The l3str package: manipulating strings of characters*

The LaTeX3 Project†

Released 2014/07/18

Contents

LaTeX3 provides a set of functions to manipulate token lists as strings of characters, ignoring the category codes of those characters.

String variables are simply specialised token lists, but by convention should be named with the suffix `...str`. Such variables should contain characters with category code 12 (other), except spaces, which have category code 10 (blank space). All the functions in this module which accept a token list argument first convert it to a string using `\tl_to_str:n` for internal processing, and will not treat a token list or the corresponding string representation differently.

Most expandable functions in this module come in three flavours:

- `\str_...:N`, which expect a token list or string variable as their argument;
- `\str_...:n`, taking any token list (or string) as an argument;

*This file describes v5226, last revised 2014/07/18.

†E-mail: latex-team@latex-project.org

- `\str_..._ignore_spaces:n`, which ignores any space encountered during the operation: these functions are typically faster than those which take care of escaping spaces appropriately.

1 Building strings

`\c_backslash_str`
`\c_left_brace_str`
`\c_right_brace_str`
`\c_hash_str`
`\c_tilde_str`
`\c_percent_str`

Constant strings, containing a single character token, with category code 12. Any character can be accessed as `\iow_char:N` `\`⟨*character*⟩.

`\str_new:N`
`\str_new:c`

`\str_new:N` ⟨*str var*⟩

Creates a new ⟨*str var*⟩ or raises an error if the name is already taken. The declaration is global. The ⟨*str var*⟩ will initially be empty.

`\str_const:Nn`
`\str_const:(Nx|cn|cx)`

`\str_const:Nn` ⟨*str var*⟩ {⟨*token list*⟩}

Creates a new constant ⟨*str var*⟩ or raises an error if the name is already taken. The value of the ⟨*str var*⟩ will be set globally to the ⟨*token list*⟩, converted to a string.

`\str_set:Nn`
`\str_set:(Nx|cn|cx)`
`\str_gset:Nn`
`\str_gset:(Nx|cn|cx)`

`\str_set:Nn` ⟨*str var*⟩ {⟨*token list*⟩}

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, and stores the result in ⟨*str var*⟩.

`\str_put_left:Nn`
`\str_put_left:(Nx|cn|cx)`
`\str_gput_left:Nn`
`\str_gput_left:(Nx|cn|cx)`

`\str_put_left:Nn` ⟨*str var*⟩ {⟨*token list*⟩}

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, and prepends the result to ⟨*str var*⟩. The current contents of the ⟨*str var*⟩ are not automatically converted to a string.

`\str_put_right:Nn`
`\str_put_right:(Nx|cn|cx)`
`\str_gput_right:Nn`
`\str_gput_right:(Nx|cn|cx)`

`\str_put_right:Nn` ⟨*str var*⟩ {⟨*token list*⟩}

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, and appends the result to ⟨*str var*⟩. The current contents of the ⟨*str var*⟩ are not automatically converted to a string.

2 Accessing the contents of a string

`\str_count:N` ⋆ `\str_count:n` ⋆ `\str_count_ignore_spaces:n` ⋆ `\str_count:n {⟨token list⟩}`

Leaves in the input stream the number of characters in the string representation of ⟨*token list*⟩, as an integer denotation. The functions differ in their treatment of spaces. In the case of `\str_count:N` and `\str_count:n`, all characters including spaces are counted. The `\str_count_ignore_spaces:n` function leaves the number of non-space characters in the input stream.

`\str_count_spaces:N` ⋆ `\str_count_spaces:n` ⋆ `\str_count_spaces:n {⟨token list⟩}`

Leaves in the input stream the number of space characters in the string representation of ⟨*token list*⟩, as an integer denotation. Of course, this function has no `_ignore_spaces` variant.

`\str_head:N` ⋆ `\str_head:n` ⋆ `\str_head_ignore_spaces:n` ⋆ `\str_head:n {⟨token list⟩}`

Converts the ⟨*token list*⟩ into a ⟨*string*⟩. The first character in the ⟨*string*⟩ is then left in the input stream, with category code "other". The functions differ if the first character is a space: `\str_head:N` and `\str_head:n` return a space token with category code 10 (blank space), while the `\str_head_ignore_spaces:n` function ignores this space character and leaves the first non-space character in the input stream. If the ⟨*string*⟩ is empty (or only contains spaces in the case of the `_ignore_spaces` function), then nothing is left on the input stream.

`\str_tail:N` ⋆ `\str_tail:n` ⋆ `\str_tail_ignore_spaces:n` ⋆ `\str_tail:n {⟨token list⟩}`

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, removes the first character, and leaves the remaining characters (if any) in the input stream, with category codes 12 and 10 (for spaces). The functions differ in the case where the first character is a space: `\str_tail:N` and `\str_tail:n` will trim only that space, while `\str_tail_ignore_spaces:n` removes the first non-space character and any space before it. If the ⟨*token list*⟩ is empty (or blank in the case of the `_ignore_spaces` variant), then nothing is left on the input stream.

\str_item:Nn ⋆ \str_item:nn {⟨*token list*⟩} {⟨*integer expression*⟩}
\str_item:nn ⋆
\str_item_ignore_spaces:nn ⋆

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, and leaves in the input stream the character in position ⟨*integer expression*⟩ of the ⟨*string*⟩, starting at 1 for the first (left-most) character. In the case of `\str_item:Nn` and `\str_item:nn`, all characters including spaces are taken into account. The `\str_item_ignore_spaces:nn` function skips spaces when counting characters. If the ⟨*integer expression*⟩ is negative, characters are counted from the end of the ⟨*string*⟩. Hence, −1 is the right-most character, *etc.*

\str_range:Nnn ⋆ \str_range:nnn {⟨*token list*⟩} {⟨*start index*⟩} {⟨*end index*⟩}
\str_range:nnn ⋆
\str_range_ignore_spaces:nnn ⋆

Converts the ⟨*token list*⟩ to a ⟨*string*⟩, and leaves in the input stream the characters from the ⟨*start index*⟩ to the ⟨*end index*⟩ inclusive. Positive ⟨*indices*⟩ are counted from the start of the string, 1 being the first character, and negative ⟨*indices*⟩ are counted from the end of the string, −1 being the last character. If either of ⟨*start index*⟩ or ⟨*end index*⟩ is 0, the result is empty. For instance,

```
\iow_term:x { \str_range:nnn { abcdef } { 2 } { 5 } }
\iow_term:x { \str_range:nnn { abcdef } { -4 } { -1 } }
\iow_term:x { \str_range:nnn { abcdef } { -2 } { -1 } }
\iow_term:x { \str_range:nnn { abcdef } { 0 } { -1 } }
```

will print `bcd`, `cdef`, `ef`, and an empty line to the terminal.

3 String conditionals

\str_if_eq_p:nn ⋆ \str_if_eq_p:nn {⟨tl_1⟩} {⟨tl_2⟩}
\str_if_eq_p:(Vn|on|no|nV|VV) ⋆ \str_if_eq:nnTF {⟨tl_1⟩} {⟨tl_2⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\str_if_eq:nnTF ⋆
\str_if_eq:(Vn|on|no|nV|VV)TF ⋆

Compares the string representations of the two ⟨*token lists*⟩ on a character by character basis, and is `true` if the two lists contain the same characters in the same order. Thus for example

```
\str_if_eq_p:no { abc } { \tl_to_str:n { abc } }
```

is logically `true`.

\str_if_eq_x_p:nn ⋆
\str_if_eq_x:nnTF ⋆

New: 2012-06-05

```
\str_if_eq_x_p:nn {⟨tl1⟩} {⟨tl2⟩}
\str_if_eq_x:nnTF {⟨tl1⟩} {⟨tl2⟩} {⟨true code⟩} {⟨false code⟩}
```

Compares the full expansion of two ⟨*token lists*⟩ on a character by character basis, and is `true` if the two lists contain the same characters in the same order. Thus for example

```
\str_if_eq_x_p:nn { abc } { \tl_to_str:n { abc } }
```

is logically `true`.

\str_case:nnTF ⋆
\str_case:onTF ⋆

New: 2013-07-24

```
\str_case:nnTF {⟨test string⟩}
  {
    {⟨string case1⟩} {⟨code case1⟩}
    {⟨string case2⟩} {⟨code case2⟩}
    ...
    {⟨string casen⟩} {⟨code casen⟩}
  }
  {⟨true code⟩}
  {⟨false code⟩}
```

This function compares the ⟨*test string*⟩ in turn with each of the ⟨*string cases*⟩. If the two are equal (as described for `\str_if_eq:nnTF` then the associated ⟨*code*⟩ is left in the input stream. If any of the cases are matched, the ⟨*true code*⟩ is also inserted into the input stream (after the code for the appropriate case), while if none match then the ⟨*false code*⟩ is inserted. The function `\str_case:nn`, which does nothing if there is no match, is also available.

\str_case_x:nnTF ⋆

New: 2013-07-24

```
\str_case_x:nnn {⟨test string⟩}
  {
    {⟨string case1⟩} {⟨code case1⟩}
    {⟨string case2⟩} {⟨code case2⟩}
    ...
    {⟨string casen⟩} {⟨code casen⟩}
  }
  {⟨true code⟩}
  {⟨false code⟩}
```

This function compares the full expansion of the ⟨*test string*⟩ in turn with the full expansion of the ⟨*string cases*⟩. If the two full expansions are equal (as described for `\str_if_eq:nnTF` then the associated ⟨*code*⟩ is left in the input stream. If any of the cases are matched, the ⟨*true code*⟩ is also inserted into the input stream (after the code for the appropriate case), while if none match then the ⟨*false code*⟩ is inserted. The function `\str_case_x:nn`, which does nothing if there is no match, is also available. The ⟨*test string*⟩ is expanded in each comparison, and must always yield the same result: for example, random numbers must not be used within this string.

4 Viewing strings

\str_show:N
\str_show:(c|n)

\str_show:N ⟨*tl var*⟩

Displays the content of the ⟨*str var*⟩ on the terminal.

5 Scratch strings

\l_tmpa_str
\l_tmpb_str

Scratch strings for local assignment. These are never used by the kernel code, and so are safe for use with any LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\g_tmpa_str
\g_tmpb_str

Scratch strings for global assignment. These are never used by the kernel code, and so are safe for use with any LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

6 Internal l3str functions

__str_to_other:n ⋆

__str_to_other:n {⟨*token list*⟩}

Converts the ⟨*token list*⟩ to a ⟨*other string*⟩, where spaces have category code "other". This function can be f-expanded without fear of losing a leading space, since spaces do not have category code 10 in its result. It takes a time quadratic in the character count of the string, but there exist non-expandable ways to reach linear time.

__str_count_unsafe:n ⋆

__str_count_unsafe:n {⟨*other string*⟩}

This function expects an argument that is entirely made of characters with category "other", as produced by __str_to_other:n. It leaves in the input stream the number of character tokens in the ⟨*other string*⟩, faster than the analogous \str_count:n function.

__str_range_unsafe:nnn ⋆

__str_range_unsafe:nnn {⟨*other string*⟩} {⟨*start index*⟩} {⟨*end index*⟩}

Identical to \str_range:nnn except that the first argument is expected to be entirely made of characters with category "other", as produced by __str_to_other:n, and the result is also an ⟨*other string*⟩.

7 Possible additions to l3str

Semantically correct copies of some tl functions.

- \c_space_str
- \str_clear:N, \str_gclear:N, \str_clear_new:N, \str_gclear_new:N.

- `\str_concat:NNN`, `\str_gconcat:NNN`
- `\str_set_eq:NN`, `\str_gset_eq:NN`
- `\str_if_empty:NTF`, `\str_if_empty_p:N`
- `\str_if_exist:NTF`, `\str_if_exist_p:N`
- `\str_use:N`

Some functions that are not copies of `tl` functions.

- `\str_if_blank:NTF`, `\str_if_blank_p:N`.
- `\str_map_inline:Nn`, `\str_map_function:NN`, `\str_map_variable:NNn`, and `:n` analogs.
- Expandable `\str_if_in:nnTF`?
- `\str_if_head_eq:nNTF`, `\str_if_head_eq_p:nN`
- `\str_if_numeric/decimal/integer:n`, perhaps in l3fp?

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.