1. Copyright.

2. *angled_string* **Thread.**
It recognizes character sequences of example < my disc > typically used in a c++ include statement or in my case files to be read either thru my own include statements or from the command line. *esc_seq* thread evaluates the c++ literal character escape sequences buried inside the angled brackets. The escape sequences cover the backslash variety — octal, hex, and character, and those pesky unicode types. Nothing is done with them at present — only their syntax is recognized. The *esc_seq* terminal returned contains the character string parsed. The *angled_string* returned contains the enclosed character string without the bounding angle brackets. So < my disc > becomes "my disc" without the quotes. The **empty string** is tollerated. It is left to the calling grammar to deal with it.

Errors:
Caused by either an invalid character escape sequence or an improper closing of the *angled_string* — an end of line or file etc before possibly omitted closing ">" bracket. These two conditions are indicated by the 2 error terminals *Err_bad_esc* and *Err_bad_eos*.

Returned T: *T_angled_string*

3. Fsm Cangled_string class.

4. Cangled_string constructor directive.

⟨ Cangled_string constructor directive 4 ⟩ ≡
```
ddd_idx_ = 0;
ddd_[ddd_idx_] = 0;
```

5. Cangled_string op directive.

⟨ Cangled_string op directive 5 ⟩ ≡
```
ddd_idx_ = 0;
ddd_[ddd_idx_] = 0;
```

6. Cangled_string user-declaration directive.

⟨ Cangled_string user-declaration directive 6 ⟩ ≡
```
public: char ddd_[1024 * 32];
  int ddd_idx_;
  void copy_str_into_buffer(std::string * Str);
  void copy_kstr_into_buffer(const char *Str);
```

7. Cangled_string user-implementation directive.

⟨ Cangled_string user-implementation directive 7 ⟩ ≡
```
  void Cangled_string::copy_str_into_buffer(std::string * Str)
  {
    const char *y = Str→c_str( );
    int x(0);
    for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
    ddd_[ddd_idx_] = 0;
  }
```

8. *copy_kstr_into_buffer*.

⟨ More code 8 ⟩ ≡

```
void Cangled_string::copy_kstr_into_buffer(const char *Str)
{
  const char *y = Str;
  int x(0);
  for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
  ddd_[ddd_idx_] = 0;
}
```

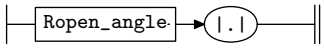
9. Cangled_string user-prefix-declaration directive.

⟨ Cangled_string user-prefix-declaration directive 9 ⟩ ≡

```
#include "esc_seq.h"
```

10. *Rangled_string* **rule.**

Rangled_string

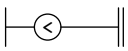


⟨ Rangled_string subrule 1 op directive 10 ⟩ ≡

```
Cangled_string *fsm = ( Cangled_string * ) rule_info__.parser__->fsm_tbl__;
CAbs_lr1_sym *sym = new T_angled_string((const char *) &fsm->ddd_);
sym->set_rc(*rule_info__.parser__->start_token__, __FILE__, __LINE__);
RSVP(sym);
```

11. *Ropen_angle* **rule.**

```
Ropen_angle
```



```
⟨ Ropen_angle subrule 1 op directive 11 ⟩ ≡
  Cangled_string * fsm = ( Cangled_string * ) rule_info__.parser__→fsm_tbl__;
loop:
  switch (rule_info__.parser__→current_token( )→enumerated_id__) {
  case T_Enum :: T_raw_lf_: goto overrun;
  case T_Enum :: T_raw_cr_: goto overrun;
  case T_Enum :: T_LR1_eog_: goto overrun;
  case T_Enum :: T_raw_gt_than_: goto closestr;
  case T_Enum :: T_raw_back_slash_: goto escseq;
  default: goto other;
  }
closestr:
  {     /* end of string */
    rule_info__.parser__→get_next_token( );
    return;     /* end of angled string */
  }
overrun:
  {
    CAbs_lr1_sym * sym = new Err_bad_eos;
    sym→set_rc(*rule_info__.parser__→start_token__, __FILE__, __LINE__);
    RSVP(sym);
    rule_info__.parser__→set_stop_parse(true);
    return;
  }
  escseq: {     /* what type of escape */
  using namespace NS_esc_seq;
  Parser :: parse_result result = rule_info__.parser__→start_manually_parallel_parsing(ITH_esc_seq.thd_id__);
  if (result ≡ Parser :: erred) {     /* in this case, it will not happen: here for education */
    rule_info__.parser__→set_abort_parse(true);
    return;
  }     /* process returned token */
  Caccept_parse & accept_parm = *rule_info__.parser__→arbitrated_token__;
  CAbs_lr1_sym * rtn_tok = accept_parm.accept_token__;
  int id = rtn_tok→enumerated_id__;
  accept_parm.accept_token__ = 0;
  if (id ≠ T_Enum :: T_T_esc_seq_) {
    RSVP(rtn_tok);
    return;
  }
  T_esc_seq * finc = ( T_esc_seq * ) (rtn_tok);
  fsm→copy_str_into_buffer(finc→esc_data( ));
  rule_info__.parser__→override_current_token(*accept_parm.la_token__, accept_parm.la_token_pos__);
  delete finc;
  goto loop; } ;
other:
  {
    fsm→copy_kstr_into_buffer(rule_info__.parser__→current_token( )→id__);
```

```
    rule_info__.parser__→get_next_token( );
    goto loop;
  }
```

12. First Set Language for O_2^{linker}.

```
/*
 File: angled_string.fsc
 Date and Time: Fri Jan  2 15:33:27 2015
*/
transitive   n
grammar-name "angled_string"
name-space   "NS_angled_string"
thread-name  "TH_angled_string"
monolithic   n
file-name    "angled_string.fsc"
no-of-T      569
list-of-native-first-set-terminals 1
   raw_less_than
end-list-of-native-first-set-terminals
list-of-transitive-threads 0
end-list-of-transitive-threads
list-of-used-threads 0
end-list-of-used-threads
fsm-comments
"Angled string lexer: < ... > with c type escape sequences."
```

13. Lr1 State Network.

⇒ State: 1 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
c	Ropen_angle		2	1	1	<		1	2	2	
c	Rangled_string		1	1	1	Ropen_angle |.|		1	3	4	

⇒$^{<}$ State: 2 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Ropen_angle		2	1	2			1	0	2	1

⇒$^{Ropen\_angle}$ State: 3 state type: ^{s}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rangled_string		1	1	2	|.|		1	4	4	

⇒\|.\| State: 4 state type: ^{r}

	← rule	→	R#	sr#	Po	← subrule element	→	Brn	Gto	Red	LA
t	Rangled_string		1	1	3			1	0	4	2

14. Index.

`|.|`: 10.
`__FILE__`: 10, 11.
`__LINE__`: 10, 11.
accept_parm: 11.
accept_token__: 11.
angled_string: 2.
arbitrated_token__: 11.
c_str: 7.
CAbs_lr1_sym: 10, 11.
Caccept_parse: 11.
Cangled_string: 7, 8, 10, 11.
closestr: 11.
copy_kstr_into_buffer: 6, 8, 11.
copy_str_into_buffer: 6, 7, 11.
current_token: 11.
*ddd_*: 4, 5, 6, 7, 8, 10.
*ddd_idx_*: 4, 5, 6, 7, 8.
enumerated_id__: 11.
Err_bad_eos: 2, 11.
Err_bad_esc: 2.
erred: 11.
esc_data: 11.
esc_seq: 2.
escseq: 11.
finc: 11.
fsm: 10, 11.
fsm_tbl__: 10, 11.
get_next_token: 11.
id: 11.
id__: 11.
ITH_esc_seq: 11.
la_token__: 11.
la_token_pos__: 11.
loop: 11.
NS_esc_seq: 11.
other: 11.
override_current_token: 11.
overrun: 11.
parse_result: 11.
Parser: 11.
parser__: 10, 11.
Rangled_string: 10.
result: 11.
`Ropen_angle`: 10.
Ropen_angle: 11.
`RSVP`: 10, 11.
rtn_tok: 11.
rule_info__: 10, 11.
set_abort_parse: 11.
set_rc: 10, 11.
set_stop_parse: 11.
start_manually_parallel_parsing: 11.
start_token__: 10, 11.
std: 6, 7.
Str: 6, 7, 8.
string: 6, 7.
sym: 10, 11.
T_angled_string: 2, 10.
T_Enum: 11.
T_esc_seq: 11.
*T_LR1_eog_*: 11.
*T_raw_back_slash_*: 11.
*T_raw_cr_*: 11.
*T_raw_gt_than_*: 11.
*T_raw_lf_*: 11.
*T_T_esc_seq_*: 11.
thd_id__: 11.
true: 11.
x: 7, 8.
y: 7, 8.

⟨ Cangled_string constructor directive 4 ⟩
⟨ Cangled_string op directive 5 ⟩
⟨ Cangled_string user-declaration directive 6 ⟩
⟨ Cangled_string user-implementation directive 7 ⟩
⟨ Cangled_string user-prefix-declaration directive 9 ⟩
⟨ More code 8 ⟩
⟨ Rangled_string subrule 1 op directive 10 ⟩
⟨ Ropen_angle subrule 1 op directive 11 ⟩

angled_string Grammar

Date: January 2, 2015 at 15:34

File: angled_string.lex Ns: NS_angled_string

Version: 1.0 Debug: false

Grammar Comments: Type: Thread

Angled string lexer: < ... > with c type escape sequences.

1 element(s) in Lookahead Expression below

eolr

	Section	Page
Copyright	1	1
angled_string **Thread**	2	2
Fsm Cangled_string class	3	2
Cangled_string constructor directive	4	2
Cangled_string op directive	5	2
Cangled_string user-declaration directive	6	2
Cangled_string user-implementation directive	7	2
copy_kstr_into_buffer	8	3
Cangled_string user-prefix-declaration directive	9	3
Rangled_string rule	10	3
Ropen_angle rule	11	4
First Set Language for O_2^{linker}	12	6
Lr1 State Network	13	7
Index	14	8