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

2	Jser manual
	1.1 $ ext{ET}_{ extstyle E}$ X $2_{\mathcal{E}}$ macros
	2.2 expl3 functions
3	mplementation
	Variable declarations
	5.2 Utilities
	5.3 The \akshar_convert function
	5.4 Front-end $\LaTeX_{\mathcal{E}} X_{\mathcal{E}}$ macros

1 Introduction

Index

When dealing with processing strings in the Devanagari script, normal 图区 commands usually find some difficulties in distinguishing "normal" characters, like 事, and "special" characters, for example of . Let's consider this example code:

- ₁ \ExplSyntaxOn
- 2 tokens.
- 2 \tl_set:Nn \l_tmpa_tl { की}
- 3 \tl_count:N \l_tmpa_tl \c_space_token tokens.
- 4 \ExplSyntaxOff

The output is 2, but the number of characters in it is only one! The reason is quite simple: the compiler treats $\hat{}$ 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 $_{\mathcal{E}}$ macros

\aksharStrLen

\aksharStrLen {\langle token list\rangle}

Return the number of Devanagari characters in the \taken list\taken.

There are 4 characters in नमस्कार. expl3 returns 7, which is wrong.

- There are \aksharStrLen{ नमस्कार} characters in नमस्कार.\par
- 2 \ExplSyntaxOn
- ³ \pkg{expl3}~returns~\tl count:n { नमस्कार},~which~is~wrong.
- 4 \ExplSyntaxOff

\aksharStrChar

 $\arstropy \arstropy \ars$

Return the n-th character of the token list.

3rd character of नमस्कार is स्का. It is not स.

- ा 3rd character of नमस्कारांs \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 LTEX3 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 \langle seq var \rangle \langle \taken list \rangle \rangle

This function converts $\langle \text{token list} \rangle$ to a sequence of characters, that sequence is stored in $\langle \text{seq var} \rangle$. The assignment to $\langle \text{seq var} \rangle$ is local to the current T_EX group.

न, म, स्का, and र

- ∖ExplSyntaxOn
- 2 \akshar_convert:Nn \l_tmpa_seq { नमस्कार}
- $_{\mbox{\tiny 3}} \ \end{\tiny \mbox{\tiny $>$}} \ \end{\tiny \mbox{\tiny $>$$}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$$}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny $>$$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny \mbox{\tiny $>$$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny \mbox{\tiny \mbox{\tiny $>$$}$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny \mbox{\tiny \mbox{\tiny $>$$}$}}}} \ \end{\tiny \mbox{\tiny \mbox{\tiny$
- 4 \ExplSyntaxOff

3 Implementation

- ₁ ⟨@a=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}
- {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 \circ .
- \c__akshar_diacritics_tl is a list of all diacritics: 问閒i; (they are ा, ि, ी, ु, ू, ो, ो, ो, ं, ः, ॣ, ू, ŏ, ŏ without the commas).

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

7 \tl_const:Nn \c__akshar_diacritics_tl {000000000000}

(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.

s \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
\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.

```
\cs_new:Npn \akshar_convert:Nn #1 #2
```

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

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

Loop through every token of the input.

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

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.

```
\seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl
  \seq_put_right:Nx \l__akshar_char_seq
    { \l_akshar_tmp_tl \l_akshar_map_tl }
}
  \tl_if_eq:NNTF \l__akshar_map_tl \c__akshar_joining_tl
```

In this case, the character is the joining character, \circ . 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.

```
\seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl
                  \seq_put_right:Nx \l__akshar_char_seq
                    { \l_akshar_tmp_tl \l_akshar_map_tl }
30
                  \bool_set_true:N \l__akshar_prev_joining_bool
                }
```

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

```
\bool_if:NTF \l__akshar_prev_joining_bool
34
                      \seq_pop_right:NN \l__akshar_char_seq \l__akshar_tmp_tl
                      \seq_put_right:Nx \l__akshar_char_seq
37
                        { \l_akshar_tmp_tl \l_akshar_map_tl }
38
                      \bool_set_false:N \l__akshar_prev_joining_bool
```

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 $\ref{eq:NoTF}$.)

3.4 Front-end $\LaTeX 2_{\mathcal{E}}$ 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.

Index

60 (/package)

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.

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

N	\seq_gset_eq
\NewExpandableDocumentCommand 50, 55	\seq_item:Nn 58
	\seq_new:N 9, 11
P	\seq_pop_right:NN 21, 28, 36
prg commands:	\seq_put_right:Nn 22, 29, 37, 42
<pre>\prg_generate_conditional</pre>	\seq_set_eq
variant:Nnn 12	\seq_set_eq:NN 47
\ProvidesExplPackage 4	
	T
R	T tl commands:
· -	T tl commands: \tl_const:Nn
R	
R	\tl_const:Nn 6, 7
R \RequirePackage	\tl_const:Nn
R \RequirePackage	\tl_const:Nn