S	
cs commands:		seq commands:	
\cs_generate_variant:Nn	49	\seq_clear:N	15
\cs_new:Npn	13	\seq_count:N	53
		\seq_gset_eq	3
		\seq_item:Nn	58
		\seq_new:N	9, 11
		\seq_pop_right:NN	21, 28, 36
		\seq_put_right:Nn	22, 29, 37, 42
		\seq_set_eq	3
		\seq_set_eq:NN	47
N		T	
\NewExpandableDocumentCommand	50, 55	tl commands:	
		\tl_const:Nn	6, 7
		\tl_if_eq:NNTF	26
		\tl_if_in:Nn	12
		\tl_if_in:NnTF	12, 19
		\tl_map_variable:NNn	17
		\tl_new:N	10