The akshar package

Vu Van Dung

Version 0.1 — 2020/05/17

Contents

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:

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { की}
\tl_count:N \l_tmpa_tl \c_space_token tokens.
\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 guide

3 Implementation

```
⟨@@=akshar⟩
⟨*package⟩
```

Declare the package. By loading fontspec, xparse, and in turn, expl3, are also loaded.

```
\RequirePackage{fontspec}
\ProvidesExplPackage {akshar} {2020/05/17} {0.1}
  {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:

- \c__akshar_joining_tl is the “connecting” character ्.
- \c__akshar_diacritics_tl is a list of all diacritics: ािीुूेैोौंःॄृॅँ (they are ा, ि, ी, ु, ू, े, ै, ो, ौ, ं, ः, ॄ, ृ, ॅ, ँ without the commas).

```
\tl_const:Nn \c__akshar_joining_tl { ्}
\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.

```
\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.

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

(End definition for `\l__akshar_char_seq`.)

`\l__akshar_tmp_tl` A temporary token list, used during the modification of the sequence.

```
\tl_new:N \l__akshar_tmp_tl
```

(End definition for `\l__akshar_tmp_tl`.)

`\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 **o**nce.

```
\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:n` This converts #1 to a sequence of true Devanagari characters. The sequence is `\l__akshar_char_seq`.

```
\cs_new:Npn \akshar_convert:n #1
 {
```

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 {#1} \l__akshar_map_tl
   {
    \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, ◌्. 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.

```
                  \bool_if:NTF \l__akshar_prev_joining_bool
                    {
                      \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_false:N \l__akshar_prev_joining_bool
                    }
                    {
                     \seq_put_right:Nx \l__akshar_char_seq { \l__akshar_map_tl }
                    }
                }
            }
        }
    }
```

(End definition for `\akshar_convert:n`. This function is documented on page **??**.)

```
⟨/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.