

The **l3str** package: manipulating strings of characters*

The L^AT_EX3 Project[†]

Released 2014/05/26

Contents

1	Building strings	2
2	Accessing the contents of a string	2
3	String conditionals	4
4	Viewing strings	5
5	Scratch strings	6
6	Internal l3str functions	6
7	Possible additions to l3str	6

Index **7**

L^AT_EX3 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 v4889, last revised 2014/05/26.

[†]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

<hr/> <code>\c_backslash_str</code> <code>\c_left_brace_str</code> <code>\c_right_brace_str</code> <code>\c_hash_str</code> <code>\c_tilde_str</code> <code>\c_percent_str</code> <hr/>	<p>Constant strings, containing a single character token, with category code 12. Any character can be accessed as <code>\iow_char:N \⟨character⟩</code>.</p>
<hr/> <code>\str_new:N</code> <code>\str_new:c</code> <hr/>	<p><code>\str_new:N ⟨str var⟩</code></p> <p>Creates a new <code>⟨str var⟩</code> or raises an error if the name is already taken. The declaration is global. The <code>⟨str var⟩</code> will initially be empty.</p>
<hr/> <code>\str_const:Nn</code> <code>\str_const:(Nx cn cx)</code> <hr/>	<p><code>\str_const:Nn ⟨str var⟩ {⟨token list⟩}</code></p> <p>Creates a new constant <code>⟨str var⟩</code> or raises an error if the name is already taken. The value of the <code>⟨str var⟩</code> will be set globally to the <code>⟨token list⟩</code>, converted to a string.</p>
<hr/> <code>\str_set:Nn</code> <code>\str_set:(Nx cn cx)</code> <code>\str_gset:Nn</code> <code>\str_gset:(Nx cn cx)</code> <hr/>	<p><code>\str_set:Nn ⟨str var⟩ {⟨token list⟩}</code></p> <p>Converts the <code>⟨token list⟩</code> to a <code>⟨string⟩</code>, and stores the result in <code>⟨str var⟩</code>.</p>
<hr/> <code>\str_put_left:Nn</code> <code>\str_put_left:(Nx cn cx)</code> <code>\str_gput_left:Nn</code> <code>\str_gput_left:(Nx cn cx)</code> <hr/>	<p><code>\str_put_left:Nn ⟨str var⟩ {⟨token list⟩}</code></p> <p>Converts the <code>⟨token list⟩</code> to a <code>⟨string⟩</code>, and prepends the result to <code>⟨str var⟩</code>. The current contents of the <code>⟨str var⟩</code> are not automatically converted to a string.</p>
<hr/> <code>\str_put_right:Nn</code> <code>\str_put_right:(Nx cn cx)</code> <code>\str_gput_right:Nn</code> <code>\str_gput_right:(Nx cn cx)</code> <hr/>	<p><code>\str_put_right:Nn ⟨str var⟩ {⟨token list⟩}</code></p> <p>Converts the <code>⟨token list⟩</code> to a <code>⟨string⟩</code>, and appends the result to <code>⟨str var⟩</code>. The current contents of the <code>⟨str var⟩</code> are not automatically converted to a string.</p>

2 Accessing the contents of a string

<code>\str_count:N</code>	★	<code>\str_count:n {⟨token list⟩}</code>
<code>\str_count:n</code>	★	
<code>\str_count_ignore_spaces:n</code>	★	

Leaves in the input stream the number of characters in the string representation of $\langle token list \rangle$, 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.

<code>\str_count_spaces:N</code>	★	<code>\str_count_spaces:n {⟨token list⟩}</code>
<code>\str_count_spaces:n</code>	★	

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

<code>\str_head:N</code>	★	<code>\str_head:n {⟨token list⟩}</code>
<code>\str_head:n</code>	★	
<code>\str_head_ignore_spaces:n</code>	★	

Converts the $\langle token list \rangle$ into a $\langle string \rangle$. The first character in the $\langle string \rangle$ 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 $\langle string \rangle$ is empty (or only contains spaces in the case of the `_ignore_spaces` function), then nothing is left on the input stream.

<code>\str_tail:N</code>	★	<code>\str_tail:n {⟨token list⟩}</code>
<code>\str_tail:n</code>	★	
<code>\str_tail_ignore_spaces:n</code>	★	

Converts the $\langle token list \rangle$ to a $\langle string \rangle$, 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 $\langle token list \rangle$ is empty (or blank in the case of the `_ignore_spaces` variant), then nothing is left on the input stream.

<code>\str_item:Nn</code>	★	<code>\str_item:nn {⟨token list⟩} {⟨integer expression⟩}</code>
<code>\str_item:nn</code>	★	
<code>\str_item_ignore_spaces:nn</code>	★	

Converts the $\langle token list \rangle$ to a $\langle string \rangle$, and leaves in the input stream the character in position $\langle integer expression \rangle$ of the $\langle string \rangle$, 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 $\langle integer expression \rangle$ is negative, characters are counted from the end of the $\langle string \rangle$. Hence, -1 is the right-most character, *etc.*

<code>\str_range:Nnn</code>	★	<code>\str_range:nnn {⟨token list⟩} {⟨start index⟩} {⟨end index⟩}</code>
<code>\str_range:nnn</code>	★	
<code>\str_range_ignore_spaces:nnn</code>	★	

Converts the $\langle token list \rangle$ to a $\langle string \rangle$, and leaves in the input stream the characters from the $\langle start index \rangle$ to the $\langle end index \rangle$ inclusive. Positive $\langle indices \rangle$ are counted from the start of the string, 1 being the first character, and negative $\langle indices \rangle$ are counted from the end of the string, -1 being the last character. If either of $\langle start index \rangle$ or $\langle end index \rangle$ 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

<code>\str_if_eq_p:nn</code>	★	<code>\str_if_eq_p:nn {⟨t₁⟩} {⟨t₂⟩}</code>
<code>\str_if_eq_p:(Vn on no nV VV)</code>	★	<code>\str_if_eq:nnTF {⟨t₁⟩} {⟨t₂⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\str_if_eq:nnTF</code>	★	
<code>\str_if_eq:(Vn on no nV VV)TF</code>	★	

Compares the string representations of the two $\langle token lists \rangle$ 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**.

<code>\str_if_eq_x_p:nn</code> ★ <code>\str_if_eq_x:nnTF</code> ★	<code>\str_if_eq_x_p:nn {<tl₁>} {<tl₂>}</code> <code>\str_if_eq_x:nnTF {<tl₁>} {<tl₂>} {<true code>} {<false code>}</code>
--	---

New: 2012-06-05

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**.

<code>\str_case:nnTF</code> ★ <code>\str_case:onTF</code> ★	<code>\str_case:nnTF {<test string>}</code> <code>{</code> <code> {<string case₁>} {<code case₁>}</code> <code> {<string case₂>} {<code case₂>}</code> <code> ...</code> <code> {<string case_n>} {<code case_n>}</code> <code>}</code> <code>{<true code>}</code> <code>{<false code>}</code>
--	--

New: 2013-07-24

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.

<code>\str_case_x:nnTF</code> ★	<code>\str_case_x:nnn {<test string>}</code> <code>{</code> <code> {<string case₁>} {<code case₁>}</code> <code> {<string case₂>} {<code case₂>}</code> <code> ...</code> <code> {<string case_n>} {<code case_n>}</code> <code>}</code> <code>{<true code>}</code> <code>{<false code>}</code>
---------------------------------	---

New: 2013-07-24

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

<hr/> <code>\str_show:N</code> <hr/>	<code>\str_show:N <tl var></code>
<code>\str_show:(c n)</code> <hr/>	Displays the content of the <code><str var></code> on the terminal.

5 Scratch strings

<hr/> <code>\l_tmpa_str</code> <code>\l_tmpb_str</code> <hr/>	Scratch strings for local assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
--	---

<hr/> <code>\g_tmpa_str</code> <code>\g_tmpb_str</code> <hr/>	Scratch strings for global assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-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

<hr/> <code>__str_to_other:n</code> ★ <hr/>	<code>__str_to_other:n {<token list>}</code> Converts the <code><token list></code> to a <code><other string></code> , 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.
--	--

<hr/> <code>__str_count_unsafe:n</code> ★ <hr/>	<code>__str_count_unsafe:n {<other string>}</code> This function expects an argument that is entirely made of characters with category “other”, as produced by <code>__str_to_other:n</code> . It leaves in the input stream the number of character tokens in the <code><other string></code> , faster than the analogous <code>\str_count:n</code> function.
--	---

<hr/> <code>__str_range_unsafe:nnn</code> ★ <hr/>	<code>__str_range_unsafe:nnn {<other string>} {<start index>} {<end index>}</code> Identical to <code>\str_range:nnn</code> except that the first argument is expected to be entirely made of characters with category “other”, as produced by <code>__str_to_other:n</code> , and the result is also an <code><other string></code> .
--	---

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:N`, `\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.

Symbols		S
_str_count_unsafe:n	6	\str_case:nnTF 5
_str_range_unsafe:nnn	6	\str_case_x:nnTF 5
_str_to_other:n	6	\str_const:Nn 2
		\str_count:N 2
		\str_count_ignore_spaces:n 2
C		\str_count_spaces:N 3
\c_backslash_str	2	\str_gput_left:Nn 2
\c_hash_str	2	\str_gput_right:Nn 2
\c_left_brace_str	2	\str_gset:Nn 2
\c_percent_str	2	\str_head:N 3
\c_right_brace_str	2	\str_head_ignore_spaces:n 3
\c_tilde_str	2	\str_if_eq:nnTF 4
		\str_if_eq_p:nn 4
		\str_if_eq_x:nnTF 4
		\str_if_eq_x_p:nn 4
G		\str_item:Nn 3
\g_tmpa_str	6	\str_item_ignore_spaces:nn 3
\g_tmpb_str	6	\str_new:N 2
		\str_put_left:Nn 2
L		
\l_tmpa_str	6	
\l_tmpb_str	6	

<code>\str_put_right:Nn</code>	2	<code>\str_show:N</code>	5
<code>\str_range:Nnn</code>	4	<code>\str_tail:N</code>	3
<code>\str_range_ignore_spaces:nnn</code>	4	<code>\str_tail_ignore_spaces:n</code>	3
<code>\str_set:Nn</code>	2		