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

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, 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 2ε macros

`\aksharStrLen` `\aksharStrLen {⟨token list⟩}`

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
\ExplSyntaxOn
\pkg{expl3}~returns~\tl_count:n { नमस्कार},~which~is~wrong.
\ExplSyntaxOff
```

`\aksharStrChar`

`\aksharStrChar {⟨token list⟩} {⟨n⟩}`

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

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

```
3rd character of नमस्कारis \aksharStrChar{ नमस्कार}{3}.\par
\ExplSyntaxOn
It~is~not~\tl_item:nn { नमस्कार} {3}.
\ExplSyntaxOff
```

2.2 expl3 functions

This section assumes that you have a basic knowledge in LaTeX3 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 TeX group.

न, म, स्का, and र

```
\ExplSyntaxOn
\akshar_convert:Nn \l_tmpa_seq { नमस्कार}
\seq_use:Nnnn \l_tmpa_seq { ~and~ } { ,~ } { ,~and~ }
\ExplSyntaxOff
```

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)}
```

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 ◌्.
- `\c__akshar_diacritics_tl` is the list of all diacritics: ◌ा, ◌ि, ◌ी, ◌ु, ◌ू, ◌े, ◌ै, ◌ो, ◌ौ, ◌ं, ◌ः, ◌ॅ, ◌ॆ, ◌ॉ, ◌ॊ, ◌ॏ, ◌ऺ, ◌ऻ, ◌़, ◌ऽ, ◌ॎ, ◌ॐ, ◌॑, ◌॒, ◌॓, ◌॔, ◌ॕ, ◌ॖ, ◌ॗ, ◌क़, ◌ख़, ◌ग़, ◌ज़, ◌ड़, ◌ढ़, ◌फ़, ◌य़, ◌ॠ, ◌ॡ, ◌ॢ, ◌ॣ, ◌।, ◌॥, ◌०, ◌१, ◌२, ◌३, ◌४, ◌५, ◌६, ◌७, ◌८, ◌९, ◌अ, ◌उ, ◌क, ◌न, ◌प, ◌र, ◌वि, ◌ऽ.

```
\tl_const:Nn \c__akshar_joining_tl { ◌्}
\tl_const:Nn \c__akshar_diacritics_tl
  {
    ◌ा, ◌ि, ◌ी, ◌ु, ◌ू, ◌े, ◌ै, ◌ो, ◌ौ, ◌ं, ◌ः, ◌ॅ, ◌ॆ, ◌ॉ, ◌ॊ, ◌ॏ, ◌ऺ, ◌ऻ, ◌़, ◌ऽ,
    ◌ॎ, ◌ॐ, ◌॑, ◌॒, ◌॓, ◌॔, ◌ॕ, ◌ॖ, ◌ॗ, ◌क़, ◌ख़, ◌ग़, ◌ज़, ◌ड़, ◌ढ़, ◌फ़, ◌य़, ◌ॠ, ◌ॡ,
    ◌ॢ, ◌ॣ, ◌।, ◌॥, ◌०, ◌१, ◌२, ◌३, ◌४, ◌५, ◌६, ◌७, ◌८, ◌९, ◌अ, ◌उ, ◌क, ◌न, ◌प, ◌र, ◌वि, ◌ऽ
  }
```

(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_tmpa_tl
\l__akshar_tmpb_tl
\l__akshar_tmpa_seq
\l__akshar_tmpb_seq
Some temporary variables.

```
\tl_new:N \l__akshar_tmpa_tl
\tl_new:N \l__akshar_tmpb_tl
\seq_new:N \l__akshar_tmpa_seq
\seq_new:N \l__akshar_tmpb_seq
```

(End definition for *\l__akshar_tmpa_tl* and others.)

3.2 Messages

In `\akshar_convert`, the argument needs to be a sequence variable. There will be an error if it isn't.

```
\msg_new:nnnn { akshar } { err_not_a_sequence_variable }
  { #1 ~ is ~ not ~ a ~ valid ~ LaTeX3 ~ sequence ~ variable. }
  {
    You ~ have ~ requested ~ me ~ to ~ assign ~ some ~ value ~ to ~ the ~
    control ~ sequence ~ #1, ~ but ~ it ~ is ~ not ~ a ~ valid ~ sequence ~
    variable. ~ Read ~ the ~ documentation ~ of ~ expl3 ~ for ~ more ~
    information. ~ Proceed ~ and ~ I ~ will ~ pretend ~ that ~ #1 ~ is ~ a ~
    local ~ sequence ~ variable ~ (beware ~ that ~ unexpected ~ behaviours ~
    may ~ occur).
  }
```

In `\aksharStrChar`, we need to guard against accessing an 'out-of-bound' character (like trying to get the 8th character in a 5-character string.)

```
\msg_new:nnnn { akshar } { err_character_out_of_bound }
  { Character ~ index ~ out ~ of ~ bound }
  {
    You ~ are ~ trying ~ to ~ get ~ the ~ #2 ~ character ~ of ~ the ~ string ~
    #1. ~ However ~ that ~ character ~ doesn't ~ exist. ~ Make ~ sure ~ that ~
    you ~ use ~ a ~ number ~ between ~ and ~ not ~ including ~ 0 ~ and ~ #3, ~
    so ~ that ~ I ~ can ~ return ~ a ~ good ~ output. ~ Proceed ~ and ~ I ~
    will ~ return ~ \token_to_str:N \scan_stop:.
  }
```

3.3 Utilities

\tl_if_in:No*TF* 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.

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

(End definition for *\tl_if_in:NoTF*.)

\seq_set_split:Nxx A variant we will need in `\__akshar_var_if_global`.

```
\cs_generate_variant:Nn \seq_set_split:Nnn { Nxx }
```

(End definition for *\seq_set_split:Nxx*.)

\msg_error:nnx
\msg_error:nnnnx

Some variants of l3msg functions that we will need when issuing error messages.

```
\cs_generate_variant:Nn \msg_error:nnn { nnx }
\cs_generate_variant:Nn \msg_error:nnnnn { nnnnx }
```

(End definition for *\msg_error:nnx* and *\msg_error:nnnnx*.)

__akshar_var_if_global:N*TF*
\c__akshar_str_g_tl
\c__akshar_str_seq_tl

This conditional checks if #1 is a global sequence variable or not. In other words, it returns true iff #1 is a control sequence in the format \g_⟨name⟩_seq. If it is not a sequence variable, this function will (TODO) issue an error message.

```
\tl_const:Nx \c__akshar_str_g_tl { \tl_to_str:n {g} }
\tl_const:Nx \c__akshar_str_seq_tl { \tl_to_str:n {seq} }
\prg_new_conditional:Npnn \__akshar_var_if_global:N #1 { T, F, TF }
  {
    \bool_if:nTF
      { \exp_last_unbraced:Nf \use_iii:nnn { \cs_split_function:N #1 } }
      {
        \msg_error:nnx { akshar } { err_not_a_sequence_variable }
          { \token_to_str:N #1 }
        \prg_return_false:
      }
      {
        \seq_set_split:Nxx \l__akshar_tmpb_seq { \token_to_str:N _ }
          { \exp_last_unbraced:Nf \use_i:nnn { \cs_split_function:N #1 } }
        \seq_get_left:NN  \l__akshar_tmpb_seq \l__akshar_tmpa_tl
        \seq_get_right:NN \l__akshar_tmpb_seq \l__akshar_tmpb_tl
        \tl_if_eq:NNTF \c__akshar_str_seq_tl \l__akshar_tmpb_tl
          {
            \tl_if_eq:NNTF \c__akshar_str_g_tl \l__akshar_tmpa_tl
              { \prg_return_true: } { \prg_return_false: }
          }
          {
            \msg_error:nnx { akshar } { err_not_a_sequence_variable }
              { \token_to_str:N #1 }
            \prg_return_false:
          }
      }
  }
```

(End definition for *__akshar_var_if_global:NTF*, *\c__akshar_str_g_tl*, and *\c__akshar_str_seq_tl*.)

3.4 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
      {
        \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_tmpa_tl
            \seq_put_right:Nx \l__akshar_char_seq
```

```
            { \l__akshar_tmpa_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_tmpa_tl
              \seq_put_right:Nx \l__akshar_char_seq
                { \l__akshar_tmpa_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_tmpa_tl
                  \seq_put_right:Nx \l__akshar_char_seq
                    { \l__akshar_tmpa_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 }
                }
            }
        }
    }
```

Set `#1` to `\l__akshar_char_seq`. The package automatically determines whether the variable is a global one or a local one.

```
    \__akshar_var_if_global:NTF #1
      { \seq_gset_eq:NN #1 \l__akshar_char_seq }
      { \seq_set_eq:NN #1 \l__akshar_char_seq }
}
```

Generate variants that might be helpful for some.

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

(End definition for *\akshar_convert:Nn*. This function is documented on page 2.)

3.5 Front-end LaTeX 2ε macros

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

```
\NewExpandableDocumentCommand \aksharStrLen {m}
  {
    \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
    \seq_count:N \l__akshar_tmpa_seq
  }
```

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

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

```
\NewExpandableDocumentCommand \aksharStrChar {mm}
  {
    \akshar_convert:Nn \l__akshar_tmpa_seq {#1}
    \bool_if:nTF
      {
        \int_compare_p:nNn { #2 } > { 0 } &&
        \int_compare_p:nNn { #2 } < { 1 + \seq_count:N \l__akshar_tmpa_seq }
      }
      { \seq_item:Nn \l__akshar_tmpa_seq {#2} }
```

```
      {
        \msg_error:nnnnx { akshar } { err_character_out_of_bound }
          { #1 } { #2th }
          { \int_eval:n { 1 + \seq_count:N \l__akshar_tmpa_seq } }
        \scan_stop:
      }
  }
```

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

⟨/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.