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 2.1 La Experimental | |
| 3 | 2.2 expl3 functions | 2 2 |
| Ind | dex | 4 |

1 Introduction

When dealing with processing strings in the Devanagari script, normal 图区 commands usually find some difficulties in distinguishing "normal" characters, like \overline{a} , and "special" characters, for example \overline{Q} or \overline{Q} . Let's consider this example code:

∖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 Later $\mathbf{Z}_{\mathcal{E}}$ macros

\aksharStrLen

\aksharStrLen {\langle token list\rangle}

Return the number of Devanagari characters in the (token list).

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

 $\arrowvert aksharStrChar {\langle token list \rangle} {\langle n \rangle}$

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)

 $\arrowvert:Nn \langle seq var \rangle \{\langle token list \rangle\}$

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

- 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}
- {Support for syllables in the Devanagari script (JV)}

\c__akshar_joining_tl
\c__akshar_diacritics tl

\c__akshar_joining_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 a list of all diacritics: িাাাৗৣী; (they are া, ি, া, ু, ্, ়, ঁ, 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.

% \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.

y \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.)
```

\tl_if_in:No<u>TF</u>

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 **o**nce.

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

(End definition for $\tloon tloon tloon to tloon the function is documented on page ??.)$

\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.

```
\seq_clear:N \l__akshar_char_seq
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.

In this case, the character is the joining character, \bigcirc . 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 }

\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.

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 \LaTeX 2_{\mathcal{E}} context.
 \aksharStrLen
                 Expands to the length of the string.
                  50 \NewExpandableDocumentCommand \aksharStrLen {m}
                      {
                  51
                         \akshar_convert:Nn \l__akshar_tmp_seq {#1}
                  52
                         \seq_count:N \l__akshar_tmp_seq
                  53
                  (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
                      {
                         \akshar_convert:Nn \l__akshar_tmp_seq {#1}
                         \seq_item:Nn \l__akshar_tmp_seq {#2}
                      }
                  (End definition for \aksharStrChar. This function is documented on page 1.)
```

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.

```
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, 6, 19
                                                R
  c_akshar_joining_tl \dots 2, 6, 26
                                 \RequirePackage ...... 3
  \l__akshar_map_tl .....
     S
  \l__akshar_prev_joining_bool ..
                                 seq commands:
     ..... 3, <u>8</u>, 16, 31, 34, 39
                                   \seq_clear:N ..... 15
  \label{local_section} $\local{local_section} 1_akshar_tmp_seq . $\underline{10}$, 52, 53, 57, 58
                                   \seq_count:N ..... 53
  \l__akshar_tmp_tl ......
                                   \seq_gset_eq ..... 3
     ..... <u>10,</u> 21, 23, 28, 30, 36, 38
\seq_item:Nn ..... 58
\seq_new:N ..... 9, 11
                                   \seq_pop_right:NN ..... 21, 28, 36
                                   \seq_put_right:Nn ..... 22, 29, 37, 42
bool commands:
                                   \seq_set_eq ..... 3
  \bool_if:NTF ..... 34
                                   \seq_set_eq:NN ..... 47
  \bool_new:N ..... 8
  \bool_set_false:N ..... 16, 39
                                                Т
  \bool_set_true:N ..... 31
                                 tl commands:
                                   \tl_const:Nn ..... 6, 7
cs commands:
                                   \tl_if_eq:NNTF ..... 26
  \cs_generate_variant:Nn ...... 49
                                   \tl_if_in:Nn ..... 12
  \cs_new:Npn ..... 13
                                   \tl_if_in:NnTF ..... 12, 19
                                   \tl_map_variable:NNn ..... 17
\NewExpandableDocumentCommand .. 50, 55
                                   \tl_new:N ..... 10
```