

**1. Copyright.**

Copyright © Dave Bone 1998 - 2015

**2. Terminal vocabulary.**

There are some terminals that spoof the grammar’s keywords like “#fsm”. The keyword recognizer recognizes the character string of “fsm” but how do u define it and reference it within the grammars defining  $O_2$ ’s very own language: the conundrum of the grammar defining itself? Ahh to recurse or not to curse, that is the? Well this is my take. The logical keys are just proxy material.

**3. Terminals-refs directive.**

Makesure c++ forward references compile.

```

<terminals-refs directive 3> ≡
  using namespace std;
  using namespace NS_yacco2_err_symbols;
  using namespace yacco2;
  struct T_called_thread_eosubrule;
  struct T_c_literal;
  struct T_identifier;
  struct T_syntax_code;
  struct T_fsm_class_phrase;
  struct T_fsm_class;
  struct T_fsm_phrase;
  struct T_parallel_parser_phrase;
  struct T_enum_phrase;
  struct T_error_symbols_phrase;
  struct T_rc_phrase;
  struct T_lr1_k_phrase;
  struct T_terminals_phrase;
  struct T_rules_phrase;
  struct T_subrules_phrase;
  struct T_rhs_bnd;
  struct refered_rule;
  struct rule_def;
  struct refered_T;
  struct T_in_stbl;
  struct rule_in_stbl;
  extern yacco2::CAbs_lr1_sym*PTR_lint_;
  extern yacco2::CAbs_lr1_sym*PTR_ws_;
  extern yacco2::CAbs_lr1_sym*PTR_eol_;

```

**4. # \*\*\*.**

Enum: T\_T\_eocode\_

Class: T\_eocode

AB: N

AD: N

$O_2$ ’s keyword ending syntax directed code block.

---

**5. # AB.**

Enum: T\_T\_AB\_

Class: T\_AB

AB: N

AD: N

“AB” means auto-abort. This attribute when present with any grammar’s vocabulary definition — rules or terminals, allows proper cleanup when a grammar aborts parsing. Rules always have this attribute turned on to house clean their parse tracings in any situation. Terminal definitions left on the aborted parse stack are also deleted if their “AB” attribute is on. This forced cleanup of the parse stack brings the grammar back to normalcy.

**6. # AD.**

Enum: T\_T\_AD\_

Class: T\_AD

AB: N

AD: N

“AD” means auto-delete. This attribute when present with any grammar’s vocabulary definition — rules or terminals, indicates that the symbol is deleted when popped from the parse stack. Rules always have this attribute turned on.

**7. # NULL.**

Enum: T\_T\_NULL\_

Class: T\_NULL

AB: N

AD: N

**8. # T-enumeration.**

Enum: T\_T\_enumeration\_

Class: T\_enumeration

AB: Y

AD: N

Introduces the enumeration construct of the grammar. The 1 + 2 + 3 scheme counting of the symbols.

**9. # T-enumeration destructor directive.**

```
< # T-enumeration destructor directive 9 > ≡
  if (R-enum_phrase_ ≠ 0) delete R-enum_phrase_;
```

**10. # T-enumeration user-declaration directive.**

```
< # T-enumeration user-declaration directive 10 > ≡
public: T_enumeration();
  void enum_phrase(T_enum_phrase *Phrase);
  T_enum_phrase *enum_phrase();
private: T_enum_phrase *enum_phrase_;
```

**11. # T-enumeration user-implementation directive.**

```

< # T-enumeration user-implementation directive 11 > ≡
  T_enumeration :: T_enumeration() T_CTOR("#T-enumeration", T_Enum :: T_T_enumeration_,
    &dtor_T_enumeration, false, true)
  {
    enum_phrase_ = 0;
  }
  T_enum_phrase *T_enumeration :: enum_phrase()
  {
    return enum_phrase_;
  }
  void T_enumeration :: enum_phrase(T_enum_phrase *Phrase)
  {
    enum_phrase_ = Phrase;
  }

```

**12. # arbitrator-code.**

Enum: T\_T\_arbitrator\_code\_

Class: T\_arbitrator\_code

AB: N

AD: N

$O_2$ 's keyword introducing c++ arbitration code within the grammar's rule.

**13. # arbitrator-code user-declaration directive.**

```

< # arbitrator-code user-declaration directive 13 > ≡
public: T_arbitrator_code();
  void syntax_code(T_syntax_code *Stc);
  T_syntax_code *syntax_code();
  void add_cweb_marker(AST *Cweb);
  AST *cweb_marker();
private: T_syntax_code *syntax_code_;
  AST *cweb_marker_;

```

**14. # arbitrator-code user-implementation directive.**

```

< # arbitrator-code user-implementation directive 14 > ≡
  T_arbitrator_code :: T_arbitrator_code() T_CTOR("#arbitrator-code", T_Enum :: T_T_arbitrator_code_, 0,
    false, false)
  {
    syntax_code_ = 0;
    cweb_marker_ = 0;
  }
  T_syntax_code * T_arbitrator_code :: syntax_code()
  {
    return syntax_code_;
  }
  void T_arbitrator_code :: syntax_code(T_syntax_code * Stc)
  {
    syntax_code_ = Stc;
  }
  void T_arbitrator_code :: add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_arbitrator_code :: cweb_marker()
  {
    return cweb_marker_;
  }

```

**15. # constant-defs.**

Enum: T\_T\_constant\_defs\_

Class: T\_constant\_defs

AB: N

AD: N

**16. # constructor.**

Enum: T\_T\_constructor\_

Class: T\_constructor

AB: N

AD: N

**17. # constructor user-declaration directive.**

```

< # constructor user-declaration directive 17 > ≡
public: T_constructor();
  void syntax_code(T_syntax_code * Stc);
  T_syntax_code * syntax_code();
private: T_syntax_code * syntax_code_;

```

**18. # constructor user-implementation directive.**

```

< # constructor user-implementation directive 18 > ≡
  T_constructor :: T_constructor()T_CTOR("#constructor", T_Enum :: T_T_constructor_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_constructor :: syntax_code()
  {
    return syntax_code_;
  }
  void T_constructor :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**19. # destructor.**

Enum: T\_T\_destructor\_

Class: T\_destructor

AB: N

AD: N

**20. # destructor user-declaration directive.**

```

< # destructor user-declaration directive 20 > ≡
public: T_destructor();
  void syntax_code(T_syntax_code *Stc);
  T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;

```

**21. # destructor user-implementation directive.**

```

< # destructor user-implementation directive 21 > ≡
  T_destructor :: T_destructor()T_CTOR("#destructor", T_Enum :: T_T_destructor_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_destructor :: syntax_code()
  {
    return syntax_code_;
  }
  void T_destructor :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**22. # error-symbols.**

Enum: T\_T\_error\_symbols\_

Class: T\_error\_symbols

AB: N

AD: N

$O_2$ 's keyword introducing the Errors vocabulary.

**23. # error-symbols destructor directive.**

```
< # error-symbols destructor directive 23 > ≡
  if (R-error-symbols_phrase_ ≠ 0) delete R-error-symbols_phrase_;
```

**24. # error-symbols user-declaration directive.**

```
< # error-symbols user-declaration directive 24 > ≡
public: T_error_symbols();
  void error_symbols_phrase(T_error_symbols_phrase *Phrase);
  T_error_symbols_phrase *error_symbols_phrase();
private: T_error_symbols_phrase *error_symbols_phrase_;
```

**25. # error-symbols user-implementation directive.**

```
< # error-symbols user-implementation directive 25 > ≡
  T_error_symbols::T_error_symbols()T_CTOR("#error-symbols", T_Enum::T_T_error_symbols_,
    &dtor_T_error_symbols, false, false)
  {
    error_symbols_phrase_ = 0;
  }
  T_error_symbols_phrase *T_error_symbols::error_symbols_phrase()
  {
    return error_symbols_phrase_;
  }
  void T_error_symbols::error_symbols_phrase(T_error_symbols_phrase *Phrase)
  {
    error_symbols_phrase_ = Phrase;
  }
```

**26. # failed.**Enum: *T\_T\_failed\_*Class: *T\_failed*

AB: N

AD: N

**27. # failed user-declaration directive.**

```
< # failed user-declaration directive 27 > ≡
public: T_failed();
  void syntax_code(T_syntax_code *Stc);
  T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;
```

**28. # failed user-implementation directive.**

```

< # failed user-implementation directive 28 > ≡
  T_failed :: T_failed() T_CTOR("#failed", T_Enum :: T_T_failed_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_failed :: syntax_code()
  {
    return syntax_code_;
  }
  void T_failed :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**29. # file-name.**

Enum: T\_T\_file\_name\_

Class: T\_file\_name

AB: N

AD: N

**30. # fsm.**

Enum: T\_T\_fsm\_

Class: T\_fsm

AB: N

AD: N

**31. # fsm destructor directive.**

```

< # fsm destructor directive 31 > ≡
  if (R-fsm_phrase_ ≠ 0) delete R-fsm_phrase_;

```

**32. # fsm user-declaration directive.**

```

< # fsm user-declaration directive 32 > ≡
public: T_fsm();
  void fsm_phrase(T_fsm_phrase *Phrase);
  T_fsm_phrase *fsm_phrase();
private: T_fsm_phrase *fsm_phrase_;

```



**33. # fsm user-implementation directive.**

```

< # fsm user-implementation directive 33 > ≡
  T_fsm :: T_fsm() T_CTOR("#fsm", T_Enum :: T_T_fsm_, &dtor_T_fsm, false, false)
  {
    fsm_phrase_ = 0;
  }
  T_fsm_phrase *T_fsm :: fsm_phrase()
  {
    return fsm_phrase_;
  }
  void T_fsm :: fsm_phrase(T_fsm_phrase *Phrase)
  {
    fsm_phrase_ = Phrase;
  }

```

**34. # fsm-class.**

Enum: T\_T\_fsm\_class\_

Class: T\_fsm\_class

AB: N

AD: N

**35. # fsm-comments.**

Enum: T\_T\_fsm\_comments\_

Class: T\_fsm\_comments

AB: N

AD: N

**36. # fsm-date.**

Enum: T\_T\_fsm\_date\_

Class: T\_fsm\_date

AB: N

AD: N

**37. # fsm-debug.**

Enum: T\_T\_fsm\_debug\_

Class: T\_fsm\_debug

AB: N

AD: N

**38. # fsm-filename.**

Enum: T\_T\_fsm\_filename\_

Class: T\_fsm\_filename

AB: N

AD: N

**39. # fsm-id.**

Enum: T\_T\_fsm\_id\_

Class: T\_fsm\_id

AB: N

AD: N

**40. # fsm-namespace.**

Enum: T\_T\_fsm\_namespace\_

Class: T\_fsm\_namespace

AB: N

AD: N

**41. # fsm-version.**

Enum: T\_T\_fsm\_version\_

Class: T\_fsm\_version

AB: N

AD: N

**42. # lhs.**

Enum: T\_T\_lhs\_

Class: T\_lhs

AB: N

AD: N

**43. # lr1-constant-symbols.**

Enum: T\_T\_lr1\_constant\_symbols\_

Class: T\_lr1\_constant\_symbols

AB: N

AD: N

 $O_2$ 's keyword introducing the lr constants vocabulary.**44. # lr1-constant-symbols destructor directive.**

```
< # lr1-constant-symbols destructor directive 44 > ≡
  if ( $R \rightarrow lr1\_k\_phrase\_ \neq 0$ ) delete  $R \rightarrow lr1\_k\_phrase\_;$ 
```

**45. # lr1-constant-symbols user-declaration directive.**

```
< # lr1-constant-symbols user-declaration directive 45 > ≡
```

```
public: T_lr1_constant_symbols();
```

```
void lr1_k_phrase(T_lr1_k_phrase *Phrase);
```

```
T_lr1_k_phrase *lr1_k_phrase();
```

```
private: T_lr1_k_phrase *lr1_k_phrase_;
```

**46. # lr1-constant-symbols user-implementation directive.**

```
< # lr1-constant-symbols user-implementation directive 46 > ≡
```

```
T_lr1_constant_symbols::T_lr1_constant_symbols() T_CTOR("#lr1-constant-symbols",
  T_Enum::T_T_lr1_constant_symbols_, &dtor_T_lr1_constant_symbols, false, false)
```

```
{
  lr1_k_phrase_ = 0;
}
```

```
T_lr1_k_phrase *T_lr1_constant_symbols::lr1_k_phrase()
```

```
{
  return lr1_k_phrase_;
}
```

```
void T_lr1_constant_symbols::lr1_k_phrase(T_lr1_k_phrase *Phrase)
```

```
{
  lr1_k_phrase_ = Phrase;
}
```

**47. # lrk-suffix.**

Enum: T\_T\_lrk\_suffix\_

Class: T\_lrk\_suffix

AB: N

AD: N

**48. # name-space.**

Enum: T\_T\_name\_space\_

Class: T\_name\_space

AB: N

AD: N

**49. # op.**

Enum: T\_T\_op\_

Class: T\_op

AB: N

AD: N

**50. # op user-declaration directive.**`< # op user-declaration directive 50 > ≡``public: T_op();``void syntax_code(T_syntax_code *Stc);``T_syntax_code *syntax_code();``private: T_syntax_code *syntax_code_;`**51. # op user-implementation directive.**`< # op user-implementation directive 51 > ≡``T_op::T_op()T_CTOR("#op", T_Enum::T_T_op_, 0, false, false)``{` `syntax_code_ = 0;``}``T_syntax_code *T_op::syntax_code()``{` `return syntax_code_;``}``void T_op::syntax_code(T_syntax_code *Stc)``{` `syntax_code_ = Stc;``}`**52. # parallel-control-monitor.**

Enum: T\_T\_parallel\_control\_monitor\_

Class: T\_parallel\_control\_monitor

AB: N

AD: N

Introduction of the rule's arbitration. For the moment, all rules using a thread call need this grammar construct that introduces the arbitration logic. Most rules do not need to arbitrate. It is determinist in the returned terminal from threads. So why must it be present? Only cuz of my frontend hacking of ideas. It will be corrected to simplify the grammar code.

**53. # parallel-la-boundary.**

Enum: T\_T\_parallel\_la\_boundary\_

Class: T\_parallel\_la\_boundary

AB: N

AD: N

Thread lookahead first set.

**54. # parallel-la-boundary user-declaration directive.**

⟨ # parallel-la-boundary user-declaration directive 54 ⟩ ≡

```

public: T_parallel_la_boundary(); void la_first_set ( std::set < T_in_stbl * > &Supplier ); std::set <
    T_in_stbl * > *la_first_set();
void la_supplier(yacco2::TOKEN_GAGGLE * Supplier);
yacco2::TOKEN_GAGGLE * la_supplier();
void add_cweb_marker(AST * Cweb);
AST * cweb_marker();
void cweb_la_srce_expr(const char *Srce_expr);
std::string * cweb_la_srce_expr(); private: std::set < T_in_stbl * > la_first_set_;
std::string cweb_la_srce_expr_;
AST * cweb_marker_;
yacco2::TOKEN_GAGGLE * la_supplier_;

```

**55. # parallel-la-boundary user-implementation directive.**

```

< # parallel-la-boundary user-implementation directive 55 > ≡
  T_parallel_la_boundary :: T_parallel_la_boundary() T_CTOR("#parallel-la-boundary",
    T_Enum :: T_T_parallel_la_boundary_, 0, false, false)
  {
    la_supplier_ = 0;
    cweb_marker_ = 0;
  }
  void T_parallel_la_boundary :: cweb_la_srce_expr(const char *Srce_expr)
  {
    cweb_la_srce_expr_.append(Srce_expr);
  }
  std::string * T_parallel_la_boundary :: cweb_la_srce_expr()
  {
    return &cweb_la_srce_expr_;
  }
  yacco2::TOKEN_GAGGLE * T_parallel_la_boundary :: la_supplier()
  {
    return la_supplier_;
  }
  void T_parallel_la_boundary :: la_supplier(yacco2::TOKEN_GAGGLE * Supplier)
  {
    la_supplier_ = Supplier;
  }
  void T_parallel_la_boundary :: add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_parallel_la_boundary :: cweb_marker()
  {
    return cweb_marker_;
  }
  std::set < T_in_stbl * > * T_parallel_la_boundary :: la_first_set()
  {
    return &la_first_set_;
  }
  void T_parallel_la_boundary :: la_first_set ( std::set < T_in_stbl * > &Supplier )
  {
    la_first_set_.insert(Supplier.begin(), Supplier.end());
  }

```

**56. # parallel-parser.**

Enum: T\_T\_parallel\_parser\_

Class: T\_parallel\_parser

AB: N

AD: N

$O_2$ 's keyword introducing the grammar as a thread: all the debutant's coming out like its thread name and look ahead expression.

---

**57. # parallel-parser destructor directive.**

```
< # parallel-parser destructor directive 57 > ≡
  if (R-parallel-parser-phrase_ ≠ 0) delete R-parallel-parser-phrase_;
```

**58. # parallel-parser user-declaration directive.**

```
< # parallel-parser user-declaration directive 58 > ≡
public: T_parallel_parser();
  void parallel_parser_phrase(T_parallel_parser_phrase *Phrase);
  T_parallel_parser_phrase *parallel_parser_phrase();
private: T_parallel_parser_phrase *parallel_parser_phrase_;
```

**59. # parallel-parser user-implementation directive.**

```
< # parallel-parser user-implementation directive 59 > ≡
  T_parallel_parser::T_parallel_parser(T_CTOR("#parallel-parser", T_Enum::T_T_parallel_parser_,
    &dtor_T_parallel_parser, false, false))
  {
    parallel_parser_phrase_ = 0;
  }
  T_parallel_parser_phrase *T_parallel_parser::parallel_parser_phrase()
  {
    return parallel_parser_phrase_;
  }
  void T_parallel_parser::parallel_parser_phrase(T_parallel_parser_phrase *Phrase)
  {
    parallel_parser_phrase_ = Phrase;
  }
```

**60. # parallel-thread-function.**Enum: **T\_T\_parallel\_thread\_function\_**Class: **T\_parallel\_thread\_function**

AB: N

AD: N

The grammar thread's name construct.

**61. # parallel-thread-function user-declaration directive.**

```
< # parallel-thread-function user-declaration directive 61 > ≡
public: T_parallel_thread_function();
  void identifier(T_identifier *Id);
  T_identifier *identifier();
  void add_cweb_marker(AST *Cweb);
  AST *cweb_marker();
private: T_identifier *id_;
  AST *cweb_marker_;
```

**62. # parallel-thread-function user-implementation directive.**

```

< # parallel-thread-function user-implementation directive 62 > ≡
  T_parallel_thread_function :: T_parallel_thread_function()T_CTOR("#parallel-thread-function",
    T_Enum :: T_T_parallel_thread_function_, 0, false, false)
  {
    id_ = 0;
    cweb_marker_ = 0;
  }
  T_identifier *T_parallel_thread_function :: identifier()
  {
    return id_;
  }
  void T_parallel_thread_function :: identifier(T_identifier *Id)
  {
    id_ = Id;
  }
  void T_parallel_thread_function :: add_cweb_marker(AST *Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST *T_parallel_thread_function :: cweb_marker()
  {
    return cweb_marker_;
  }

```

**63. # raw-characters.**

Enum: T\_T\_raw\_characters\_

Class: T\_raw\_characters

AB: N

AD: N

$O_2$ 's keyword introducing the raw characters vocabulary.

**64. # raw-characters destructor directive.**

```

< # raw-characters destructor directive 64 > ≡
  if (R_rc_phrase_ ≠ 0) delete R_rc_phrase_;

```

**65. # raw-characters user-declaration directive.**

```

< # raw-characters user-declaration directive 65 > ≡
public: T_raw_characters();
  void rc_phrase(T_rc_phrase *Phrase);
  T_rc_phrase *rc_phrase();
private: T_rc_phrase *rc_phrase_;

```

**66. # raw-characters user-implementation directive.**

```

< # raw-characters user-implementation directive 66 > ≡
  T_raw_characters :: T_raw_characters() T_CTOR("#raw-characters", T_Enum :: T_T_raw_characters_,
      &dtor_T_raw_characters, false, false)
  {
    rc_phrase_ = 0;
  }
  T_rc_phrase *T_raw_characters :: rc_phrase()
  {
    return rc_phrase_;
  }
  void T_raw_characters :: rc_phrase(T_rc_phrase *Phrase)
  {
    rc_phrase_ = Phrase;
  }

```

**67. # rules.**

Enum: T\_T\_rules\_

Class: T\_rules

AB: Y

AD: N

Introduces the rules construct of the grammar.

**68. # rules destructor directive.**

```

< # rules destructor directive 68 > ≡
  if (R→rules_phrase_ ≠ 0) delete R→rules_phrase_;

```

**69. # rules user-declaration directive.**

```

< # rules user-declaration directive 69 > ≡
public: T_rules();
  void rules_phrase(T_rules_phrase *Phrase);
  T_rules_phrase *rules_phrase();
private: T_rules_phrase *rules_phrase_;

```

**70. # rules user-implementation directive.**

```

< # rules user-implementation directive 70 > ≡
  T_rules :: T_rules() T_CTOR("#rules", T_Enum :: T_T_rules_, &dtor_T_rules, false, true)
  {
    rules_phrase_ = 0;
  }
  T_rules_phrase *T_rules :: rules_phrase()
  {
    return rules_phrase_;
  }
  void T_rules :: rules_phrase(T_rules_phrase *Phrase)
  {
    rules_phrase_ = Phrase;
  }

```



**71. # sym-class.**

Enum: T\_T\_sym\_class\_

Class: T\_sym\_class

AB: N

AD: N

**72. # terminals.**

Enum: T\_T\_terminals\_

Class: T\_terminals

AB: Y

AD: N

Introduces the Terminal vocabulary phrase of the grammar. Note the “auto abort” indicator that should match with the CTOR macro. Which vice is it?

**73. # terminals destructor directive.**

```
< # terminals destructor directive 73 > ≡
  if (R-terminals_phrase_ ≠ 0) delete R-terminals_phrase_;
```

**74. # terminals user-declaration directive.**

```
< # terminals user-declaration directive 74 > ≡
public: T_terminals();
  void terminals_phrase(T_terminals_phrase *Phrase);
  T_terminals_phrase *terminals_phrase();
private: T_terminals_phrase *terminals_phrase_;
```

**75. # terminals user-implementation directive.**

```
< # terminals user-implementation directive 75 > ≡
  T_terminals :: T_terminals() T_CTOR("#terminals", T_Enum :: T_T_terminals_, &dtor_T_terminals, false,
    true)
  {
    terminals_phrase_ = 0;
  }
  T_terminals_phrase *T_terminals :: terminals_phrase()
  {
    return terminals_phrase_;
  }
  void T_terminals :: terminals_phrase(T_terminals_phrase *Phrase)
  {
    terminals_phrase_ = Phrase;
  }
```

**76. # terminals-refs.**

Enum: T\_T\_terminals\_refs\_

Class: T\_terminals\_refs

AB: N

AD: N

**77. # terminals-suffix.**

Enum: T\_T\_terminals\_suffix\_

Class: T\_terminals\_suffix

AB: N

AD: N

**78. # user-declaration.**

Enum: T\_T\_user\_declaration\_

Class: T\_user\_declaration

AB: N

AD: N

**79. # user-declaration user-declaration directive.**

⟨ # user-declaration user-declaration directive 79 ⟩ ≡

```
public: T_user_declaration();
       void syntax_code(T_syntax_code *Stc);
       T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;
```

**80. # user-declaration user-implementation directive.**

⟨ # user-declaration user-implementation directive 80 ⟩ ≡

```
T_user_declaration::T_user_declaration()T_CTOR("#user-declaration",
        T_Enum::T_T_user_declaration_,0,false,false)
{
    syntax_code_ = 0;
}
T_syntax_code *T_user_declaration::syntax_code()
{
    return syntax_code_;
}
void T_user_declaration::syntax_code(T_syntax_code *Stc)
{
    syntax_code_ = Stc;
}
```

**81. # user-imp-sym.**

Enum: T\_T\_user\_imp\_sym\_

Class: T\_user\_imp\_sym

AB: N

AD: N

**82. # user-imp-sym user-declaration directive.**

⟨ # user-imp-sym user-declaration directive 82 ⟩ ≡

```
public: T_user_imp_sym();
       void syntax_code(T_syntax_code *Stc);
       T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;
```

**83. # user-imp-sym user-implementation directive.**

```

< # user-imp-sym user-implementation directive 83 > ≡
  T_user_imp_sym :: T_user_imp_sym()T_CTOR("#user-imp-sym", T_Enum :: T_T_user_imp_sym_, 0, false,
    false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_user_imp_sym :: syntax_code()
  {
    return syntax_code_;
  }
  void T_user_imp_sym :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**84. # user-imp-tbl.**

Enum: T\_T\_user\_imp\_tbl\_

Class: T\_user\_imp\_tbl

AB: N

AD: N

**85. # user-imp-tbl user-declaration directive.**

```

< # user-imp-tbl user-declaration directive 85 > ≡
public: T_user_imp_tbl();
  void syntax_code(T_syntax_code *Stc);
  T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;

```

**86. # user-imp-tbl user-implementation directive.**

```

< # user-imp-tbl user-implementation directive 86 > ≡
  T_user_imp_tbl :: T_user_imp_tbl()T_CTOR("#user-imp-tbl", T_Enum :: T_T_user_imp_tbl_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_user_imp_tbl :: syntax_code()
  {
    return syntax_code_;
  }
  void T_user_imp_tbl :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**87. # user-implementation.**

Enum: T\_T\_user\_implementation\_

Class: T\_user\_implementation

AB: N

AD: N

**88. # user-implementation user-declaration directive.**

```

< # user-implementation user-declaration directive 88 > ≡
public: T_user_implementation();
    void syntax_code(T_syntax_code *Stc);
    T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;

```

**89. # user-implementation user-implementation directive.**

```

< # user-implementation user-implementation directive 89 > ≡
T_user_implementation::T_user_implementation()T_CTOR("#user-implementation",
    T_Enum::T_T_user_implementation_, 0, false, false)
{
    syntax_code_ = 0;
}
T_syntax_code *T_user_implementation::syntax_code()
{
    return syntax_code_;
}
void T_user_implementation::syntax_code(T_syntax_code *Stc)
{
    syntax_code_ = Stc;
}

```

**90. # user-prefix-declaration.**

Enum: T\_T\_user\_prefix\_declaration\_

Class: T\_user\_prefix\_declaration

AB: N

AD: N

**91. # user-prefix-declaration user-declaration directive.**

```

< # user-prefix-declaration user-declaration directive 91 > ≡
public: T_user_prefix_declaration();
    void syntax_code(T_syntax_code *Stc);
    T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;

```

**92. # user-prefix-declaration user-implementation directive.**

```

< # user-prefix-declaration user-implementation directive 92 > ≡
  T_user_prefix_declaration :: T_user_prefix_declaration() T_CTOR("#user-prefix-declaration",
    T_Enum :: T_T_user_prefix_declaration_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_user_prefix_declaration :: syntax_code()
  {
    return syntax_code_;
  }
  void T_user_prefix_declaration :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**93. # user-suffix-declaration.**

Enum: T\_T\_user\_suffix\_declaration\_

Class: T\_user\_suffix\_declaration

AB: N

AD: N

**94. # user-suffix-declaration user-declaration directive.**

```

< # user-suffix-declaration user-declaration directive 94 > ≡
public: T_user_suffix_declaration();
  void syntax_code(T_syntax_code *Stc);
  T_syntax_code *syntax_code();
private: T_syntax_code *syntax_code_;

```

**95. # user-suffix-declaration user-implementation directive.**

```

< # user-suffix-declaration user-implementation directive 95 > ≡
  T_user_suffix_declaration :: T_user_suffix_declaration() T_CTOR("#user-suffix-declaration",
    T_Enum :: T_T_user_suffix_declaration_, 0, false, false)
  {
    syntax_code_ = 0;
  }
  T_syntax_code *T_user_suffix_declaration :: syntax_code()
  {
    return syntax_code_;
  }
  void T_user_suffix_declaration :: syntax_code(T_syntax_code *Stc)
  {
    syntax_code_ = Stc;
  }

```

**96. - > .**

Enum: T\_T\_selector\_

Class: T\_selector

AB: N

AD: N

**97. ::.**

Enum: T\_T\_2colon\_

Class: T\_2colon

AB: N

AD: N

**98. T-alphabet.**

Enum: T\_T\_T\_alphabet\_

Class: T\_T\_alphabet

AB: N

AD: N

**99. T-attributes.**

Enum: T\_T\_attributes\_

Class: T\_attributes

AB: N

AD: N

**100. T-attributes user-declaration directive.**

⟨T-attributes user-declaration directive 100⟩ ≡

**public:** *T\_attributes*(**const char** \**Fully\_qualified\_T\_name*, **int** *Enum*);    **std** :: *string\_qualified\_T\_name\_;*    **int** *T\_enum\_;***101. T-attributes user-implementation directive.**

⟨T-attributes user-implementation directive 101⟩ ≡

*T\_attributes* :: *T\_attributes*(**const char** \**Fully\_qualified\_T\_name*, **int** *Enum*)T\_CTOR("T-attributes",  
    *T\_Enum* :: *T\_T\_attributes\_*, 0, false, false)    {  
        *fully\_qualified\_T\_name\_* += *Fully\_qualified\_T\_name*;  
        *T\_enum\_* = *Enum*;  
    }**102. T-enum-phrase.**

Enum: T\_T\_enum\_phrase\_

Class: T\_enum\_phrase

AB: N

AD: N

**103. T-enum-phrase destructor directive.**

⟨T-enum-phrase destructor directive 103⟩ ≡

**delete** *R\_filename\_id\_;*    **delete** *R\_namespace\_id\_;*    **delete** *R\_kdefs\_;*

**104. T-enum-phrase user-declaration directive.**

⟨T-enum-phrase user-declaration directive 104⟩ ≡

```

public: T_enum_phrase();
    T_identifier *filename_id();
    void filename_id(T_identifier *Id);
    T_identifier *namespace_id();
    void namespace_id(T_identifier *Id);
    T_syntax_code *kdefs();
    void kdefs(T_syntax_code *Kdefs);
    void start_lrk_enumerate(int Enum);
    void stop_lrk_enumerate(int Enum);
    int start_lrk_enumerate();
    int stop_lrk_enumerate();
    void start_err_enumerate(int Enum);
    void stop_err_enumerate(int Enum);
    int start_err_enumerate();
    int stop_err_enumerate();
    void start_rc_enumerate(int Enum);
    void stop_rc_enumerate(int Enum);
    int start_rc_enumerate();
    int stop_rc_enumerate();
    void start_T_enumerate(int Enum);
    void stop_T_enumerate(int Enum);
    int start_T_enumerate();
    int stop_T_enumerate();
    int total_T_enumerate();
    void total_T_enumerate(int Enum);
    int total_rc_enumerate();
    void total_rc_enumerate(int Enum);
    int total_err_enumerate();
    void total_err_enumerate(int Enum);
    int total_lrk_enumerate();
    void total_lrk_enumerate(int Enum);
    void total_enumerate(int Enum);
    int total_enumerate();
    void phrase_tree(AST * Tree);
    AST * phrase_tree();
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
    int total_no_subrules();
    void total_no_subrules(int Total_subrules);
private: T_syntax_code *kdefs_;
    T_identifier *filename_id_;
    T_identifier *namespace_id_;
    int start_lrk_enumerate_;
    int stop_lrk_enumerate_;
    int start_err_enumerate_;
    int stop_err_enumerate_;
    int start_rc_enumerate_;
    int stop_rc_enumerate_;
    int start_T_enumerate_;

```

```
int stop_T_enumerate_;  
int total_enumerate_;  
int total_T_enumerate_;  
int total_rc_enumerate_;  
int total_err_enumerate_;  
int total_lrk_enumerate_;  
int total_no_subrules_;  
AST * phrase_tree_;  
AST * cweb_marker_;
```



**105. T-enum-phrase user-implementation directive.**

```

⟨T-enum-phrase user-implementation directive 105⟩ ≡
  T_enum_phrase::T_enum_phrase()T_CTOR("T-enum-phrase", T_Enum::T_T_enum_phrase_,
    &dtor_T_enum_phrase, false, false)
  {
    kdefs_ = 0;
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_enum_phrase::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_enum_phrase::cweb_marker()
  {
    return cweb_marker_;
  }
  void T_enum_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_enum_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  T_identifier *T_enum_phrase::filename_id()
  {
    return filename_id_;
  }
  void T_enum_phrase::filename_id(T_identifier *Id)
  {
    filename_id_ = Id;
  }
  T_identifier *T_enum_phrase::namespace_id()
  {
    return namespace_id_;
  }
  void T_enum_phrase::namespace_id(T_identifier *Id)
  {
    namespace_id_ = Id;
  }
  T_syntax_code *T_enum_phrase::kdefs()
  {
    return kdefs_;
  }
  void T_enum_phrase::kdefs(T_syntax_code *Kdefs)
  {
    kdefs_ = Kdefs;
  }

```

```

void T_enum_phrase::start_lrk_enumerate(int Enum)
{
    start_lrk_enumerate_ = Enum;
}
void T_enum_phrase::stop_lrk_enumerate(int Enum)
{
    stop_lrk_enumerate_ = Enum;
}
int T_enum_phrase::start_lrk_enumerate()
{
    return start_lrk_enumerate_;
}
int T_enum_phrase::stop_lrk_enumerate()
{
    return stop_lrk_enumerate_;
}
void T_enum_phrase::start_err_enumerate(int Enum)
{
    start_err_enumerate_ = Enum;
}
void T_enum_phrase::stop_err_enumerate(int Enum)
{
    stop_err_enumerate_ = Enum;
}
int T_enum_phrase::start_err_enumerate()
{
    return start_err_enumerate_;
}
int T_enum_phrase::stop_err_enumerate()
{
    return stop_err_enumerate_;
}
void T_enum_phrase::start_rc_enumerate(int Enum)
{
    start_rc_enumerate_ = Enum;
}
void T_enum_phrase::stop_rc_enumerate(int Enum)
{
    stop_rc_enumerate_ = Enum;
}
int T_enum_phrase::start_rc_enumerate()
{
    return start_rc_enumerate_;
}
int T_enum_phrase::stop_rc_enumerate()
{
    return stop_rc_enumerate_;
}
void T_enum_phrase::start_T_enumerate(int Enum)

```

```
{
    start_T_enumerate_ = Enum;
}
void T_enum_phrase::stop_T_enumerate(int Enum)
{
    stop_T_enumerate_ = Enum;
}
int T_enum_phrase::start_T_enumerate()
{
    return start_T_enumerate_;
}
int T_enum_phrase::stop_T_enumerate()
{
    return stop_T_enumerate_;
}
void T_enum_phrase::total_enumerate(int Enum)
{
    total_enumerate_ = Enum;
}
int T_enum_phrase::total_enumerate()
{
    return total_enumerate_;
}
void T_enum_phrase::total_T_enumerate(int Enum)
{
    total_T_enumerate_ = Enum;
}
int T_enum_phrase::total_T_enumerate()
{
    return total_T_enumerate_;
}
void T_enum_phrase::total_rc_enumerate(int Enum)
{
    total_rc_enumerate_ = Enum;
}
int T_enum_phrase::total_rc_enumerate()
{
    return total_rc_enumerate_;
}
void T_enum_phrase::total_err_enumerate(int Enum)
{
    total_err_enumerate_ = Enum;
}
int T_enum_phrase::total_err_enumerate()
{
    return total_err_enumerate_;
}
void T_enum_phrase::total_lrk_enumerate(int Enum)
{
```

```

    total_lrk_enumerate_ = Enum;
}
int T_enum_phrase::total_lrk_enumerate()
{
    return total_lrk_enumerate_;
}
int T_enum_phrase::total_no_subrules()
{
    return total_no_subrules_;
}
void T_enum_phrase::total_no_subrules(int Total_subrules)
{
    total_no_subrules_ = Total_subrules;
}

```

**106. T-in-stbl.**

Enum: T\_T\_in\_stbl\_

Class: T\_in\_stbl

AB: N

AD: N

**107. T-in-stbl user-declaration directive.**

⟨T-in-stbl user-declaration directive 107⟩ ≡

```

public: T_in_stbl(T_terminal_def * Fnd_T_in_stbl, CAbs_lr1_sym * Rc, yacco2::Parser * P);
    T_terminal_def * t_def(); std::list < referred_T * > *xref_t();
    void add_T_into_xref(referred_T &T);
    void stbl_idx(yacco2::UINT Idx);
    int stbl_idx();
private: T_terminal_def * t_def_; std::list < referred_T * > xref_t_;
    int stbl_idx_;

```

**108. T-in-stbl user-implementation directive.**

```

⟨T-in-stbl user-implementation directive 108⟩ ≡
  T_in_stbl::T_in_stbl(T_terminal_def * Fnd_T_in_stbl, CAbs_lr1_sym * Rc,
    yacco2::Parser * P)T_CTOR("T-in-stbl", T_Enum::T_T_in_stbl, 0, false, false)
  {
    t_def_ = Fnd_T_in_stbl;
    set_rc(*Rc, __FILE__, __LINE__); /* set its source co-ordinates */
    stbl_idx_ = -1;
  }
  T_terminal_def * T_in_stbl::t_def()
  {
    return t_def_;
  }
  std::list < referred_T * > *T_in_stbl::xref_t()
  {
    return &xref_t_;
  };
  void T_in_stbl::add_T_into_xref(referred_T &T)
  {
    xref_t_.push_back(&T);
  }
  void T_in_stbl::stbl_idx(yacco2::UINTIdx)
  {
    stbl_idx_ = Idx;
  }
  int T_in_stbl::stbl_idx()
  {
    return stbl_idx_;
  }

```

**109. angled-string.**

Enum: T\_T\_angled\_string\_

Class: T\_angled\_string

AB: N

AD: N

Returned from *angled\_string* grammar whereby the *angled\_string\_* contains its content without the angle bars. Escape sequences are verified and not converted but just concatenated to its content.

**110. angled-string user-declaration directive.**

```

⟨angled-string user-declaration directive 110⟩ ≡
public: T_angled_string(const std::string& Angled_string);
  std::string * angled_string();
private: std::string angled_string_;

```

**111. angled-string user-implementation directive.**

```

⟨angled-string user-implementation directive 111⟩ ≡
  T_angled_string :: T_angled_string(const string & Angled_string) T_CTOR("angled-string",
    T_Enum :: T_T_angled_string_, 0, false, false)
  {
    angled_string_ = Angled_string;
  }
  std :: string * T_angled_string :: angled_string()
  {
    return &angled_string_;
  }

```

**112. basic-char.**

Enum: T\_T\_basic\_char\_

Class: T\_basic\_char

AB: N

AD: N

Basic source character set indicator. It can be used to assess the good, the bad, and the ugly character traits. For example, prescan the characters against the *bad\_char\_set* thread and return one of the 2 terminals. It is left here but not used.

**113. basic-char destructor directive.**

```

⟨basic-char destructor directive 113⟩ ≡
  if (R-basic_char_ ≠ 0) delete R-basic_char();

```

**114. basic-char user-declaration directive.**

```

⟨basic-char user-declaration directive 114⟩ ≡
public: T_basic_char(CAbs_lr1_sym * Basic_char);
  CAbs_lr1_sym * basic_char() const;
  void zero_out_basic_char();
private: CAbs_lr1_sym * basic_char_;

```

**115. basic-char user-implementation directive.**

```

⟨basic-char user-implementation directive 115⟩ ≡
  T_basic_char :: T_basic_char(CAbs_lr1_sym * Basic_char) T_CTOR("basic-char",
    T_Enum :: T_T_basic_char_, &dtor_T_basic_char, false, false)
  {
    basic_char_ = Basic_char;
  }
  void T_basic_char :: zero_out_basic_char()
  {
    basic_char_ = 0;
  }
  CAbs_lr1_sym * T_basic_char :: basic_char() const
  {
    return basic_char_;
  }

```

**116. block.**

Enum: T\_T\_block\_

Class: T\_block

AB: Y

AD: N

**117. block destructor directive.**

```

⟨block destructor directive 117⟩ ≡
  if (ABORT_STATUS ≡ true) {
    yacco2::AST::zero_content(*R→ast()); /* break dbl delete of self */
    yacco2::AST::ast_delete(*R→ast(), ABORT_STATUS);
  }

```

**118. block user-declaration directive.**

```

⟨block user-declaration directive 118⟩ ≡
public: T_block(yacco2::AST * Ast);
  yacco2::AST * ast();
private: yacco2::AST * ast_;

```

**119. block user-implementation directive.**

```

⟨block user-implementation directive 119⟩ ≡
  T_block::T_block(yacco2::AST * Ast)T_CTOR("block", T_Enum::T_T_block_, &dtor_T_block, false, true)
  {
    ast_ = Ast;
  }
  yacco2::AST * T_block::ast()
  {
    return ast_;
  }

```

**120. c-literal.**

Enum: T\_T\_c\_literal\_

Class: T\_c\_literal

AB: N

AD: N

Returned from *c\_literal* grammar recognizing a c++ literal. *c\_literal\_* contains its content without the single quotes. 'hi there' is an example. Escape sequences are verified and not converted but concatenated to its content.

**121. c-literal user-declaration directive.**

```

⟨c-literal user-declaration directive 121⟩ ≡
public: T_c_literal(const std::string& C_literal);
  std::string * c_literal();
private: std::string c_literal_;

```

**122. c-literal user-implementation directive.**

```

<c-literal user-implementation directive 122> ≡
  T_c_literal::T_c_literal(const string&C_literal)T_CTOR("c-literal", T_Enum::T_T_c_literal_, 0,
    false, false)
  {
    c_literal_ = C_literal;
  }
  std::string * T_c_literal::c_literal()
  {
    return &c_literal_;
  }

```

**123. c-string.**

Enum: T\_T\_c\_string\_

Class: T\_c\_string

AB: N

AD: N

Returned from *c\_string* grammar recognizing a c++ string. *c\_string\_* contains its content without the double quotes.

**124. c-string user-declaration directive.**

```

<c-string user-declaration directive 124> ≡
public: T_c_string(const std::string&C_string);
  std::string * c_string();
private: std::string c_string_;

```

**125. c-string user-implementation directive.**

```

<c-string user-implementation directive 125> ≡
  T_c_string::T_c_string(const std::string&C_string)T_CTOR("c-string", T_Enum::T_T_c_string_, 0,
    false, false)
  {
    c_string_ = C_string;
  }
  std::string * T_c_string::c_string()
  {
    return &c_string_;
  }

```

**126. called thread eosubrle.**

Enum: T\_T\_called\_thread\_eosubrle\_

Class: T\_called\_thread\_eosubrle

AB: N

AD: N



**127. called thread eosubrule user-declaration directive.**

⟨called thread eosubrule user-declaration directive 127⟩ ≡

```
public: T_identifier *Ns, T_identifier *Thread_name);
    T_subrule_def *its_subrule_def();
void its_subrule_def(T_subrule_def *Its_subrule);
std::string *grammar_s_enumerate();
void grammar_s_enumerate(const char *Enumerate);
AST *tree_node();
void tree_node(AST *Tree_node);
rule_def *its_rule_def();
T_identifier *ns();
T_identifier *called_thread_name();
int element_pos();
void element_pos(int Pos);
private: T_subrule_def *its_subrule_def_;
    std::string grammar_s_enumerate_;
    AST *tree_node_;
    T_identifier *ns_;
    T_identifier *called_thread_name_;
    int element_pos_;
```

**128. called thread eosubrule user-implementation directive.**

⟨called thread eosubrule user-implementation directive 128⟩ ≡

```

T_called_thread_eosubrule :: T_called_thread_eosubrule(T_identifier *Ns, T_identifier
    *Thd_name)T_CTOR("called_thread_eosubrule", T_Enum :: T_T_called_thread_eosubrule_, 0,
    false, false)
{
    its_subrule_def_ = 0;
    tree_node_ = 0;
    ns_ = Ns;
    called_thread_name_ = Thd_name;
    element_pos_ = 0;
}
int T_called_thread_eosubrule :: element_pos()
{
    return element_pos_;
}
void T_called_thread_eosubrule :: element_pos(int Pos)
{
    element_pos_ = Pos;
}
T_identifier *T_called_thread_eosubrule :: ns()
{
    return ns_;
}
T_identifier *T_called_thread_eosubrule :: called_thread_name()
{
    return called_thread_name_;
}
rule_def *T_called_thread_eosubrule :: its_rule_def()
{
    return its_subrule_def_→its_rule_def();
}
AST * T_called_thread_eosubrule :: tree_node()
{
    return tree_node_;
}
void T_called_thread_eosubrule :: tree_node(AST * Tree_node)
{
    tree_node_ = Tree_node;
}
std :: string * T_called_thread_eosubrule :: grammar_s_enumerate()
{
    return &grammar_s_enumerate_;
}
void T_called_thread_eosubrule :: grammar_s_enumerate(const char *Enumerate)
{
    grammar_s_enumerate_ += Enumerate;
}
T_subrule_def * T_called_thread_eosubrule :: its_subrule_def()

```

```

{
    return its_subrule_def_;
}
void T_called_thread_eosubrule::its_subrule_def(T_subrule_def * Its_subrule)
{
    its_subrule_def_ = Its_subrule;
}

```

**129. comment.**

Enum: T\_T\_comment\_

Class: T\_comment

AB: N

AD: N

Basic c++ comment in its 2 forms:  
 /\* ...\*/ and single line //. The *c\_comment* grammar returns it.

**130. comment user-declaration directive.**

⟨comment user-declaration directive 130⟩ ≡

```

public: T_comment(const std::string& Comment_data);
       std::string * comment_data();
private: std::string comment_data_;

```

**131. comment user-implementation directive.**

⟨comment user-implementation directive 131⟩ ≡

```

T_comment::T_comment(const std::string& Comment_data)T_CTOR("comment",
    T_Enum::T_T_comment_, 0, false, false)
{
    comment_data_ = Comment_data;
}
std::string * T_comment::comment_data()
{
    return &comment_data_;
}

```

**132. cweb-comment.**

Enum: T\_T\_cweb\_comment\_

Class: T\_cweb\_comment

AB: N

AD: N

**133. cweb-comment user-declaration directive.**

⟨cweb-comment user-declaration directive 133⟩ ≡

```

public: T_cweb_comment(const std::string& Comment_data);
       std::string * comment_data();
private: std::string comment_data_;

```

**134. cweb-comment user-implementation directive.**

```

<cweb-comment user-implementation directive 134> ≡
  T_cweb_comment :: T_cweb_comment(const std::string& Comment_data)T_CTOR("cweb-comment",
    T_Enum :: T_T_cweb_comment_, 0, false, false)
  {
    comment_data_ = Comment_data;
  }
  std::string * T_cweb_comment :: comment_data()
  {
    return &comment_data_;
  }

```

**135. cweb-marker.**

Enum: T\_T\_cweb\_marker\_

Class: T\_cweb\_marker

AB: N

AD: N

**136. cweb-marker user-declaration directive.**

```

<cweb-marker user-declaration directive 136> ≡

```

**public:** T\_cweb\_marker(AST \* Node);

T\_cweb\_marker();

AST \* ast();

**private:** AST \* ast\_;**137. cweb-marker user-implementation directive.**

```

<cweb-marker user-implementation directive 137> ≡

```

```

  T_cweb_marker :: T_cweb_marker(AST * Node)T_CTOR("cweb-marker", T_Enum :: T_T_cweb_marker_, 0,
    false, false)

```

```

  {
    ast_ = Node;
  }

```

```

  T_cweb_marker :: T_cweb_marker()T_CTOR("cweb-marker", T_Enum :: T_T_cweb_marker_, 0, false, false)

```

```

  {
    ast_ = 0;
  }

```

```

  AST * T_cweb_marker :: ast()

```

```

  {
    return ast_;
  }

```

**138. emitfile.**

Enum: T\_T\_emitfile\_

Class: T\_emitfile

AB: N

AD: N

**139. end-T-alphabet.**

Enum: T\_T\_end\_T\_alphabet\_

Class: T\_end\_T\_alphabet

AB: N

AD: N

**140. end-list-of-native-first-set-terminals.**Enum: `T_T_end_list_of_native_first_set_terminals_`Class: `T_end_list_of_native_first_set_terminals` AB: N AD: N**141. end-list-of-transitive-threads.**Enum: `T_T_end_list_of_transitive_threads_`Class: `T_end_list_of_transitive_threads` AB: N AD: N**142. end-list-of-used-threads.**Enum: `T_T_end_list_of_used_threads_`Class: `T_end_list_of_used_threads` AB: N AD: N**143. end-preamble.**Enum: `T_T_end_preamble_`Class: `T_end_preamble` AB: N AD: N**144. eol.**Enum: `T_T_eol_`Class: `T_eol` AB: N AD: NThis is the end-of-line indicator returned from the *eol* thread.**145. eol user-declaration directive.**`<eol user-declaration directive 145> ≡`**public:** `T_eol()`;**146. eol user-implementation directive.**`<eol user-implementation directive 146> ≡``T_eol :: T_eol()T_CTOR("eol", T_Enum :: T_T_eol_, 0, false, false)``{ }``T_eol_eol_;``yacco2 :: CAbs_lr1_sym * NS_yacco2_terminals :: PTR_eol_ = &eol_;`**147. eosubruler.**Enum: `T_T_eosubruler_`Class: `T_eosubruler` AB: N AD: N

**148. eosubrule user-declaration directive.**

⟨eosubrule user-declaration directive 148⟩ ≡

```
public: T_eosubrule();  
         T_subrule_def * its_subrule_def();  
void its_subrule_def(T_subrule_def * Its_subrule);  
std::string * grammar_s_enumerate();  
void grammar_s_enumerate(const char *Enumerate);  
AST * tree_node();  
void tree_node(AST * Tree_node);  
rule_def *its_rule_def();  
int element_pos();  
void element_pos(int Pos);  
private: T_subrule_def * its_subrule_def_;  
         std::string grammar_s_enumerate_;  
         AST * tree_node_;  
         int element_pos_;
```

**149. eosubrule user-implementation directive.**

```

⟨eosubrule user-implementation directive 149⟩ ≡
  T_eosubrule :: T_eosubrule()T_CTOR("eosubrule", T_Enum :: T_T_eosubrule_, 0, false, false)
  {
    its_subrule_def_ = 0;
    tree_node_ = 0;
    element_pos_ = 0;
  }
  int T_eosubrule :: element_pos()
  {
    return element_pos_;
  }
  void T_eosubrule :: element_pos(int Pos)
  {
    element_pos_ = Pos;
  }
  rule_def *T_eosubrule :: its_rule_def()
  {
    return its_subrule_def_→its_rule_def();
  }
  AST * T_eosubrule :: tree_node()
  {
    return tree_node_;
  }
  void T_eosubrule :: tree_node(AST * Tree_node)
  {
    tree_node_ = Tree_node;
  }
  std::string * T_eosubrule :: grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
  void T_eosubrule :: grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
  T_subrule_def * T_eosubrule :: its_subrule_def()
  {
    return its_subrule_def_;
  }
  void T_eosubrule :: its_subrule_def(T_subrule_def * Its_subrule)
  {
    its_subrule_def_ = Its_subrule;
  }

```

**150. error-symbols-phrase.**

Enum: T\_T\_error\_symbols\_phrase\_

Class: T\_error\_symbols\_phrase

AB: N

AD: N

**151. error-symbols-phrase destructor directive.**

⟨error-symbols-phrase destructor directive 151⟩ ≡

```
R-destroy_alphabet();
delete R-filename_id_;
delete R-namespace_id_;
```

**152. error-symbols-phrase user-declaration directive.**

⟨error-symbols-phrase user-declaration directive 152⟩ ≡

```
public: T_error_symbols_phrase();
    T_identifier *filename_id();
    void filename_id(T_identifier *Id);
    T_identifier *namespace_id();
    void namespace_id(T_identifier *Id);
    void destroy_alphabet();
    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *alphabet();
    CAbs_lr1_sym * add_t_to_alphabet(T_terminal_def *T, yacco2::Parser *P);
    std::vector < T_terminal_def *> *crt_order();
    void phrase_tree(AST * Tree);
    AST * phrase_tree();
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
private: T_identifier *filename_id_;
    T_identifier *namespace_id_;
    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> alphabet_;
    std::vector < T_terminal_def *> crt_order_;
    AST * phrase_tree_;
    AST * cweb_marker_;
```



**153. error-symbols-phrase user-implementation directive.**

```

<error-symbols-phrase user-implementation directive 153> ≡
  T_error_symbols_phrase::T_error_symbols_phrase()T_CTOR("error-symbols-phrase",
    T_Enum::T_T_error_symbols_phrase_, &dtor_T_error_symbols_phrase, false, false)
  {
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_error_symbols_phrase::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_error_symbols_phrase::cweb_marker()
  {
    return cweb_marker_;
  }
  void T_error_symbols_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_error_symbols_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  T_identifier *T_error_symbols_phrase::filename_id()
  {
    return filename_id_;
  }
  void T_error_symbols_phrase::filename_id(T_identifier *Id)
  {
    filename_id_ = Id;
  }
  T_identifier *T_error_symbols_phrase::namespace_id()
  {
    return namespace_id_;
  }
  void T_error_symbols_phrase::namespace_id(T_identifier *Id)
  {
    namespace_id_ = Id;
  }
  std::map < std::string,
    NS_yacco2_terminals::T_terminal_def *> *T_error_symbols_phrase::alphabet()
  {
    return &alphabet_;
  }
  std::vector < T_terminal_def *> *T_error_symbols_phrase::crt_order()
  {
    return &crt_order_;
  }

```

```

void T_error_symbols_phrase::destroy_alphabet()
{
    std::vector < T_terminal_def *> ::iterator i = crt_order_.begin();
    std::vector < T_terminal_def *> ::iterator ie = crt_order_.end();
    for (; i != ie; ++i) {
        CAbs_lr1_sym * sym = *i;
        delete sym;
    }
    alphabet_.clear();
}
CAbs_lr1_sym * T_error_symbols_phrase::add_t_to_alphabet(T_terminal_def *T, yacco2::Parser *P)
{
    std::stringkey(T->t_name()-c_str());
    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> ::iterator i = alphabet_.find(key);
    if (i != alphabet_.end()) {
        CAbs_lr1_sym * sym = new Err_dup_entry_in_alphabet;
        sym->set_rc(*T, __FILE__, __LINE__);
        return sym;
    }
    alphabet_[key] = T;
    crt_order_.push_back(T);
    return 0;
}

```

**154. esc-seq.**

Enum: T\_T\_esc\_seq\_

Class: T\_esc\_seq

AB: N

AD: Y

Returned T from the *esc\_seq* thread. Note within the *ctor* macro that the “auto delete” indicator is explicitly turned on and matches the “AD” switch. As i do not parse the c++ code, **watch your self**. This is a dangerous bend in T<sub>E</sub>Xparlance.

**155. esc-seq user-declaration directive.**

<esc-seq user-declaration directive 155> ≡

**public:** T\_esc\_seq(const char \*Esc\_data);

std::string \* esc\_data();

**private:** std::string esc\_data\_;

**156. esc-seq user-implementation directive.**

<esc-seq user-implementation directive 156> ≡

T\_esc\_seq::T\_esc\_seq(const char \*Esc\_data)T\_CTOR("esc-seq", T\_Enum::T\_T\_esc\_seq\_, 0, true, false)

```
{
    esc_data_ += Esc_data;
}
```

std::string \* T\_esc\_seq::esc\_data()

```
{
    return &esc_data_;
}
```

```
}
```

```
;
```

**157. file-inclusion.**

Enum: T\_T\_file\_inclusion\_

Class: T\_file\_inclusion

AB: N

AD: N

Used in processing file inclusion: hence the name. Why the *error\_sym\_* variable? Normally i use this terminal by calling a file processing procedure. Deposited in this variable are conditions found to not allow the reading of the file. See *pass3* thread of how it applies.

**158. file-inclusion user-declaration directive.**

⟨file-inclusion user-declaration directive 158⟩ ≡

```
public: T_file_inclusion(T_c_string * File_name_, CAbs_lr1_sym * Error);
    ~T_file_inclusion();
    T_c_string * file_name();
    CAbs_lr1_sym * error_sym();
    void error_sym(CAbs_lr1_sym * Error);
private: T_c_string * file_name_;
    CAbs_lr1_sym * error_sym_;
```

**159. file-inclusion user-implementation directive.**

⟨file-inclusion user-implementation directive 159⟩ ≡

```
T_file_inclusion::T_file_inclusion(T_c_string * File_name,
    CAbs_lr1_sym * Error)T_CTOR("file-inclusion", T_Enum::T_T_file_inclusion_, 0, false, false)
{
    file_name_ = File_name;
    error_sym_ = Error;
}
CAbs_lr1_sym * T_file_inclusion::error_sym()
{
    return error_sym_;
}
void T_file_inclusion::error_sym(CAbs_lr1_sym * Error)
{
    error_sym_ = Error;
}
T_c_string * T_file_inclusion::file_name()
{
    return file_name_;
}
T_file_inclusion::~~T_file_inclusion()
{
    delete file_name_;
    delete error_sym_;
}
```

**160. file-of-T-alphabet.**

Enum: T\_T\_file\_of\_T\_alphabet\_

Class: T\_file\_of\_T\_alphabet

AB: N

AD: N

**161. fsm-class-phrase.**

Enum: T\_T\_fsm\_class\_phrase\_

Class: T\_fsm\_class\_phrase

AB: N

AD: N

**162. fsm-class-phrase destructor directive.**

```

⟨ fsm-class-phrase destructor directive 162 ⟩ ≡
  if (R-identifier_ ≠ 0) delete R-identifier_;
  R-remove_directives_from_map();

```

**163. fsm-class-phrase user-declaration directive.**

```

⟨ fsm-class-phrase user-declaration directive 163 ⟩ ≡
public: T_fsm_class_phrase();
  T_identifier *identifier();
  void identifier(T_identifier *Id);
  void remove_directives_from_map();
  std::map < std::string, CAbs_lr1_sym *> *directives_map();
  CAbs_lr1_sym * /* 0 - ok, or error */
  add_directive_to_map(CAbs_lr1_sym * Directive, yacco2::Parser * P);
  void phrase_tree(AST * Tree);
  AST * phrase_tree();
private: T_identifier *identifier_;
  std::map < std::string, CAbs_lr1_sym *> directives_map_;
  AST * phrase_tree_;

```

**164. fsm-class-phrase user-implementation directive.**

```

⟨fsm-class-phrase user-implementation directive 164⟩ ≡
  T_fsm_class_phrase :: T_fsm_class_phrase() T_CTOR("fsm-class-phrase",
    T_Enum :: T_T_fsm_class_phrase_, &dtor_T_fsm_class_phrase, false, false)
  {
    identifier_ = 0;
    phrase_tree_ = 0;
  }
  void T_fsm_class_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_fsm_class_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  T_identifier * T_fsm_class_phrase::identifier()
  {
    return identifier_;
  }
  void T_fsm_class_phrase::identifier(T_identifier * Id)
  {
    identifier_ = Id;
  }
  std::map < string, CAbs_lr1_sym *> * T_fsm_class_phrase::directives_map()
  {
    return &directives_map_;
  }
  void T_fsm_class_phrase::remove_directives_from_map()
  {
    std::map < std::string, CAbs_lr1_sym *> ::iterator i = directives_map_.begin();
    std::map < std::string, CAbs_lr1_sym *> ::iterator ie = directives_map_.end();
    for (; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = i->second;
      delete sym;
    }
  }
  CAbs_lr1_sym * /* 0 - ok, or error */
  T_fsm_class_phrase::add_directive_to_map(CAbs_lr1_sym * Directive, yacco2::Parser * P)
  {
    std::map < std::string, CAbs_lr1_sym *> ::iterator i;
    stringkey(Directive-id_);
    i = directives_map_.find(key);
    if (i ≡ directives_map_.end()) {
      directives_map_[key] = Directive;
      return 0;
    }
    CAbs_lr1_sym * sym = new Err_duplicate_directive;
    sym->set_rc(*Directive, __FILE__, __LINE__);
    remove_directives_from_map();
  }

```

```
    return sym;  
}
```

**165. fsm-phrase.**

Enum: T\_T\_fsm\_phrase\_

Class: T\_fsm\_phrase

AB: N

AD: N

---

**166. fsm-phrase destructor directive.**

⟨ fsm-phrase destructor directive 166 ⟩ ≡

```
delete R-fsm_id_;  
delete R-filename_id_;  
delete R-namespace_id_;  
delete R-fsm_class_phrase_;  
delete R-version_;  
delete R-date_;  
delete R-debug_;  
delete R-comment_;
```

**167. fsm-phrase user-declaration directive.**

⟨ fsm-phrase user-declaration directive 167 ⟩ ≡

```

public: T_fsm_phrase();
    T_c_string * fsm_id();
void fsm_id(T_c_string * Id);
T_identifier * filename_id();
void filename_id(T_identifier * Id);
T_identifier * namespace_id();
void namespace_id(T_identifier * Id);
T_fsm_class_phrase * fsm_class_phrase();
void fsm_class_phrase(T_fsm_class_phrase * Id);
    T_c_string * version();
void version(T_c_string * Id);
    T_c_string * date();
void date(T_c_string * Id);
    T_c_string * debug();
void debug(T_c_string * Id);
    T_c_string * comment();
void comment(T_c_string * Id);
void phrase_tree(AST * Tree);
AST * phrase_tree();
void add_cweb_marker(AST * Cweb);
AST * cweb_marker();
private: T_c_string * fsm_id_;
    T_identifier * filename_id_;
    string filename_no_ext_;
    T_identifier * namespace_id_;
    T_fsm_class_phrase * fsm_class_phrase_;
    T_c_string * version_;
    T_c_string * date_;
    T_c_string * debug_;
    T_c_string * comment_;
AST * phrase_tree_;
AST * cweb_marker_;

```

**168. fsm-phrase user-implementation directive.**

⟨ fsm-phrase user-implementation directive 168 ⟩ ≡

```

T_fsm_phrase::T_fsm_phrase()T_CTOR("fsm-phrase", T_Enum::T_T_fsm_phrase_,
    &dtor_T_fsm_phrase, false, false)
{
    fsm_id_ = 0;
    filename_id_ = 0;
    namespace_id_ = 0;
    fsm_class_phrase_ = 0;
    version_ = 0;
    date_ = 0;
    debug_ = 0;
    comment_ = 0;
    phrase_tree_ = 0;
    cweb_marker_ = 0;
}
void T_fsm_phrase::add_cweb_marker(AST * Cweb)
{
    cweb_marker_ = Cweb;
}
AST * T_fsm_phrase::cweb_marker()
{
    return cweb_marker_;
}
void T_fsm_phrase::phrase_tree(AST * Tree)
{
    phrase_tree_ = Tree;
}
AST * T_fsm_phrase::phrase_tree()
{
    return phrase_tree_;
}
T_c_string * T_fsm_phrase::fsm_id()
{
    return fsm_id_;
}
void T_fsm_phrase::fsm_id(T_c_string * Id)
{
    fsm_id_ = Id;
}
T_identifier * T_fsm_phrase::filename_id()
{
    return filename_id_;
}
void T_fsm_phrase::filename_id(T_identifier * Id)
{
    filename_id_ = Id;
}
T_identifier * T_fsm_phrase::namespace_id()
{

```



```

    return namespace_id_;
}
void T_fsm_phrase::namespace_id(T_identifier *Id)
{
    namespace_id_ = Id;
}
T_fsm_class_phrase *T_fsm_phrase::fsm_class_phrase()
{
    return fsm_class_phrase_;
}
void T_fsm_phrase::fsm_class_phrase(T_fsm_class_phrase *Id)
{
    fsm_class_phrase_ = Id;
}
T_c_string *T_fsm_phrase::version()
{
    return version_;
}
void T_fsm_phrase::version(T_c_string * Id)
{
    version_ = Id;
}
T_c_string *T_fsm_phrase::date()
{
    return date_;
}
void T_fsm_phrase::date(T_c_string * Id)
{
    date_ = Id;
}
T_c_string *T_fsm_phrase::debug()
{
    return debug_;
}
void T_fsm_phrase::debug(T_c_string * Id)
{
    debug_ = Id;
}
T_c_string *T_fsm_phrase::comment()
{
    return comment_;
}
void T_fsm_phrase::comment(T_c_string * Id)
{
    comment_ = Id;
}

```

**169. grammar-name.**

Enum: T\_T\_grammar\_name\_

Class: T\_grammar\_name AB: N AD: N

**170. grammar-phrase.**

Enum: T\_T\_grammar\_phrase\_

Class: T\_grammar\_phrase AB: N AD: N

**171. identifier.**

Enum: T\_T\_identifier\_

Class: T\_identifier AB: N AD: N

Returned terminal from the *identifier* thread. Just your basic word classifier.**172. identifier user-declaration directive.**

```

<identifier user-declaration directive 172> ≡
public: T_identifier(yacco2::KCHARP Identifier);
       std::string * identifier();
private: std::string identifier_;

```

**173. identifier user-implementation directive.**

```

<identifier user-implementation directive 173> ≡
T_identifier::T_identifier(yacco2::KCHARP Identifier)T_CTOR("identifier",
    T_Enum::T_T_identifier_, 0, false, false)
{
    identifier_ = Identifier;
}
std::string * T_identifier::identifier()
{
    return &identifier_;
}

```

**174. int-no.**

Enum: T\_T\_int\_no\_

Class: T\_int\_no AB: N AD: N

Carrier returned from *integer\_no* thread. Within *integer\_no* the number is converted into its binary format.**175. int-no user-declaration directive.**

```

<int-no user-declaration directive 175> ≡
public: T_int_no(long S_no);
       long no();
private: long no_;

```

**176. int-no user-implementation directive.**

```

⟨int-no user-implementation directive 176⟩ ≡
  T_int_no :: T_int_no(long S_no)T_CTOR("int-no", T_Enum :: T_T_int_no_, 0, false, false)
  {
    no_ = S_no;
  }
  long T_int_no :: no()
  {
    return no_;
  }
  ;

```

**177. kw-in-stbl.**

Enum: T\_kw\_in\_stbl\_

Class: kw\_in\_stbl

AB: N

AD: N

**178. kw-in-stbl user-declaration directive.**

```

⟨kw-in-stbl user-declaration directive 178⟩ ≡
public: kw_in_stbl(CAbs_lr1_sym * Fnd_kw_in_stbl);
  CAbs_lr1_sym * keyword_in_stbl();
  void stbl_idx(yacco2 :: UINT Idx);
  int stbl_idx();
private: CAbs_lr1_sym * kw_in_stbl_;
  int stbl_idx_;

```

**179. kw-in-stbl user-implementation directive.**

```

⟨kw-in-stbl user-implementation directive 179⟩ ≡
  kw_in_stbl :: kw_in_stbl(CAbs_lr1_sym * Fnd_kw_in_stbl)T_CTOR("kw-in-stbl", T_Enum :: T_kw_in_stbl_, 0,
    false, false)
  {
    kw_in_stbl_ = Fnd_kw_in_stbl;
    stbl_idx_ = -1;
  }
  CAbs_lr1_sym * kw_in_stbl :: keyword_in_stbl()
  {
    return kw_in_stbl_;
  }
  void kw_in_stbl :: stbl_idx(yacco2 :: UINT Idx)
  {
    stbl_idx_ = Idx;
  }
  int kw_in_stbl :: stbl_idx()
  {
    return stbl_idx_;
  }

```

**180. la-express-source.**

Enum: T\_T\_la\_expr\_src\_

Class: T\_la\_expr\_src

AB: N

AD: N

**181. la-express-source user-declaration directive.**

⟨la-express-source user-declaration directive 181⟩ ≡

**public:** *T\_la\_expr\_src*();**yacco2**::TOKEN\_GAGGLE \* *la\_tok\_can*();**void** *zero\_la\_tok\_can*();**private:** **yacco2**::TOKEN\_GAGGLE \* *la\_tok\_can\_*;**182. la-express-source user-implementation directive.**

⟨la-express-source user-implementation directive 182⟩ ≡

*T\_la\_expr\_src*::*T\_la\_expr\_src*()T\_CTOR("la-express-source", *T\_Enum*::*T\_T\_la\_expr\_src\_*, 0, false, false)

{

*la\_tok\_can\_* = new **yacco2**::TOKEN\_GAGGLE;

}

**void** *T\_la\_expr\_src*::*zero\_la\_tok\_can*()

{

*la\_tok\_can\_* = 0;

}

**yacco2**::TOKEN\_GAGGLE \* *T\_la\_expr\_src*::*la\_tok\_can*()

{

**return** *la\_tok\_can\_*;

}

**183. lint.**

Enum: T\_T\_lint\_

Class: T\_lint

AB: N

AD: N

Gathers the fluff.

**184. lint user-declaration directive.**

⟨lint user-declaration directive 184⟩ ≡

*T\_lint*();**185. lint user-implementation directive.**

⟨lint user-implementation directive 185⟩ ≡

*T\_lint*::*T\_lint*()T\_CTOR("lint", *T\_Enum*::*T\_T\_lint\_*, 0, false, false)

{

*T\_lint**lint\_*;**yacco2**::*CAbs\_lr1\_sym* \* *NS\_yacco2\_terminals*::*PTR\_lint\_* = &*lint\_*;**186. list-of-native-first-set-terminals.**

Enum: T\_T\_list\_of\_native\_first\_set\_terminals\_

Class: T\_list\_of\_native\_first\_set\_terminals

AB: N

AD: N

**187. list-of-transitive-threads.**Enum: `T_T_list_of_transitive_threads_`Class: `T_list_of_transitive_threads`

AB: N

AD: N

**188. list-of-used-threads.**Enum: `T_T_list_of_used_threads_`Class: `T_list_of_used_threads`

AB: N

AD: N

**189. lr1-k-phrase.**Enum: `T_T_lr1_k_phrase_`Class: `T_lr1_k_phrase`

AB: N

AD: N

**190. lr1-k-phrase destructor directive.**`<lr1-k-phrase destructor directive 190> ≡`

```

R→destroy_alphabet();
delete R→filename_id_;
delete R→namespace_id_;
delete R→lrk_sufx_code_;

```

**191. lr1-k-phrase user-declaration directive.**`<lr1-k-phrase user-declaration directive 191> ≡`**public:** `T_lr1_k_phrase`();

```

T_identifier *filename_id();
void filename_id(T_identifier *Id);
T_identifier *namespace_id();
void namespace_id(T_identifier *Id);
void destroy_alphabet();
std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *alphabet();
CAbs_lr1_sym * add_t_to_alphabet(T_terminal_def *T, yacco2::Parser *P);
std::vector < T_terminal_def *> *crt_order();
T_syntax_code *lrk_sufx_code();
void lrk_sufx_code(T_syntax_code *Code);
void phrase_tree(AST *Tree);
AST * phrase_tree();
void add_cweb_marker(AST *Cweb);
AST * cweb_marker();
private: T_identifier *filename_id_;
T_identifier *namespace_id_;
std::map < std::string, NS_yacco2_terminals::T_terminal_def *> alphabet_;
std::vector < T_terminal_def *> crt_order_;
T_syntax_code *lrk_sufx_code_;
AST * phrase_tree_;
AST * cweb_marker_;

```

**192. lr1-k-phrase user-implementation directive.**

```

⟨lr1-k-phrase user-implementation directive 192⟩ ≡
  T_lr1_k_phrase::T_lr1_k_phrase()T_CTOR("lr1-k-phrase", T_Enum::T_T_lr1_k_phrase_,
    &dtor_T_lr1_k_phrase, false, false)
  {
    lrk_sufx_code_ = 0;
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_lr1_k_phrase::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_lr1_k_phrase::cweb_marker()
  {
    return cweb_marker_;
  }
  void T_lr1_k_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_lr1_k_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  T_identifier *T_lr1_k_phrase::filename_id()
  {
    return filename_id_;
  }
  void T_lr1_k_phrase::filename_id(T_identifier *Id)
  {
    filename_id_ = Id;
  }
  T_identifier *T_lr1_k_phrase::namespace_id()
  {
    return namespace_id_;
  }
  void T_lr1_k_phrase::namespace_id(T_identifier *Id)
  {
    namespace_id_ = Id;
  }
  std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *T_lr1_k_phrase::alphabet()
  {
    return &alphabet_;
  }
  std::vector < T_terminal_def *> *T_lr1_k_phrase::crt_order()
  {
    return &crt_order_;
  }

```

```

void T_lr1_k_phrase::destroy_alphabet()
{
    std::vector < T_terminal_def * > ::iterator i = crt_order_.begin();
    std::vector < T_terminal_def * > ::iterator ie = crt_order_.end();
    for ( ; i ≠ ie; ++i) {
        CAbs_lr1_sym * sym = *i;
        delete sym;
    }
    alphabet_.clear();
}

CAbs_lr1_sym * T_lr1_k_phrase::add_t_to_alphabet(T_terminal_def * T, yacco2::Parser * P)
{
    std::stringkey(T-t_name()-c_str());
    std::map < std::string, NS_yacco2_terminals::T_terminal_def * > ::iterator i = alphabet_.find(key);
    if (i ≠ alphabet_.end()) {
        CAbs_lr1_sym * sym = new Err_dup_entry_in_alphabet;
        sym-set_rc(*T, __FILE__, __LINE__);
        return sym;
    }
    alphabet_[key] = T;
    crt_order_.push_back(T);
    return 0;
}

T_syntax_code * T_lr1_k_phrase::lrk_sufx_code()
{
    return lrk_sufx_code_;
}

void T_lr1_k_phrase::lrk_sufx_code(T_syntax_code * Code)
{
    lrk_sufx_code_ = Code;
}

```

**193. monolithic.**

Enum: T\_T\_monolithic\_

Class: T\_monolithic

AB: N

AD: N

**194. no-of-T.**

Enum: T\_T\_no\_of\_T\_

Class: T\_no\_of\_T

AB: N

AD: N

**195. null call thread eosubrule.**

Enum: T\_T\_null\_call\_thread\_eosubrule\_

Class: T\_null\_call\_thread\_eosubrule

AB: N

AD: N

**196. null call thread eosubrule user-declaration directive.**

⟨ null call thread eosubrule user-declaration directive 196 ⟩ ≡

```
public: T_subrule_def * its_subrule_def();  
         T_subrule_def * its_subrule_def();  
void its_subrule_def(T_subrule_def * Its_subrule);  
std::string * grammar_s_enumerate();  
void grammar_s_enumerate(const char *Enumerate);  
AST * tree_node();  
void tree_node(AST * Tree_node);  
rule_def *its_rule_def();  
int element_pos();  
void element_pos(int Pos);  
private: T_subrule_def * its_subrule_def_;  
         std::string grammar_s_enumerate_;  
         AST * tree_node_;  
         int element_pos_;
```



**197. null call thread eosubrule user-implementation directive.**

```

⟨null call thread eosubrule user-implementation directive 197⟩ ≡
  T_null_call_thread_eosubrule :: T_null_call_thread_eosubrule() T_CTOR("null_call_thread_eosubrule",
    T_Enum :: T_T_null_call_thread_eosubrule_, 0, false, false)
  {
    its_subrule_def_ = 0;
    tree_node_ = 0;
    element_pos_ = 0;
  }
  int T_null_call_thread_eosubrule :: element_pos()
  {
    return element_pos_;
  }
  void T_null_call_thread_eosubrule :: element_pos(int Pos)
  {
    element_pos_ = Pos;
  }
  rule_def *T_null_call_thread_eosubrule :: its_rule_def()
  {
    return its_subrule_def_→its_rule_def();
  }
  AST * T_null_call_thread_eosubrule :: tree_node()
  {
    return tree_node_;
  }
  void T_null_call_thread_eosubrule :: tree_node(AST * Tree_node)
  {
    tree_node_ = Tree_node;
  }
  std::string * T_null_call_thread_eosubrule :: grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
  void T_null_call_thread_eosubrule :: grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
  T_subrule_def * T_null_call_thread_eosubrule :: its_subrule_def()
  {
    return its_subrule_def_;
  }
  void T_null_call_thread_eosubrule :: its_subrule_def(T_subrule_def * Its_subrule)
  {
    its_subrule_def_ = Its_subrule;
  }

```

**198. option-err.**

Enum: T\_T\_option\_err\_

Class: T\_option\_err

AB: N

AD: Y

Command line option for  $O_2$  indicating generate Error vocabulary code. Returned from *yacco2\_lcl\_option* thread.

**199. option-p.**

Enum: T\_T\_option\_p\_

Class: T\_option\_p

AB: N

AD: Y

Command line option for  $O_2$  indicating generate cweb type documents. Returned from *yacco2\_lcl\_option* thread.

**200. option-t.**

Enum: T\_T\_option\_t\_

Class: T\_option\_t

AB: N

AD: Y

Command line option for  $O_2$  indicating generate T vocabulary code. Returned from *yacco2\_lcl\_option* thread.

**201. parallel-monitor-phrase.**

Enum: T\_T\_parallel\_monitor\_phrase\_

Class: T\_parallel\_monitor\_phrase

AB: N

AD: N

**202. parallel-monitor-phrase destructor directive.**

⟨parallel-monitor-phrase destructor directive 202⟩ ≡

```
R-remove_mntr_directives();
```

**203. parallel-monitor-phrase user-declaration directive.**

⟨parallel-monitor-phrase user-declaration directive 203⟩ ≡

```
public: T_parallel_monitor_phrase();
```

```
void remove_mntr_directives();
```

```
std::map < std::string, CAbs_lr1_sym *> *mntr_directives_map();
```

```
CAbs_lr1_sym * /* 0 - ok, or error */
```

```
add_directive_to_mntr(CAbs_lr1_sym * Directive, yacco2::Parser * P);
```

```
void phrase_tree(AST * Tree);
```

```
AST * phrase_tree();
```

```
private: map < std::string, CAbs_lr1_sym *> mntr_directives_map_;
```

```
AST * phrase_tree_;
```

**204. parallel-monitor-phrase user-implementation directive.**

```

⟨parallel-monitor-phrase user-implementation directive 204⟩ ≡
  T_parallel_monitor_phrase :: T_parallel_monitor_phrase()T_CTOR("parallel-monitor-phrase",
    T_Enum :: T_T_parallel_monitor_phrase_, &dtor_T_parallel_monitor_phrase, false, false)
  {
    phrase_tree_ = 0;
  }
  void T_parallel_monitor_phrase :: phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_parallel_monitor_phrase :: phrase_tree()
  {
    return phrase_tree_;
  }
  map < string, CAbs_lr1_sym *> *T_parallel_monitor_phrase :: mntr_directives_map()
  {
    return &mntr_directives_map_;
  }
  void T_parallel_monitor_phrase :: remove_mntr_directives()
  {
    std::map < std::string, CAbs_lr1_sym *> ::iterator i = mntr_directives_map_.begin();
    std::map < std::string, CAbs_lr1_sym *> ::iterator ie = mntr_directives_map_.end();
    for (; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = i->second;
      delete sym;
    }
  }
  CAbs_lr1_sym * /* 0 - ok, or error */
  T_parallel_monitor_phrase :: add_directive_to_mntr(CAbs_lr1_sym * Directive, yacco2::Parser * P)
  {
    map < std::string, CAbs_lr1_sym *> ::iterator i;
    std::string key(Directive->id_);
    i = mntr_directives_map_.find(key);
    if (i ≡ mntr_directives_map_.end()) {
      mntr_directives_map_[key] = Directive;
      return 0;
    }
    CAbs_lr1_sym * sym = new Err_duplicate_directive;
    sym->set_rc(*Directive, __FILE__, __LINE__);
    remove_mntr_directives();
    return sym;
  }

```

**205. parallel-parser-phrase.**

Enum: T\_T\_parallel\_parser\_phrase\_

Class: T\_parallel\_parser\_phrase

AB: N

AD: N

**206. parallel-parser-phrase destructor directive.**

⟨parallel-parser-phrase destructor directive 206⟩ ≡  
**delete** *R-la\_bndry\_*;  
**delete** *R-pp\_funct\_*;

**207. parallel-parser-phrase user-declaration directive.**

⟨parallel-parser-phrase user-declaration directive 207⟩ ≡  
**public: T\_parallel\_parser\_phrase**();  
*T\_parallel\_thread\_function* \* *pp\_funct*();  
**void** *pp\_funct*(*T\_parallel\_thread\_function* \* *PP\_funct*);  
*T\_parallel\_la\_boundary* \* *la\_bndry*();  
**void** *la\_bndry*(*T\_parallel\_la\_boundary* \* *La\_bndry*);  
**void** *phrase\_tree*(AST \* *Tree*);  
AST \* *phrase\_tree*();  
**void** *add\_cweb\_marker*(AST \* *Cweb*);  
AST \* *cweb\_marker*();  
**private: T\_parallel\_thread\_function** \* *pp\_funct\_*;  
*T\_parallel\_la\_boundary* \* *la\_bndry\_*;  
AST \* *phrase\_tree\_*;  
AST \* *cweb\_marker\_*;

**208. parallel-parser-phrase user-implementation directive.**

```

⟨parallel-parser-phrase user-implementation directive 208⟩ ≡
  T_parallel_parser_phrase :: T_parallel_parser_phrase()T_CTOR("parallel-parser-phrase",
    T_Enum :: T_T_parallel_parser_phrase_, &dtor_T_parallel_parser_phrase, false, false)
  {
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_parallel_parser_phrase :: add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_parallel_parser_phrase :: cweb_marker()
  {
    return cweb_marker_;
  }
  void T_parallel_parser_phrase :: phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_parallel_parser_phrase :: phrase_tree()
  {
    return phrase_tree_;
  }
  T_parallel_thread_function * T_parallel_parser_phrase :: pp_funct()
  {
    return pp_funct_;
  }
  void T_parallel_parser_phrase :: pp_funct(T_parallel_thread_function * PP_funct)
  {
    pp_funct_ = PP_funct;
  }
  T_parallel_la_boundary * T_parallel_parser_phrase :: la_bndry()
  {
    return la_bndry_;
  }
  void T_parallel_parser_phrase :: la_bndry(T_parallel_la_boundary * La_bndry)
  {
    la_bndry_ = La_bndry;
  }

```

**209. preamble.**

Enum: T\_T\_preamble\_

Class: T\_preamble

AB: N

AD: N

**210. raw-char.**

Enum: T\_T\_raw\_char\_

Class: T\_raw\_char

AB: N

AD: N

A universal carrier from the raw text character into c++ objects. Still don't know why there is also a *raw - characters* terminal.

**211. raw-char destructor directive.**

```
<raw-char destructor directive 211> ≡
  if (R-raw_char_ ≠ 0) delete R-raw_char();
```

**212. raw-char user-declaration directive.**

```
<raw-char user-declaration directive 212> ≡
public: T_raw_char(CAbs_lr1_sym * Raw_char);
       CAbs_lr1_sym * raw_char() const;
       void zero_out_raw_char();
private: CAbs_lr1_sym * raw_char_;
```

**213. raw-char user-implementation directive.**

```
<raw-char user-implementation directive 213> ≡
  T_raw_char :: T_raw_char(CAbs_lr1_sym * Raw_char) T_CTOR("raw-char", T_Enum :: T_T_raw_char_, 0,
    false, false)
  {
    raw_char_ = Raw_char;
  }
  void T_raw_char :: zero_out_raw_char()
  {
    raw_char_ = 0;
  }
  CAbs_lr1_sym * T_raw_char :: raw_char() const
  {
    return raw_char_;
  }
```

**214. rc-phrase.**

Enum: T\_T\_rc\_phrase\_

Class: T\_rc\_phrase

AB: N

AD: N

**215. rc-phrase destructor directive.**

```
<rc-phrase destructor directive 215> ≡
  R-destroy_alphabet();
  delete R-filename_id_;
  delete R-namespace_id_;
```

**216. rcphrase user-declaration directive.**

⟨rcphrase user-declaration directive 216⟩ ≡

```
public: T_rcphrase();
    T_identifier *filename_id();
    void filename_id(T_identifier *Id);
    T_identifier *namespace_id();
    void namespace_id(T_identifier *Id);
    void destroy_alphabet();
    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *alphabet();
    CAbs_lr1_sym *add_t_to_alphabet(T_terminal_def *T, yacco2::Parser *P);
    std::vector < T_terminal_def *> *crt_order();
    void phrase_tree(AST * Tree);
    AST * phrase_tree();
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
private: T_identifier *filename_id_;
    T_identifier *namespace_id_;
    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> alphabet_;
    std::vector < T_terminal_def *> crt_order_;
    AST * phrase_tree_;
    AST * cweb_marker_;
```

**217. rc-phrase user-implementation directive.**

⟨rc-phrase user-implementation directive 217⟩ ≡

```

T_rc_phrase::T_rc_phrase()T_CTOR("rc-phrase", T_Enum::T_T_rc_phrase_, &dtor_T_rc_phrase,
    false, false)
{
    phrase_tree_ = 0;
    cweb_marker_ = 0;
}
void T_rc_phrase::add_cweb_marker(AST * Cweb)
{
    cweb_marker_ = Cweb;
}
AST * T_rc_phrase::cweb_marker()
{
    return cweb_marker_;
}
void T_rc_phrase::phrase_tree(AST * Tree)
{
    phrase_tree_ = Tree;
}
AST * T_rc_phrase::phrase_tree()
{
    return phrase_tree_;
}
T_identifier * T_rc_phrase::filename_id()
{
    return filename_id_;
}
void T_rc_phrase::filename_id(T_identifier * Id)
{
    filename_id_ = Id;
}
T_identifier * T_rc_phrase::namespace_id()
{
    return namespace_id_;
}
void T_rc_phrase::namespace_id(T_identifier * Id)
{
    namespace_id_ = Id;
}
std::map < std::string, NS_yacco2_terminals::T_terminal_def * > * T_rc_phrase::alphabet()
{
    return &alphabet_;
}
std::vector < T_terminal_def * > * T_rc_phrase::crt_order()
{
    return &crt_order_;
}
void T_rc_phrase::destroy_alphabet()

```



```

{
  std::vector < T_terminal_def *> ::iterator i = crt_order_.begin();
  std::vector < T_terminal_def *> ::iterator ie = crt_order_.end();
  for ( ; i ≠ ie; ++i) {
    CAbs_lr1_sym * sym = *i;
    delete sym;
  }
  alphabet_.clear();
}
CAbs_lr1_sym * T_rc_phrase::add_t_to_alphabet(T_terminal_def * T, yacco2::Parser * P)
{
  std::stringkey(T->t_name()->c_str());
  std::map < std::string, NS_yacco2_terminals::T_terminal_def *> ::iterator i = alphabet_.find(key);
  if (i ≠ alphabet_.end()) {
    CAbs_lr1_sym * sym = new Err_dup_entry_in_alphabet;
    sym->set_rc(*T, __FILE__, __LINE__);
    return sym;
  }
  alphabet_[key] = T;
  crt_order_.push_back(T);
  return 0;
}

```

**218. refered-T.**

Enum: T\_refered\_T\_

Class: refered\_T

AB: N

AD: N

**219. refered-T user-declaration directive.**

⟨refered-T user-declaration directive 219⟩ ≡

```

public: refered_T(T_in_stbl &Term_in_stbl);
  T_in_stbl *t_in_stbl();
  T_terminal_def * its_t_def();
  T_subrule_def * its_subrule_def();
  void its_subrule_def(T_subrule_def * Its_subrule);
  std::string * grammar_s_enumerate();
  void grammar_s_enumerate(const char *Enumerate);
  AST * tree_node();
  void tree_node(AST * Tree_node);
  rule_def *its_rule_def();
  int element_pos();
  void element_pos(int Pos);
private: T_in_stbl *t_in_stbl_;
  T_subrule_def * its_subrule_def_;
  std::string grammar_s_enumerate_;
  AST * tree_node_;
  int element_pos_;

```

**220. referred-T user-implementation directive.**

```

⟨referred-T user-implementation directive 220⟩ ≡
  referred_T::referred_T(T_in_stbl &Term_in_stbl)T_CTOR("referred-T", T_Enum::T_referred_T_, 0,
    false, false)
  {
    t_in_stbl_ = &Term_in_stbl;
    its_subrule_def_ = 0;
    tree_node_ = 0;
    element_pos_ = 0;
  }
  int referred_T::element_pos()
  {
    return element_pos_;
  }
  void referred_T::element_pos(int Pos)
  {
    element_pos_ = Pos;
  }
  T_terminal_def *referred_T::its_t_def()
  {
    return t_in_stbl_→t_def();
  }
  rule_def *referred_T::its_rule_def()
  {
    return its_subrule_def_→its_rule_def();
  }
  AST *referred_T::tree_node()
  {
    return tree_node_;
  }
  void referred_T::tree_node(AST *Tree_node)
  {
    tree_node_ = Tree_node;
  }
  std::string *referred_T::grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
  void referred_T::grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
  T_in_stbl *referred_T::t_in_stbl()
  {
    return t_in_stbl_;
  }
  T_subrule_def *referred_T::its_subrule_def()
  {
    return its_subrule_def_;
  }

```

```

}
void referred_T::its_subrule_def(T_subrule_def * Its_subrule)
{
    its_subrule_def = Its_subrule;
}

```

**221. referred-rule.**

Enum: T\_referred\_rule\_

Class: referred\_rule

AB: N

AD: N

**222. referred-rule user-declaration directive.**⟨referred-rule user-declaration directive [222](#)⟩ ≡

```

public: referred_rule(rule_in_stbl &Rule_in_stbl);
    rule_in_stbl *Rule_in_stbl();
    T_subrule_def *its_subrule_def();
    void its_subrule_def(T_subrule_def * Its_subrule);
    std::string *grammar_s_enumerate();
    void grammar_s_enumerate(const char *Enumerate);
    AST *tree_node();
    void tree_node(AST * Tree_node);
    rule_def *its_rule_def();
    int element_pos();
    void element_pos(int Pos);
private: rule_in_stbl *rule_in_stbl_;
    T_subrule_def *its_subrule_def_;
    std::string grammar_s_enumerate_;
    AST *tree_node_;
    int element_pos_;

```

**223. referred-rule user-implementation directive.**

```

⟨referred-rule user-implementation directive 223⟩ ≡
  referred_rule::referred_rule(rule_in_stbl &Rule_in_stbl)T_CTOR("referred-rule",
    T_Enum::T_referred_rule_, 0, false, false)
  {
    rule_in_stbl_ = &Rule_in_stbl;
    its_subrule_def_ = 0;
    tree_node_ = 0;
    element_pos_ = 0;
  }
  int referred_rule::element_pos()
  {
    return element_pos_;
  }
  void referred_rule::element_pos(int Pos)
  {
    element_pos_ = Pos;
  }
  rule_def *referred_rule::its_rule_def()
  {
    return rule_in_stbl_→r_def();
  }
  AST *referred_rule::tree_node()
  {
    return tree_node_;
  }
  void referred_rule::tree_node(AST * Tree_node)
  {
    tree_node_ = Tree_node;
  }
  std::string *referred_rule::grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
  void referred_rule::grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
  T_subrule_def *referred_rule::its_subrule_def()
  {
    return its_subrule_def_;
  }
  void referred_rule::its_subrule_def(T_subrule_def * Its_subrule)
  {
    its_subrule_def_ = Its_subrule;
  }
  rule_in_stbl *referred_rule::Rule_in_stbl()
  {
    return rule_in_stbl_;
  }

```

**224. rule-def.**

Enum: T\_rule\_def\_

Class: rule\_def

AB: N

AD: N

---

**225. rule-def destructor directive.**

⟨rule-def destructor directive 225⟩ ≡

**delete** *R-parallel\_mntr\_;***delete** *R-rule\_lhs\_;***delete** *R-subrules\_;*

**226. rule-def user-declaration directive.**

⟨ rule-def user-declaration directive 226 ⟩ ≡

```

public: rule_def(const char *Key);
    T_rule_lhs_phrase * rule_lhs();
    void rule_lhs(T_rule_lhs_phrase * Lhs);
    int rule_no();
    void rule_no(int Rule_no);
    int lhs_use_cnt();
    void lhs_use_cnt(int Use_cnt);
    int rhs_use_cnt();
    void rhs_use_cnt(int Use_cnt);
    bool recursive();
    void recursive(bool Recursive);
    void parallel_mntr(T_parallel_monitor_phrase * Mntr);
    T_parallel_monitor_phrase * parallel_mntr();
    T_subrules_phrase * subrules();
    void subrules(T_subrules_phrase * Subrules);
    std::string * rule_name();
    bool autodelete();
    void autodelete(bool Ton);
    bool autoabort();
    void autoabort(bool Ton);
    bool epsilon();
    void epsilon(bool Ton);
    bool derive_t();
    void derive_t(bool Ton);
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
    void bld_its_tree();
    AST * rule_s_tree();
    void enum_id(int Id);
    int enum_id(); std::set < T_in_stbl * > *first_set();
    void add_to_first_set(T_in_stbl &T); std::set < T_called_thread_eosubrule
        * > *called_thread_first_set();
    void add_to_called_thread_first_set(T_called_thread_eosubrule *T);
    std::string * grammar_s_enumerate();
    void grammar_s_enumerate(const char *Enumerate);
    void add_closure_rule_making_up_first_set(rule_in_stbl *Rule); std::set < rule_in_stbl
        * > *closure_rules_making_up_first_set();
    void add_rule_adding_T_in_first_set(T_in_stbl *T, std::string * Rule_enumerate); std::map < T_in_stbl
        *, std::set < std::string * > > *rule_adding_T_in_first_set();
    AST * rule_use_skeleton();
    void rule_use_skeleton(AST * Tree);
private: std::string rule_name_;
    T_parallel_monitor_phrase * parallel_mntr_;
    T_subrules_phrase * subrules_;
    T_rule_lhs_phrase * rule_lhs_;
    bool auto_delete_;

```

```

bool auto_abort_;
AST * cweb_marker_;
AST * its_tree_;
int enum_id_;
int rule_no_;
int lhs_use_cnt_; /* for rule recycling */
int rhs_use_cnt_; /* for rule recycling */
bool recursive_;
bool epsilon_;
bool derive_t_;
std::string grammar_s_enumerate_; std::set < T_in_stbl * > first_set_; std::set
  < T_called_thread_eosubrule * > called_thread_first_set_; std::set <
  rule_in_stbl * > closure_rules_making_up_first_set_; std::map < T_in_stbl *,
  std::set < std::string *>> rules_adding_T_in_first_set_;
AST * rule_use_skeleton_;

```

**227. rule-def user-implementation directive.**

```

⟨ rule-def user-implementation directive 227 ⟩ ≡
  rule_def :: rule_def(const char *Key)T_CTOR("rule-def", T_Enum :: T_rule_def_, &dtor_rule_def, false,
    false)
  {
    auto_delete_ = false;
    auto_abort_ = false;
    rule_no_ = 0; /* starts cnt at 1 */
    recursive_ = NO;
    lhs_use_cnt_ = 0;
    rhs_use_cnt_ = 0;
    rule_use_skeleton_ = 0;
    parallel_mntr_ = 0;
    subrules_ = 0;
    rule_lhs_ = 0;
    rule_name_ += Key;
    epsilon_ = false;
    derive_t_ = false;
    cweb_marker_ = 0;
    its_tree_ = 0;
    enum_id_ = -1;
    first_set_.clear();
  }
std::set < rule_in_stbl * > *rule_def :: closure_rules_making_up_first_set()
{
  return &closure_rules_making_up_first_set_;
}
AST * rule_def :: rule_use_skeleton()
{
  return rule_use_skeleton_;
}
void rule_def :: rule_use_skeleton(AST * Tree)
{
  rule_use_skeleton_ = Tree;
}
std::map < T_in_stbl *, std::set < std::string * >> *rule_def :: rule_adding_T_in_first_set()
{
  return &rules_adding_T_in_first_set_;
}
int rule_def :: rule_no()
{
  return rule_no_;
}
void rule_def :: rule_no(int Rule_no)
{
  rule_no_ = Rule_no;
}
bool rule_def :: recursive()
{
  return recursive_;
}

```



```

void rule_def :: recursive(bool Recursive)
{
    recursive_ = Recursive;
}
int rule_def :: lhs_use_cnt()
{
    return lhs_use_cnt_;
}
;
void rule_def :: lhs_use_cnt(int Use_cnt)
{
    lhs_use_cnt_ = Use_cnt;
}
int rule_def :: rhs_use_cnt()
{
    return rhs_use_cnt_;
}
void rule_def :: rhs_use_cnt(int Use_cnt)
{
    rhs_use_cnt_ = Use_cnt;
}
void rule_def :: add_closure_rule_making_up_first_set(rule_in_stbl *Rule)
{
    if (closure_rules_making_up_first_set_.find(Rule) ≠ closure_rules_making_up_first_set_.end()) return;
    closure_rules_making_up_first_set_.insert(Rule);
}
void rule_def :: add_rule_adding_T_in_first_set(T_in_stbl *T, std::string * Rule_enumerate){ std::map
    < T_in_stbl *, std::set < std::string *>> ::iterator i; std::map < T_in_stbl *,
    std::set < std::string *>> ::iterator ie;
    ie = rules_adding_T_in_first_set_.end();
    i = rules_adding_T_in_first_set_.find(T);
    if (i ≡ ie) {
        rules_adding_T_in_first_set_[T] = std::set < std::string *> ();
        i = rules_adding_T_in_first_set_.find(T);
        i->second.insert(Rule_enumerate);
        return;
    }
    i->second.insert(Rule_enumerate); } std::string * rule_def :: grammar_s_enumerate()
{
    return &grammar_s_enumerate_;
}
void rule_def :: grammar_s_enumerate(const char *Enumerate)
{
    grammar_s_enumerate_ += Enumerate;
}
std::set < T_in_stbl * > *rule_def :: first_set()
{
    return &first_set_;
}
std::set < T_called_thread_eosubrule * > *rule_def :: called_thread_first_set()

```

```

{
  return &called_thread_first_set_;
}
void rule_def::add_to_first_set(T_in_stbl &T){ T_in_stbl *t = &T; std::set < T_in_stbl
  * > ::iterator i;
  i = first_set_.find(t);
  if (i == first_set_.end()) {
    first_set_.insert(t);
  }
} void rule_def::add_to_called_thread_first_set(T_called_thread_eosubrule *T){ std::set <
  T_called_thread_eosubrule * > ::iterator i;
  i = called_thread_first_set_.find(T);
  if (i == called_thread_first_set_.end()) {
    called_thread_first_set_.insert(T);
  }
} void rule_def::enum_id(int Id)
{
  enum_id_ = Id;
}
int rule_def::enum_id()
{
  return enum_id_;
}
void rule_def::bld_its_tree()
{
  its_tree_ = new AST(*this);
  T_subrules_phrase *subrules_ph = subrules();
  std::vector < T_subrule_def * > *subrule_list = subrules_ph->subrules();
  std::vector < T_subrule_def * > ::iterator sri = subrule_list->begin();
  std::vector < T_subrule_def * > ::iterator srie = subrule_list->end();
  AST * subrule_lvl = 0;
  for (; sri != srie; ++sri) { /* walk subrules */
    T_subrule_def * srdef = *sri;
    AST * srt = srdef->subrule_s_tree();
    if (subrule_lvl == 0) {
      AST::crt_tree_of_1son(*its_tree_, *srt);
      subrule_lvl = srt;
    }
    else { /* subrule brothers */
      AST::join_sts(*subrule_lvl, *srt);
      subrule_lvl = srt;
    }
  }
}
AST * rule_def::rule_s_tree()
{
  return its_tree_;
}
void rule_def::add_cweb_marker(AST * Cweb)
{

```

```

    cweb_marker_ = Cweb;
}
AST * rule_def :: cweb_marker()
{
    return cweb_marker_;
}
T_rule_lhs_phrase * rule_def :: rule_lhs()
{
    return rule_lhs_;
}
void rule_def :: rule_lhs(T_rule_lhs_phrase * Lhs)
{
    rule_lhs_ = Lhs;
}
void rule_def :: parallel_mntr(T_parallel_monitor_phrase * Mntr)
{
    parallel_mntr_ = Mntr;
}
T_parallel_monitor_phrase * rule_def :: parallel_mntr()
{
    return parallel_mntr_;
}
T_subrules_phrase * rule_def :: subrules()
{
    return subrules_;
}
void rule_def :: subrules(T_subrules_phrase * Subrules)
{
    subrules_ = Subrules;
}
std::string * rule_def :: rule_name()
{
    return &rule_name_;
}
bool rule_def :: autodelete()
{
    return auto_delete_;
}
void rule_def :: autodelete(bool Ton)
{
    auto_delete_ = Ton;
}
bool rule_def :: autoabort()
{
    return auto_abort_;
}
void rule_def :: autoabort(bool Ton)
{
    auto_abort_ = Ton;
}

```

```

    }
    bool rule_def :: epsilon()
    {
        return epsilon_;
    }
    void rule_def :: epsilon(bool Ton)
    {
        epsilon_ = Ton;
    }
    bool rule_def :: derive_t()
    {
        return derive_t_;
    }
    void rule_def :: derive_t(bool Ton)
    {
        derive_t_ = Ton;
    }
}

```

**228. rule-in-stbl.**

Enum: T\_rule\_in\_stbl\_

Class: rule\_in\_stbl

AB: N

AD: N

**229. rule-in-stbl user-declaration directive.**

⟨rule-in-stbl user-declaration directive 229⟩ ≡

```

public: rule_in_stbl(rule_def &Rule_def);
    rule_def *r_def(); std::list <referred_rule * > *xref_r();
    void add_R_into_xref(referred_rule &R);
    void stbl_idx(yacco2::UINT Idx);
    int stbl_idx();
private: rule_def *r_def_; std::list <referred_rule * > xref_r_;
    int stbl_idx_;

```

**230. rule-in-stbl user-implementation directive.**

```

⟨rule-in-stbl user-implementation directive 230⟩ ≡
  rule_in_stbl::rule_in_stbl(rule_def &Rule_def)T_CTOR("rule-in-stbl", T_Enum::T_rule_in_stbl, 0,
    false, false)
  {
    r_def_ = &Rule_def;
    stbl_idx_ = -1;
  }
  rule_def *rule_in_stbl::r_def()
  {
    return r_def_;
  }
  std::list < referred_rule * > *rule_in_stbl::xref_r()
  {
    return &xref_r_;
  };
  void rule_in_stbl::add_R_into_xref(referred_rule &R)
  {
    xref_r_.push_back(&R);
  }
  void rule_in_stbl::stbl_idx(yacco2::UINT Idx)
  {
    stbl_idx_ = Idx;
  }
  int rule_in_stbl::stbl_idx()
  {
    return stbl_idx_;
  }

```

**231. rule-lhs-phrase.**

Enum: T\_T\_rule\_lhs\_phrase\_

Class: T\_rule\_lhs\_phrase

AB: N

AD: N

**232. rule-lhs-phrase destructor directive.**

```

⟨rule-lhs-phrase destructor directive 232⟩ ≡
  R_remove_lhs_directives();

```

**233. rule-lhs-phrase user-declaration directive.**

⟨rule-lhs-phrase user-declaration directive 233⟩ ≡

```

public: T_rule_lhs_phrase();
    void remove_lhs_directives();
    std::map < std::string, CAbs_lr1_sym *> lhs_directives_map();
    CAbs_lr1_sym * /* 0 - ok, or error */
    add_directive_to_lhs(CAbs_lr1_sym * Directive, yacco2::Parser * P);
    void phrase_tree(AST * Tree);
    AST * phrase_tree();
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
private: map < std::string, CAbs_lr1_sym *> lhs_directives_map_;
    AST * phrase_tree_;
    AST * cweb_marker_;

```

**234. rule-lhs-phrase user-implementation directive.**

```

⟨rule-lhs-phrase user-implementation directive 234⟩ ≡
  T_rule_lhs_phrase :: T_rule_lhs_phrase() T_CTOR("rule-lhs-phrase", T_Enum :: T_T_rule_lhs_phrase_,
    & dtor_T_rule_lhs_phrase, false, false)
  {
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_rule_lhs_phrase :: add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_rule_lhs_phrase :: cweb_marker()
  {
    return cweb_marker_;
  }
  void T_rule_lhs_phrase :: phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_rule_lhs_phrase :: phrase_tree()
  {
    return phrase_tree_;
  }
  map < std::string, CAbs_lr1_sym *> *T_rule_lhs_phrase :: lhs_directives_map()
  {
    return &lhs_directives_map_;
  }
  void T_rule_lhs_phrase :: remove_lhs_directives()
  {
    std::map < std::string, CAbs_lr1_sym *> ::iterator i = lhs_directives_map_.begin();
    std::map < std::string, CAbs_lr1_sym *> ::iterator ie = lhs_directives_map_.end();
    for (; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = i->second;
      delete sym;
    }
  }
  CAbs_lr1_sym * /* 0 - ok, or error */
  T_rule_lhs_phrase :: add_directive_to_lhs(CAbs_lr1_sym * Directive, yacco2::Parser * P)
  {
    map < std::string, CAbs_lr1_sym *> ::iterator i;
    stringkey(Directive-id_);
    i = lhs_directives_map_.find(key);
    if (i ≡ lhs_directives_map_.end()) {
      lhs_directives_map_[key] = Directive;
      return 0;
    }
    CAbs_lr1_sym * sym = new Err_duplicate_directive;
    sym->set_rc(*Directive, __FILE__, __LINE__);
    remove_lhs_directives();
  }

```

```

    return sym;
}

```

**235. rules-phrase.**

Enum: T\_T\_rules\_phrase\_

Class: T\_rules\_phrase

AB: N

AD: N

**236. rules-phrase destructor directive.**

⟨rules-phrase destructor directive 236⟩ ≡

```

R-destroy_rules_alphabet();

```

**237. rules-phrase user-declaration directive.**

⟨rules-phrase user-declaration directive 237⟩ ≡

```

public: T_rules_phrase();

```

```

    void destroy_rules_alphabet(); std::map < std::string , rule_def * > *rules_alphabet();

```

```

    CAbs_lr1_sym * add_r_to_alphabet(rule_def *R, yacco2::Parser * P); std::vector < rule_def
        * > *crt_order();

```

```

    void rules_tree(AST * Tree);

```

```

    void phrase_tree(AST * Tree);

```

```

    AST * phrase_tree();

```

```

    void add_cweb_marker(AST * Cweb);

```

```

    AST * cweb_marker(); private: std::map < std::string , rule_def * > rules_alphabet_; std::vector <
        rule_def * > crt_order_;

```

```

    AST * phrase_tree_;

```

```

    AST * cweb_marker_;

```



**238. rules-phrase user-implementation directive.**

```

⟨ rules-phrase user-implementation directive 238 ⟩ ≡
  T_rules_phrase::T_rules_phrase( )T_CTOR("rules-phrase", T_Enum::T_T_rules_phrase_,
    &dctor_T_rules_phrase, false, false)
  {
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
void T_rules_phrase::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
AST * T_rules_phrase::cweb_marker( )
  {
    return cweb_marker_;
  }
void T_rules_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
AST * T_rules_phrase::phrase_tree( )
  {
    return phrase_tree_;
  }
std::map < string , rule_def * > *T_rules_phrase::rules_alphabet( )
  {
    return &rules_alphabet_;
  }
void T_rules_phrase::destroy_rules_alphabet( ) { std::vector < rule_def
  * > ::iterator i = crt_order_.begin( ); std::vector < rule_def * > ::iterator ie = crt_order_.end( );
  for ( ; i ≠ ie; ++i) {
    CAbs_lr1_sym * sym = *i;
    delete sym;
  }
  rules_alphabet_.clear( ); } CAbs_lr1_sym * T_rules_phrase::add_r_to_alphabet(rule_def
  *R, yacco2::Parser *P) { std::stringkey (R-rule_name( )-c_str( )); std::map < std::string
  , rule_def * > ::iterator i = rules_alphabet_.find(key);
  if (i ≠ rules_alphabet_.end( )) {
    CAbs_lr1_sym * sym = new Err_dup_entry_in_alphabet;
    sym->set_rc(*R, __FILE__, __LINE__);
    return sym;
  }
  rules_alphabet_[key] = R;
  crt_order_.push_back(R);
  return 0; } std::vector < rule_def * > *T_rules_phrase::crt_order( )
  {
    return &crt_order_;
  }

```

**239. subrule-def.**

Enum: T.T\_subrule\_def\_

Class: T\_subrule\_def

AB: N

AD: N

The *its\_grammar\_s\_pos\_* provides its position within the grammar. Not important in gening of the grammar but used in the cross reference document. As the address of the *T\_subrule\_def* object does not guarantee ascending order, this does.

---

**240. subrule-def destructor directive.** $\langle \text{subrule-def destructor directive } 240 \rangle \equiv$  $R\text{-remove\_subrule\_directives}();$  $R\text{-remove\_subrule\_elems\_from\_vector}();$

**241. subrule-def user-declaration directive.**

```

⟨subrule-def user-declaration directive 241⟩ ≡
public: T_subrule_def();
    int subrule_no_of_rule();
    void subrule_no_of_rule(int Subrule_no);
    void remove_subrule_directives();
    std::map < std::string, CAbs_lr1_sym *> *subrule_directives();
    CAbs_lr1_sym * /* 0 - ok, or error */
    add_directive_to_subrule(CAbs_lr1_sym * Directive, yacco2::Parser * P);
    std::vector < CAbs_lr1_sym *> *subrule_elems();
    void add_elem_to_subrule_vector(CAbs_lr1_sym * Elem);
    int no_of_elems();
    void remove_subrule_elems_from_vector();
    bool epsilon();
    void epsilon(bool Epsilon);
    bool derive_t();
    void derive_t(bool Epsilon);
    rule_def *its_rule_def();
    void its_rule_def(rule_def *Its_rule_def);
    void add_cweb_marker(AST * Cweb);
    AST * cweb_marker();
    void bld_its_tree();
    AST * subrule_s_tree();
    std::string * grammar_s_enumerate();
    void grammar_s_enumerate(const char *Enumerate);
    int its_grammar_s_pos();
    void its_grammar_s_pos(int Pos);
private: std::map < std::string, CAbs_lr1_sym *> subrule_directives_;
    std::vector < CAbs_lr1_sym *> subrule_elems_;
    int subrule_no_of_rule_; /* rel 1 */
    int no_of_elems_;
    bool epsilon_;
    bool derive_t_;
    rule_def *its_rule_def_;
    AST * cweb_marker_;
    AST * its_tree_;
    int its_grammar_s_pos_;
    std::string grammar_s_enumerate_;

```

**242. subrule-def user-implementation directive.**

```

⟨subrule-def user-implementation directive 242⟩ ≡
  T_subrule_def :: T_subrule_def() T_CTOR("subrule-def", T_Enum :: T_T_subrule_def_,
    & dtor_T_subrule_def, false, false)
  {
    subrule_no_of_rule_ = 0;
    no_of_elems_ = 0;
    epsilon_ = false;
    derive_t_ = false;
    its_rule_def_ = 0;
    cweb_marker_ = 0;
    its_tree_ = 0;
    its_grammar_s_pos_ = 0;
  }
int T_subrule_def :: its_grammar_s_pos()
  {
    return its_grammar_s_pos_;
  }
void T_subrule_def :: its_grammar_s_pos(int Pos)
  {
    its_grammar_s_pos_ = Pos;
  }
int T_subrule_def :: subrule_no_of_rule()
  {
    return subrule_no_of_rule_;
  }
void T_subrule_def :: subrule_no_of_rule(int Subrule_no)
  {
    subrule_no_of_rule_ = Subrule_no;
  }
std::string * T_subrule_def :: grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
void T_subrule_def :: grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
void T_subrule_def :: bld_its_tree(){ its_tree_ = new AST(*this);
  std::vector < CAbs_lr1_sym *> *element_list = subrule_elems();
  std::vector < CAbs_lr1_sym *> :: iterator ei = element_list->begin();
  std::vector < CAbs_lr1_sym *> :: iterator eie = element_list->end();
  AST * element_lvl = 0; for ( ; ei ≠ eie; ++ei) { /* walk elements of subrule */
    CAbs_lr1_sym * elem_def = *ei;
    AST * elemtt = new AST(*elem_def);
    int id = elem_def->enumerated_id_;
    using namespace NS_yacco2_terminals; switch (id) {
  case T_Enum :: T_referred_rule_:
    {
      referred_rule *rrule = (referred_rule *) elem_def;

```

```

    rrule-tree_node(elemtt);
    break;
}
case T_Enum::T_T_eosubrule_: { T_eosubrule * eos = ( T_eosubrule * ) elem_def;
eos-tree_node(elemtt);
break; }
case T_Enum::T_refered_T_:
{
    refered_T *rt = (refered_T *) elem_def;
    rt-tree_node(elemtt);
    break;
}
}
if (element_lvl == 0) {
    AST::crt_tree_of_1son(*its_tree_,*elemtt);
    element_lvl = elemtt;
}
else { /* element brothers */
    AST::join_sts(*element_lvl,*elemtt);
    element_lvl = elemtt;
}
} } AST * T_subrule_def::subrule_s_tree()
{
    return its_tree_;
}
void T_subrule_def::add_cweb_marker(AST * Cweb)
{
    cweb_marker_ = Cweb;
}
AST * T_subrule_def::cweb_marker()
{
    return cweb_marker_;
}
rule_def *T_subrule_def::its_rule_def()
{
    return its_rule_def_;
}
void T_subrule_def::its_rule_def(rule_def *Its_rule_def)
{
    its_rule_def_ = Its_rule_def;
}
std::vector < CAbs_lr1_sym *> *T_subrule_def::subrule_elems()
{
    return &subrule_elems_;
}
bool T_subrule_def::epsilon()
{
    return epsilon_;
}
void T_subrule_def::epsilon(bool Epsilon)

```

```

{
  epsilon_ = Epsilon;
}
bool T_subrule_def :: derive_t()
{
  return derive_t_;
}
void T_subrule_def :: derive_t(bool Ton)
{
  derive_t_ = Ton;
}
void T_subrule_def :: add_elem_to_subrule_vector(CAbs_lr1_sym * Elem)
{
  ++no_of_elems_;
  subrule_elems_.push_back(Elem);
}
int T_subrule_def :: no_of_elems()
{
  return no_of_elems_;
}
map < std::string, CAbs_lr1_sym *> *T_subrule_def :: subrule_directives()
{
  return &subrule_directives_;
}
void T_subrule_def :: remove_subrule_directives()
{
  std::map < std::string, CAbs_lr1_sym *> ::iterator i = subrule_directives_.begin();
  std::map < std::string, CAbs_lr1_sym *> ::iterator ie = subrule_directives_.end();
  for (; i != ie; ++i) {
    CAbs_lr1_sym * sym = i->second;
    delete sym;
  }
}
CAbs_lr1_sym * /* 0 - ok, or error */
T_subrule_def :: add_directive_to_subrule(CAbs_lr1_sym * Directive, yacco2::Parser * P)
{
  std::map < std::string, CAbs_lr1_sym *> ::iterator i;
  std::string key(Directive->id_);
  i = subrule_directives_.find(key);
  if (i == subrule_directives_.end()) {
    subrule_directives_[key] = Directive;
    return 0;
  }
  CAbs_lr1_sym * sym = new Err_duplicate_directive;
  sym->set_rc(*Directive, __FILE__, __LINE__);
  remove_subrule_directives();
  return sym;
}
void T_subrule_def :: remove_subrule_elems_from_vector()
{

```

```

std::vector < CAbs_lr1_sym *> ::iterator i = subrule_elems_.begin();
std::vector < CAbs_lr1_sym *> ::iterator ie = subrule_elems_.end();
for ( ; i ≠ ie; ++i) {
    delete *i;
}

```

**243. subrules-phrase.**

Enum: T\_T\_subrules\_phrase\_

Class: T\_subrules\_phrase

AB: N

AD: N

**244. subrules-phrase destructor directive.**

⟨subrules-phrase destructor directive 244⟩ ≡

*R*-destroy\_subrules();**245. subrules-phrase user-declaration directive.**

⟨subrules-phrase user-declaration directive 245⟩ ≡

**public:** T\_subrules\_phrase();**void** destroy\_subrules();**std**::vector < T\_subrule\_def \*> \*subrules();**void** add\_sr\_to\_subrules(T\_subrule\_def \* *R*);**int** no\_subrules();**void** phrase\_tree(AST \* *Tree*);

AST \* phrase\_tree();

**private:** **std**::vector < T\_subrule\_def \*> subrules\_;**int** no\_subrules\_;

AST \* phrase\_tree\_;

**246. subrules-phrase user-implementation directive.**

```

⟨subrules-phrase user-implementation directive 246⟩ ≡
  T_subrules_phrase::T_subrules_phrase()T_CTOR("subrules-phrase",
    T_Enum::T_T_subrules_phrase_, &dtor_T_subrules_phrase, false, false)
  {
    no_subrules_ = 0;
    phrase_tree_ = 0;
  }
  void T_subrules_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_subrules_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  int T_subrules_phrase::no_subrules()
  {
    return no_subrules_;
  }
  std::vector < T_subrule_def *> *T_subrules_phrase::subrules()
  {
    return &subrules_;
  }
  void T_subrules_phrase::destroy_subrules()
  {
    std::vector < T_subrule_def *> ::iterator i = subrules_.begin();
    std::vector < T_subrule_def *> ::iterator ie = subrules_.end();
    for (; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = *i;
      delete sym;
    }
  }
  void T_subrules_phrase::add_sr_to_subrules(T_subrule_def * R)
  {
    ++no_subrules_;
    subrules_.push_back(R);
  }

```

**247. sym-tbl-report-card.**

Enum: T\_T\_sym\_tbl\_report\_card\_

Class: T\_sym\_tbl\_report\_card

AB: N

AD: N



**248. sym-tbl-report-card user-declaration directive.**

⟨sym-tbl-report-card user-declaration directive 248⟩ ≡

```
public: enum status {
    okay, failure, fatal
};
enum action {
    not_fnd, fnd, inserted, aborted, unknown
};
T_sym_tbl_report_card();
T_sym_tbl_report_card
    ::status status_;
T_sym_tbl_report_card
    ::action action_;
table_entry * tbl_entry_;
CAbs_lr1_sym * err_entry_;
int pos_;
int key_len_;
```

**249. sym-tbl-report-card user-implementation directive.**

⟨sym-tbl-report-card user-implementation directive 249⟩ ≡

```
T_sym_tbl_report_card :: T_sym_tbl_report_card() T_CTOR("sym-tbl-report-card",
    T_Enum :: T_T_sym_tbl_report_card_, 0, false, false)
{
    status_ = okay;
    action_ = unknown;
    tbl_entry_ = 0;
    pos_ = 0;
    key_len_ = 0;
}
```

**250. syntax-code.**

Enum: T\_T\_syntax\_code\_

Class: T\_syntax\_code

AB: N

AD: N

**251. syntax-code user-declaration directive.**

⟨syntax-code user-declaration directive 251⟩ ≡

```
public: T_syntax_code(const char *Syntax_code);
void add_cweb_marker(AST * Cweb);
AST * cweb_marker();
std::string * syntax_code();
private: std::string syntax_code_;
AST * cweb_marker_;
```

**252. syntax-code user-implementation directive.**

```

⟨syntax-code user-implementation directive 252⟩ ≡
  T_syntax_code::T_syntax_code(const char *Syntax_code)T_CTOR("syntax-code",
    T_Enum::T_T_syntax_code_, 0, false, false)
  {
    syntax_code_ += Syntax_code;
    cweb_marker_ = 0;
  }
  std::string * T_syntax_code::syntax_code()
  {
    return &syntax_code_;
  }
  void T_syntax_code::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_syntax_code::cweb_marker()
  {
    return cweb_marker_;
  }

```

**253. table-entry.**

Enum: T\_table\_entry\_

Class: table\_entry

AB: N

AD: N

**254. table-entry user-declaration directive.**

```

⟨table-entry user-declaration directive 254⟩ ≡
public: enum entry_typ {
  terminal, rule, keyword, unknown, thread
};
enum defined_or_used_typ {
  defed, used
};
table_entry();
bool vacant_;
bool defined_;
bool used_;
entry_typ type_;
CAbs_lr1_sym * symbol_;
const char *key_;
int pos_;
int key_len_;

```

**255. table-entry user-implementation directive.**

```

⟨table-entry user-implementation directive 255⟩ ≡
  table_entry :: table_entry()T_CTOR("symbol-table", T_Enum :: T_table_entry_, 0, false, false)
  {
    vacant_ = true;
    defined_ = false;
    used_ = false;
    type_ = table_entry :: unknown;
    symbol_ = 0;
    key_ = 0;
    pos_ = -1;
    key_len_ = 0;
  }

```

**256. terminal-def.**

Enum: T\_T\_terminal\_def\_

Class: T\_terminal\_def

AB: N

AD: N

Watch out.

The individual syntax directed code directives are unique whereby each directive terminal contains their source code. Depending on their key within the “directives\_map\_”, u recast the abstract symbol into the specific directive.

For example:

The “SDC\_user\_declaration” key representing the “user-declaration” directive would cast the mapped symbol to *T\_user\_declaration*.

**257. terminal-def destructor directive.**

```

⟨terminal-def destructor directive 257⟩ ≡
  R_remove_directives_from_map();

```

**258. terminal-def user-declaration directive.**

```

⟨ terminal-def user-declaration directive 258 ⟩ ≡
public: enum classification_typ {
    not_classed, err, rc, lrk, t
};
T_terminal_def();
void remove_directives_from_map();
std::map < std::string, CAbs_lr1_sym *> *directives_map();
CAbs_lr1_sym * /* 0 - ok, or error */
add_directive_to_map(CAbs_lr1_sym * Directive, yacco2::Parser * P);
std::string * t_name();
void t_name(const char *Id);
std::string * classsym();
void classsym(const char *Id);
bool autodelete();
void autodelete(bool Ton);
bool autoabort();
void autoabort(bool Ton);
classification_typ classification();
void classification(classification_typ C);
void add_cweb_marker(AST * Cweb);
AST * cweb_marker();
void enum_id(int Id);
int enum_id();
private: std::string t_name_;
std::string class_sym_;
bool auto_delete_;
bool auto_abort_;
map < std::string, CAbs_lr1_sym *> directives_map_;
classification_typ classification_;
AST * cweb_marker_;
int enum_id_;

```

**259. terminal-def user-implementation directive.**

```

⟨terminal-def user-implementation directive 259⟩ ≡
  T_terminal_def :: T_terminal_def()T_CTOR("terminal-def", T_Enum :: T_T_terminal_def_,
    &dtor_T_terminal_def, false, false)
  {
    auto_delete_ = false;
    auto_abort_ = false;
    classification_ = T_terminal_def :: not_classed;
    cweb_marker_ = 0;
    enum_id_ = -1;
  }
  void T_terminal_def :: enum_id(int Id)
  {
    enum_id_ = Id;
  }
  int T_terminal_def :: enum_id()
  {
    return enum_id_;
  }
  void T_terminal_def :: add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_terminal_def :: cweb_marker()
  {
    return cweb_marker_;
  }
  map < std::string, CAbs_lr1_sym *> *T_terminal_def :: directives_map()
  {
    return &directives_map_;
  }
  void T_terminal_def :: remove_directives_from_map()
  {
    std::map < std::string, CAbs_lr1_sym *> ::iterator i = directives_map_.begin();
    std::map < std::string, CAbs_lr1_sym *> ::iterator ie = directives_map_.end();
    for (; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = i->second;
      delete sym;
    }
  }
  CAbs_lr1_sym * /* 0 - ok, or error */
  T_terminal_def :: add_directive_to_map(CAbs_lr1_sym * Directive, yacco2::Parser * P)
  {
    map < std::string, CAbs_lr1_sym *> ::iterator i;
    std::string key(Directive->id_);
    i = directives_map_.find(key);
    if (i ≡ directives_map_.end()) {
      directives_map_[key] = Directive;
      return 0;
    }
  }

```

```

    CAbs_lr1_sym * sym = new Err_duplicate_directive;
    sym->set_rc(*Directive, __FILE__, __LINE__);
    remove_directives_from_map();
    return sym;
}
std::string * T_terminal_def::t_name()
{
    return &t_name_;
}
void T_terminal_def::t_name(const char *Id)
{
    t_name_ += Id;
}
std::string * T_terminal_def::classsym()
{
    return &class_sym_;
}
void T_terminal_def::classsym(const char *Id)
{
    class_sym_ += Id;
}
bool T_terminal_def::autodelete()
{
    return auto_delete_;
}
void T_terminal_def::autodelete(bool Ton)
{
    auto_delete_ = Ton;
}
bool T_terminal_def::autoabort()
{
    return auto_abort_;
}
void T_terminal_def::autoabort(bool Ton)
{
    auto_abort_ = Ton;
}
T_terminal_def::classification_typ T_terminal_def::classification()
{
    return classification_;
}
void T_terminal_def::classification ( T_terminal_def::classification_typ C )
{
    classification_ = C;
}

```

**260. terminals-phrase.**

Enum: T\_T\_terminals\_phrase\_

Class: T\_terminals\_phrase

AB: N

AD: N

**261. terminals-phrase destructor directive.**

⟨terminals-phrase destructor directive 261⟩ ≡

```

R-destroy_alphabet();
delete R-filename_id_;
delete R-namespace_id_;
delete R-terminals_refs_code_;
delete R-terminals_sufx_code_;

```

**262. terminals-phrase user-declaration directive.**

⟨terminals-phrase user-declaration directive 262⟩ ≡

```

public: T_terminals_phrase();
    T_identifier *filename_id();
    void filename_id(T_identifier *Id);
    T_identifier *namespace_id();
    void namespace_id(T_identifier *Id);
    void destroy_alphabet();

    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *alphabet();
    CAbs_lr1_sym * add_t_to_alphabet(T_terminal_def *T, yacco2::Parser *P);
    std::vector < T_terminal_def *> *crt_order();

    T_syntax_code *terminals_refs_code();
    void terminals_refs_code(T_syntax_code *Code);
    T_syntax_code *terminals_sufx_code();
    void terminals_sufx_code(T_syntax_code *Code);
    void phrase_tree(AST * Tree);

    AST * phrase_tree();

    void add_cweb_marker(AST * Cweb);

    AST * cweb_marker();

private: T_identifier *filename_id_;
    T_identifier *namespace_id_;

    std::map < std::string, NS_yacco2_terminals::T_terminal_def *> alphabet_;
    std::vector < T_terminal_def *> crt_order_;

    T_syntax_code *terminals_refs_code_;
    T_syntax_code *terminals_sufx_code_;

    AST * phrase_tree_;
    AST * cweb_marker_;

```

**263. terminals-phrase user-implementation directive.**

```

⟨ terminals-phrase user-implementation directive 263 ⟩ ≡
  T_terminals_phrase::T_terminals_phrase()T_CTOR("terminals-phrase",
    T_Enum::T_T_terminals_phrase_, &dtor_T_terminals_phrase, false, false)
  {
    terminals_refs_code_ = 0;
    terminals_sufx_code_ = 0;
    phrase_tree_ = 0;
    cweb_marker_ = 0;
  }
  void T_terminals_phrase::add_cweb_marker(AST * Cweb)
  {
    cweb_marker_ = Cweb;
  }
  AST * T_terminals_phrase::cweb_marker()
  {
    return cweb_marker_;
  }
  void T_terminals_phrase::phrase_tree(AST * Tree)
  {
    phrase_tree_ = Tree;
  }
  AST * T_terminals_phrase::phrase_tree()
  {
    return phrase_tree_;
  }
  T_identifier *T_terminals_phrase::filename_id()
  {
    return filename_id_;
  }
  void T_terminals_phrase::filename_id(T_identifier *Id)
  {
    filename_id_ = Id;
  }
  T_identifier *T_terminals_phrase::namespace_id()
  {
    return namespace_id_;
  }
  void T_terminals_phrase::namespace_id(T_identifier *Id)
  {
    namespace_id_ = Id;
  }
  std::map < std::string, NS_yacco2_terminals::T_terminal_def *> *T_terminals_phrase::alphabet()
  {
    return &alphabet_;
  }
  std::vector < T_terminal_def *> *T_terminals_phrase::crt_order()
  {
    return &crt_order_;
  }

```



```

}
void T_terminals_phrase::destroy_alphabet()
{
    std::vector < T_terminal_def * > ::iterator i = crt_order_.begin();
    std::vector < T_terminal_def * > ::iterator ie = crt_order_.end();
    for ( ; i ≠ ie; ++i) {
        CAbs_lr1_sym * sym = *i;
        delete sym;
    }
    alphabet_.clear();
}
CAbs_lr1_sym * T_terminals_phrase::add_t_to_alphabet(T_terminal_def * T, yacco2::Parser * P)
{
    stringkey(T->t_name()-c_str());
    std::map < std::string, NS_yacco2_terminals::T_terminal_def * > ::iterator i = alphabet_.find(key);
    if (i ≠ alphabet_.end()) {
        CAbs_lr1_sym * sym = new Err_dup_entry_in_alphabet;
        sym->set_rc(*T, __FILE__, __LINE__);
        return sym;
    }
    alphabet_[key] = T;
    crt_order_.push_back(T);
    return 0;
}
T_syntax_code * T_terminals_phrase::terminals_refs_code()
{
    return terminals_refs_code_;
}
void T_terminals_phrase::terminals_refs_code(T_syntax_code * Code)
{
    terminals_refs_code_ = Code;
}
T_syntax_code * T_terminals_phrase::terminals_sufx_code()
{
    return terminals_sufx_code_;
}
void T_terminals_phrase::terminals_sufx_code(T_syntax_code * Code)
{
    terminals_sufx_code_ = Code;
}

```

**264. th-in-stbl.**

Enum: T\_th\_in\_stbl

Class: th\_in\_stbl

AB: N

AD: N

**265. th-in-stbl user-declaration directive.**

```

⟨ th-in-stbl user-declaration directive 265 ⟩ ≡
public: th_in_stbl(thread_attributes * Fnd_th_in_stbl);
         thread_attributes * thread_in_stbl();
         void stbl_idx(yacco2::UINT Idx);
         int stbl_idx();
private: thread_attributes * th_in_stbl_;
         int stbl_idx_;

```

**266. th-in-stbl user-implementation directive.**

```

⟨ th-in-stbl user-implementation directive 266 ⟩ ≡
  th_in_stbl::th_in_stbl(thread_attributes * Fnd_th_in_stbl)T_CTOR("th-in-stbl", T_Enum::T_th_in_stbl, 0,
    false, false)
  {
    th_in_stbl_ = Fnd_th_in_stbl;
    stbl_idx_ = -1;
  }
  thread_attributes * th_in_stbl::thread_in_stbl()
  {
    return th_in_stbl_;
  }
  void th_in_stbl::stbl_idx(yacco2::UINT Idx)
  {
    stbl_idx_ = Idx;
  }
  int th_in_stbl::stbl_idx()
  {
    return stbl_idx_;
  }

```

**267. thread-attributes.**Enum: *T\_thread\_attributes\_*Class: *thread\_attributes*

AB: N

AD: N

**268. thread-attributes user-declaration directive.**

⟨thread-attributes user-declaration directive 268⟩ ≡

```

public: thread_attributes(const char *Fully_qualified_th_name); thread_attributes (const char
    *Fully_qualified_th_name, char Transitive, T_c_string * Grammar_name,
    T_c_string * Name_space_name, T_c_string * Thread_name, char Monolithic, T_c_string * File_name,
    std::vector < int > &List_of_Ts, std::vector < thread_attributes * > &List_of_transitive_thds,
    T_c_string*Fsm_comments );
std::string fully_qualified_th_name_;
char transitive_;
T_c_string * grammar_file_name_;
T_c_string * name_space_name_;
T_c_string * thread_name_;
char monolithic_;
T_c_string * file_name_; std::vector < int > list_of_Ts_; std::set < int > fs_;
std::vector < thread_attributes * > list_of_transitive_threads_;
int th_enum_;
T_c_string * fsm_comments_;
AST * called_thread_graph_;

```

**269. thread-attributes user-implementation directive.**

```

⟨thread-attributes user-implementation directive 269⟩ ≡
  thread_attributes :: thread_attributes(const char *Fully_qualified_th_name)T_CTOR("thread-attributes",
    T_Enum :: T_thread_attributes_, 0, false, false)
  {
    fully_qualified_th_name_ += Fully_qualified_th_name;
    transitive_ = '␣';
    grammar_file_name_ = 0;
    name_space_name_ = 0;
    thread_name_ = 0;
    monolithic_ = '␣';
    file_name_ = 0;
    th_enum_ = -1;
    fsm_comments_ = 0;
    called_thread_graph_ = 0;
  }
  thread_attributes :: thread_attributes (const char *Fully_qualified_th_name, char Transitive,
    T_c_string * Grammar_name, T_c_string * Name_space_name, T_c_string * Thread_name,
    char Monolithic, T_c_string * File_name, std::vector < int > &List_of_Ts,
    std::vector < thread_attributes * > &List_of_transitive_thds, T_c_string * Fsm_comments
    ) T_CTOR("thread-attributes", T_Enum :: T_thread_attributes_, 0, false, false){
    fully_qualified_th_name_ += Fully_qualified_th_name;
    transitive_ = Transitive;
    grammar_file_name_ = Grammar_name;
    name_space_name_ = Name_space_name;
    thread_name_ = Thread_name;
    monolithic_ = Monolithic;
    file_name_ = File_name; if (¬List_of_Ts.empty()) { copy(List_of_Ts.begin(), List_of_Ts.end(),
      back_inserter(list_of_Ts_)); std::vector < int > ::iterator i = List_of_Ts.begin(); std::vector <
      int > ::iterator ie = List_of_Ts.end();
    for ( ; i ≠ ie; ++i) {
      fs_.insert(*i);
    }
  }
  if (¬List_of_transitive_thds.empty()) copy(List_of_transitive_thds.begin(),
    List_of_transitive_thds.end(), back_inserter(list_of_transitive_threads_));
  th_enum_ = -1;
  fsm_comments_ = Fsm_comments;
  called_thread_graph_ = 0; }

```

**270. thread-name.**

Enum: T\_T\_thread\_name\_

Class: T\_thread\_name

AB: N

AD: N

**271. transitive.**

Enum: T\_T\_transitive\_

Class: T\_transitive

AB: N

AD: N

**272. tth-in-stbl.**

Enum: T\_tth\_in\_stbl\_

Class: tth\_in\_stbl

AB: N

AD: N

**273. tth-in-stbl user-declaration directive.**

⟨tth-in-stbl user-declaration directive 273⟩ ≡

```
public: tth_in_stbl(T_attributes * Fnd_T_in_stbl, CAbs_lr1_sym * Rc, yacco2::Parser * P);
    T_attributes * t_in_stbl();
    void stbl_idx(yacco2::UINT Idx);
    int stbl_idx();
private: T_attributes * t_in_stbl_;
    int stbl_idx_;
```

**274. tth-in-stbl user-implementation directive.**

⟨tth-in-stbl user-implementation directive 274⟩ ≡

```
tth_in_stbl::tth_in_stbl(T_attributes * Fnd_T_in_stbl, CAbs_lr1_sym * Rc,
    yacco2::Parser * P)T_CTOR("tth-in-stbl", T_Enum::T_tth_in_stbl_, 0, false, false)
{
    t_in_stbl_ = Fnd_T_in_stbl;
    set_rc(*Rc, __FILE__, __LINE__); /* set its source co-ordinates */
    stbl_idx_ = -1;
}
T_attributes * tth_in_stbl::t_in_stbl()
{
    return t_in_stbl_;
}
void tth_in_stbl::stbl_idx(yacco2::UINT Idx)
{
    stbl_idx_ = Idx;
}
int tth_in_stbl::stbl_idx()
{
    return stbl_idx_;
}
```

**275. unquoted-string.**

Enum: T\_T\_unquoted\_string\_

Class: T\_unquoted\_string

AB: N

AD: N

Just as described. Gather all the characters until an end-of-line met. Used in the command line. Watch yourself as this is a gobbler: marginal utility.

**276. unquoted-string user-declaration directive.**

⟨unquoted-string user-declaration directive 276⟩ ≡

```
public: T_unquoted_string(const std::string& String);
    std::string * unquoted_string();
private: std::string unquoted_string_;
```

**277. unquoted-string user-implementation directive.**

```

⟨unquoted-string user-implementation directive 277⟩ ≡
  T_unquoted_string :: T_unquoted_string(const std::string&String)T_CTOR("unquoted-string",
    T_Enum :: T_T_unquoted_string_, 0, false, false)
  {
    unquoted_string_ = String;
  }
  std::string * T_unquoted_string :: unquoted_string()
  {
    return &unquoted_string_;
  }

```

**278. ws.**

Enum: T\_T\_ws\_

Class: T\_ws

AB: N

AD: N

Just white space of spaces, vertical and horizontal tabs, and form feeds. This is a vacuum cleaner of inter-word separators.

**279. ws user-declaration directive.**

```

⟨ws user-declaration directive 279⟩ ≡
public: T_ws(const std::string&White_space_data);
  T_ws();
  std::string * ws_data();
private: std::string ws_data_;

```

**280. ws user-implementation directive.**

```

⟨ws user-implementation directive 280⟩ ≡
  T_ws :: T_ws(const std::string&Ws_data)T_CTOR("ws", T_Enum :: T_T_ws_, 0, false, false)
  {
    ws_data_ += Ws_data;
  }
  T_ws :: T_ws()T_CTOR("ws", T_Enum :: T_T_ws_, 0, false, false)
  {}
  std::string * T_ws :: ws_data()
  {
    return &ws_data_;
  }
  T_ws ws_;
  yacco2 :: CAbs_lr1_sym * NS_yacco2_terminals :: PTR_ws_ = &ws_;

```

**281. xc-str.**

Enum: T\_T\_xc\_str\_

Class: T\_xc\_str

AB: N

AD: N

Returned from *xc-str* grammar recognizing a c++ string without the escape sequence check. This allows one to input at the command level a string containing backslash etc in single quantities. Example, "nogoodb". This does not end a line nor ring your bell. The surrounding double quotes are not added to the content. It's just a slight variation on "c-string" using a different grammar.

**282. xc-str user-declaration directive.**

```

⟨xc-str user-declaration directive 282⟩ ≡
public: T_xc_str(const std::string& C_string);
       std::string * c_string();
private: std::string c_string_;

```

**283. xc-str user-implementation directive.**

```

⟨xc-str user-implementation directive 283⟩ ≡
T_xc_str::T_xc_str(const std::string& C_string)T_CTOR("xc-str", T_Enum::T_T_xc_str_, 0, false, false)
{
    c_string_ = C_string;
}
std::string * T_xc_str::c_string()
{
    return &c_string_;
}

```

**284. Index.**

- \_\_FILE\_\_**: 108, 153, 164, 192, 204, 217, 234, 238, 242, 259, 263, 274.  
**\_\_LINE\_\_**: 108, 153, 164, 192, 204, 217, 234, 238, 242, 259, 263, 274.  
**ABORT\_STATUS**: 117.  
*aborted*: 248.  
**action**: 248.  
*action\_*: 248, 249.  
*add\_closure\_rule\_making\_up\_first\_set*: 226, 227.  
*add\_cweb\_marker*: 13, 14, 54, 55, 61, 62, 104, 105, 152, 153, 167, 168, 191, 192, 207, 208, 216, 217, 226, 227, 233, 234, 237, 238, 241, 242, 251, 252, 258, 259, 262, 263.  
*add\_directive\_to\_lhs*: 233, 234.  
*add\_directive\_to\_map*: 163, 164, 258, 259.  
*add\_directive\_to\_mntr*: 203, 204.  
*add\_directive\_to\_subrule*: 241, 242.  
*add\_elem\_to\_subrule\_vector*: 241, 242.  
*add\_R\_into\_xref*: 229, 230.  
*add\_r\_to\_alphabet*: 237, 238.  
*add\_rule\_adding\_T\_in\_first\_set*: 226, 227.  
*add\_sr\_to\_subrules*: 245, 246.  
*add\_T\_into\_xref*: 107, 108.  
*add\_t\_to\_alphabet*: 152, 153, 191, 192, 216, 217, 262, 263.  
*add\_to\_called\_thread\_first\_set*: 226, 227.  
*add\_to\_first\_set*: 226, 227.  
*alphabet*: 152, 153, 191, 192, 216, 217, 262, 263.  
*alphabet\_*: 152, 153, 191, 192, 216, 217, 262, 263.  
*angled\_string*: 109, 110, 111.  
*Angled\_string*: 110, 111.  
*angled\_string\_*: 109, 110, 111.  
*append*: 55.  
**AST**: 13, 14, 54, 55, 61, 62, 104, 105, 117, 118, 119, 127, 128, 136, 137, 148, 149, 152, 153, 163, 164, 167, 168, 191, 192, 196, 197, 203, 204, 207, 208, 216, 217, 219, 220, 222, 223, 226, 227, 233, 234, 237, 238, 241, 242, 245, 246, 251, 252, 258, 259, 262, 263, 268.  
*Ast*: 118, 119.  
*ast*: 117, 118, 119, 136, 137.  
*ast\_*: 118, 119, 136, 137.  
*ast\_delete*: 117.  
*auto\_abort\_*: 226, 227, 258, 259.  
*auto\_delete\_*: 226, 227, 258, 259.  
*autoabort*: 226, 227, 258, 259.  
*autodelete*: 226, 227, 258, 259.  
*back\_inserter*: 269.  
*bad\_char\_set*: 112.  
*Basic\_char*: 114, 115.  
*basic\_char*: 113, 114, 115.  
*basic\_char\_*: 113, 114, 115.  
*begin*: 55, 153, 164, 192, 204, 217, 227, 234, 238, 242, 246, 259, 263, 269.  
*bld\_its\_tree*: 226, 227, 241, 242.  
**C**: 258, 259.  
*c\_comment*: 129.  
*c\_literal*: 120, 121, 122.  
*C\_literal*: 121, 122.  
*c\_literal\_*: 120, 121, 122.  
*c\_str*: 153, 192, 217, 238, 263.  
*C\_string*: 124, 125, 282, 283.  
*c\_string*: 123, 124, 125, 282, 283.  
*c\_string\_*: 123, 124, 125, 282, 283.  
*CAbs\_lr1\_sym*: 3, 107, 108, 114, 115, 146, 152, 153, 158, 159, 163, 164, 178, 179, 185, 191, 192, 203, 204, 212, 213, 216, 217, 233, 234, 237, 238, 241, 242, 246, 248, 254, 258, 259, 262, 263, 273, 274, 280.  
*called\_thread\_first\_set*: 226, 227.  
*called\_thread\_first\_set\_*: 226, 227.  
*called\_thread\_graph\_*: 268, 269.  
*called\_thread\_name*: 127, 128.  
*called\_thread\_name\_*: 127, 128.  
*characters*: 210.  
*class\_sym\_*: 258, 259.  
*classification*: 258, 259.  
*classification\_*: 258, 259.  
**classification\_typ**: 258, 259.  
*classsym*: 258, 259.  
*clear*: 153, 192, 217, 227, 238, 263.  
*closure\_rules\_making\_up\_first\_set*: 226, 227.  
*closure\_rules\_making\_up\_first\_set\_*: 226, 227.  
**Code**: 191, 192, 262, 263.  
*comment*: 167, 168.  
*comment\_*: 166, 167, 168.  
*Comment\_data*: 130, 131, 133, 134.  
*comment\_data*: 130, 131, 133, 134.  
*comment\_data\_*: 130, 131, 133, 134.  
*copy*: 269.  
*crt\_order*: 152, 153, 191, 192, 216, 217, 237, 238, 262, 263.  
*crt\_order\_*: 152, 153, 191, 192, 216, 217, 237, 238, 262, 263.  
*crt\_tree\_of\_lison*: 227, 242.  
*ctor*: 154.  
**CTOR**: 72.  
*Cweb*: 13, 14, 54, 55, 61, 62, 104, 105, 152, 153, 167, 168, 191, 192, 207, 208, 216, 217, 226, 227, 233, 234, 237, 238, 241, 242, 251, 252, 258, 259, 262, 263.  
*cweb\_la\_srce\_expr*: 54, 55.



- cweb\_la\_srce\_expr\_*: 54, 55.  
*cweb\_marker*: 13, 14, 54, 55, 61, 62, 104, 105, 152, 153, 167, 168, 191, 192, 207, 208, 216, 217, 226, 227, 233, 234, 237, 238, 241, 242, 251, 252, 258, 259, 262, 263.  
*cweb\_marker\_*: 13, 14, 54, 55, 61, 62, 104, 105, 152, 153, 167, 168, 191, 192, 207, 208, 216, 217, 226, 227, 233, 234, 237, 238, 241, 242, 251, 252, 258, 259, 262, 263.  
*date*: 167, 168.  
*date\_*: 166, 167, 168.  
*debug*: 167, 168.  
*debug\_*: 166, 167, 168.  
*defed*: 254.  
*defined\_*: 254, 255.  
**defined\_or\_used\_typ**: 254.  
*derive\_t*: 226, 227, 241, 242.  
*derive\_t\_*: 226, 227, 241, 242.  
*destroy\_alphabet*: 151, 152, 153, 190, 191, 192, 215, 216, 217, 261, 262, 263.  
*destroy\_rules\_alphabet*: 236, 237, 238.  
*destroy\_subrules*: 244, 245, 246.  
*Directive*: 163, 164, 203, 204, 233, 234, 241, 242, 258, 259.  
*directives\_map*: 163, 164, 258, 259.  
*directives\_map\_*: 163, 164, 258, 259.  
*dtor\_rule\_def*: 227.  
*dtor\_T\_basic\_char*: 115.  
*dtor\_T\_block*: 119.  
*dtor\_T\_enum\_phrase*: 105.  
*dtor\_T\_enumeration*: 11.  
*dtor\_T\_error\_symbols*: 25.  
*dtor\_T\_error\_symbols\_phrase*: 153.  
*dtor\_T\_fsm*: 33.  
*dtor\_T\_fsm\_class\_phrase*: 164.  
*dtor\_T\_fsm\_phrase*: 168.  
*dtor\_T\_lr1\_constant\_symbols*: 46.  
*dtor\_T\_lr1\_k\_phrase*: 192.  
*dtor\_T\_parallel\_monitor\_phrase*: 204.  
*dtor\_T\_parallel\_parser*: 59.  
*dtor\_T\_parallel\_parser\_phrase*: 208.  
*dtor\_T\_raw\_characters*: 66.  
*dtor\_T\_rc\_phrase*: 217.  
*dtor\_T\_rule\_lhs\_phrase*: 234.  
*dtor\_T\_rules*: 70.  
*dtor\_T\_rules\_phrase*: 238.  
*dtor\_T\_subrule\_def*: 242.  
*dtor\_T\_subrules\_phrase*: 246.  
*dtor\_T\_terminal\_def*: 259.  
*dtor\_T\_terminals*: 75.  
*dtor\_T\_terminals\_phrase*: 263.  
*ei*: 242.  
*eie*: 242.  
*Elem*: 241, 242.  
*elem\_def*: 242.  
*element\_list*: 242.  
*element\_lvl*: 242.  
*element\_pos*: 127, 128, 148, 149, 196, 197, 219, 220, 222, 223.  
*element\_pos\_*: 127, 128, 148, 149, 196, 197, 219, 220, 222, 223.  
*elemtt*: 242.  
*empty*: 269.  
*end*: 55, 153, 164, 192, 204, 217, 227, 234, 238, 242, 246, 259, 263, 269.  
**entry\_typ**: 254.  
*Enum*: 100, 101, 104, 105.  
*enum\_id*: 226, 227, 258, 259.  
*enum\_id\_*: 226, 227, 258, 259.  
*enum\_phrase*: 10, 11.  
*enum\_phrase\_*: 9, 10, 11.  
*Enumerate*: 127, 128, 148, 149, 196, 197, 219, 220, 222, 223, 226, 227, 241, 242.  
*enumerated\_id\_*: 242.  
*eol*: 144.  
*eol\_*: 146.  
*eos*: 242.  
*Epsilon*: 241, 242.  
*epsilon*: 226, 227, 241, 242.  
*epsilon\_*: 226, 227, 241, 242.  
*err*: 258.  
*Err\_dup\_entry\_in\_alphabet*: 153, 192, 217, 238, 263.  
*Err\_duplicate\_directive*: 164, 204, 234, 242, 259.  
*err\_entry\_*: 248.  
*Error*: 158, 159.  
*error\_sym*: 158, 159.  
*error\_sym\_*: 157, 158, 159.  
*error\_symbols\_phrase*: 24, 25.  
*error\_symbols\_phrase\_*: 23, 24, 25.  
*Esc\_data*: 155, 156.  
*esc\_data*: 155, 156.  
*esc\_data\_*: 155, 156.  
*esc\_seq*: 154.  
*failure*: 248.  
*false*: 11, 14, 18, 21, 25, 28, 33, 46, 51, 55, 59, 62, 66, 70, 75, 80, 83, 86, 89, 92, 95, 101, 105, 108, 111, 115, 119, 122, 125, 128, 131, 134, 137, 146, 149, 153, 156, 159, 164, 168, 173, 176, 179, 182, 185, 192, 197, 204, 208, 213, 217, 220, 223, 227, 230, 234, 238, 242, 246, 249, 252, 255, 259, 263, 266, 269, 274, 277, 280, 283.  
*fatal*: 248.  
*file\_name*: 158, 159.  
*File\_name*: 159, 268, 269.

- File\_name\_*: 158.  
*file\_name\_*: 158, 159, 268, 269.  
*filename\_id*: [104](#), [105](#), [152](#), [153](#), [167](#), [168](#), [191](#),  
[192](#), [216](#), [217](#), [262](#), [263](#).  
*filename\_id\_*: 103, [104](#), 105, 151, [152](#), 153, 166, [167](#),  
168, 190, [191](#), 192, 215, [216](#), 217, 261, [262](#), 263.  
*filename\_no\_ext\_*: 167.  
*find*: 153, 164, 192, 204, 217, 227, 234, 238,  
242, 259, 263.  
*first\_set*: 226, [227](#).  
*first\_set\_*: 226, [227](#).  
*fnd*: 248.  
*Fnd\_kw\_in\_stbl*: 178, 179.  
*Fnd\_T\_in\_stbl*: 107, 108, 273, 274.  
*Fnd\_th\_in\_stbl*: 265, 266.  
*fs\_*: 268, 269.  
*fsm\_class\_phrase*: [167](#), [168](#).  
*fsm\_class\_phrase\_*: 166, [167](#), 168.  
*Fsm\_comments*: [268](#), [269](#).  
*fsm\_comments\_*: 268, 269.  
*fsm\_id*: [167](#), [168](#).  
*fsm\_id\_*: 166, 167, 168.  
*fsm\_phrase*: [32](#), [33](#).  
*fsm\_phrase\_*: 31, [32](#), 33.  
*Fully\_qualified\_T\_name*: [100](#), [101](#).  
*fully\_qualified\_T\_name\_*: 100, 101.  
*Fully\_qualified\_th\_name*: [268](#), [269](#).  
*fully\_qualified\_th\_name\_*: 268, 269.  
*grammar\_file\_name\_*: 268, 269.  
*Grammar\_name*: 268, 269.  
*grammar\_s\_enumerate*: [127](#), [128](#), [148](#), [149](#), [196](#),  
[197](#), [219](#), [220](#), [222](#), [223](#), [226](#), [227](#), [241](#), [242](#).  
*grammar\_s\_enumerate\_*: 127, 128, 148, 149, 196,  
197, 219, 220, 222, 223, 226, 227, 241, 242.  
*id*: [242](#).  
*Id*: [61](#), [62](#), [104](#), [105](#), [152](#), [153](#), [163](#), [164](#), [167](#), [168](#),  
[191](#), [192](#), [216](#), [217](#), [226](#), [227](#), [258](#), [259](#), [262](#), [263](#).  
*id\_*: [61](#), 62.  
*id\_*: 164, 204, 234, 242, 259.  
*Identifier*: 172, 173.  
*identifier*: [61](#), [62](#), [163](#), [164](#), 171, 172, 173.  
*identifier\_*: 162, [163](#), 164, 172, 173.  
*Idx*: 107, 108, 178, 179, 229, 230, 265, 266,  
273, 274.  
*ie*: 153, 164, 192, 204, 217, 227, 234, 238, 242,  
246, 259, 263, 269.  
*insert*: 55, 227, 269.  
*inserted*: 248.  
*integer\_no*: 174.  
*iterator*: 153, 164, 192, 204, 217, 227, 234, 238,  
242, 246, 259, 263, 269.  
*its\_grammar\_s\_pos*: [241](#), [242](#).  
*its\_grammar\_s\_pos\_*: 239, [241](#), 242.  
*Its\_rule\_def*: [241](#), [242](#).  
*its\_rule\_def*: [127](#), [128](#), [148](#), [149](#), [196](#), [197](#), [219](#),  
[220](#), [222](#), [223](#), [241](#), [242](#).  
*its\_rule\_def\_*: [241](#), 242.  
*Its\_subrule*: 127, 128, 148, 149, 196, 197, 219,  
220, 222, 223.  
*its\_subrule\_def*: [127](#), [128](#), [148](#), [149](#), [196](#), [197](#),  
[219](#), [220](#), [222](#), [223](#).  
*its\_subrule\_def\_*: 127, 128, 148, 149, 196, 197,  
219, 220, 222, 223.  
*its\_t\_def*: 219, 220.  
*its\_tree\_*: 226, 227, 241, 242.  
*join\_sts*: 227, 242.  
KCHARP: 172, 173.  
*Kdefs*: [104](#), [105](#).  
*kdefs*: [104](#), [105](#).  
*kdefs\_*: 103, [104](#), 105.  
*Key*: [226](#), [227](#).  
*key*: 153, 164, 192, 204, 217, 234, 238, 242,  
259, 263.  
*key\_*: [254](#), 255.  
*key\_len\_*: [248](#), 249, [254](#), 255.  
*keyword*: 254.  
*keyword\_in\_stbl*: 178, 179.  
*kw\_in\_stbl*: 178, [179](#).  
*kw\_in\_stbl\_*: 178, 179.  
*La\_bndry*: 207, 208.  
*la\_bndry*: [207](#), [208](#).  
*la\_bndry\_*: 206, 207, 208.  
*la\_first\_set*: 54, 55.  
*la\_first\_set\_*: 54, 55.  
*la\_supplier*: [54](#), [55](#).  
*la\_supplier\_*: 54, 55.  
*la\_tok\_can*: 181, 182.  
*la\_tok\_can\_*: 181, 182.  
*Lhs*: 226, 227.  
*lhs\_directives\_map*: 233, 234.  
*lhs\_directives\_map\_*: 233, 234.  
*lhs\_use\_cnt*: [226](#), [227](#).  
*lhs\_use\_cnt\_*: [226](#), 227.  
*lint\_*: 185.  
*list*: 107, 108, 229, 230.  
*List\_of\_transitive\_thds*: 268, 269.  
*list\_of\_transitive\_threads\_*: 268, 269.  
*List\_of\_Ts*: 268, 269.  
*list\_of\_Ts\_*: 268, 269.  
*lrk*: 258.  
*lrk\_sufx\_code*: [191](#), [192](#).  
*lrk\_sufx\_code\_*: 190, [191](#), 192.  
*lr1\_k\_phrase*: [45](#), [46](#).  
*lr1\_k\_phrase\_*: 44, [45](#), 46.

- map*: [152](#), [153](#), [163](#), [164](#), [191](#), [192](#), [203](#), [204](#), [216](#),  
[217](#), [226](#), [227](#), [233](#), [234](#), [237](#), [238](#), [241](#), [242](#),  
[258](#), [259](#), [262](#), [263](#).  
*Mntr*: [226](#), [227](#).  
*mnr\_directives\_map*: [203](#), [204](#).  
*mnr\_directives\_map\_*: [203](#), [204](#).  
*Monolithic*: [268](#), [269](#).  
*monolithic\_*: [268](#), [269](#).  
*Name\_space\_name*: [268](#), [269](#).  
*name\_space\_name\_*: [268](#), [269](#).  
*namespace\_id*: [104](#), [105](#), [152](#), [153](#), [167](#), [168](#), [191](#),  
[192](#), [216](#), [217](#), [262](#), [263](#).  
*namespace\_id\_*: [103](#), [104](#), [105](#), [151](#), [152](#), [153](#),  
[166](#), [167](#), [168](#), [190](#), [191](#), [192](#), [215](#), [216](#), [217](#),  
[261](#), [262](#), [263](#).  
*no*: [175](#), [176](#).  
*NO*: [227](#).  
*no\_*: [175](#), [176](#).  
*no\_of\_elems*: [241](#), [242](#).  
*no\_of\_elems\_*: [241](#), [242](#).  
*no\_subrules*: [245](#), [246](#).  
*no\_subrules\_*: [245](#), [246](#).  
*Node*: [136](#), [137](#).  
*not\_classed*: [258](#), [259](#).  
*not\_fnd*: [248](#).  
*ns*: [127](#), [128](#).  
*Ns*: [127](#), [128](#).  
*ns\_*: [127](#), [128](#).  
**NS\_yacco2\_err\_symbols**: [3](#).  
**NS\_yacco2\_terminals**: [146](#), [152](#), [153](#), [185](#), [191](#),  
[192](#), [216](#), [217](#), [242](#), [262](#), [263](#), [280](#).  
*okay*: [248](#), [249](#).  
*parallel\_mntr*: [226](#), [227](#).  
*parallel\_mntr\_*: [225](#), [226](#), [227](#).  
*parallel\_parser\_phrase*: [58](#), [59](#).  
*parallel\_parser\_phrase\_*: [57](#), [58](#), [59](#).  
*Parser*: [107](#), [108](#), [152](#), [153](#), [163](#), [164](#), [191](#), [192](#),  
[203](#), [204](#), [216](#), [217](#), [233](#), [234](#), [237](#), [238](#), [241](#),  
[242](#), [258](#), [259](#), [262](#), [263](#), [273](#), [274](#).  
*pass3*: [157](#).  
*Phrase*: [10](#), [11](#), [24](#), [25](#), [32](#), [33](#), [45](#), [46](#), [58](#), [59](#),  
[65](#), [66](#), [69](#), [70](#), [74](#), [75](#).  
*phrase\_tree*: [104](#), [105](#), [152](#), [153](#), [163](#), [164](#), [167](#), [168](#),  
[191](#), [192](#), [203](#), [204](#), [207](#), [208](#), [216](#), [217](#), [233](#),  
[234](#), [237](#), [238](#), [245](#), [246](#), [262](#), [263](#).  
*phrase\_tree\_*: [104](#), [105](#), [152](#), [153](#), [163](#), [164](#), [167](#),  
[168](#), [191](#), [192](#), [203](#), [204](#), [207](#), [208](#), [216](#), [217](#), [233](#),  
[234](#), [237](#), [238](#), [245](#), [246](#), [262](#), [263](#).  
*Pos*: [127](#), [128](#), [148](#), [149](#), [196](#), [197](#), [219](#), [220](#),  
[222](#), [223](#), [241](#), [242](#).  
*pos\_*: [248](#), [249](#), [254](#), [255](#).  
*PP\_funct*: [207](#), [208](#).  
*pp\_funct*: [207](#), [208](#).  
*pp\_funct\_*: [206](#), [207](#), [208](#).  
*PTR\_eol\_*: [3](#), [146](#).  
*PTR\_lint\_*: [3](#), [185](#).  
*PTR\_ws\_*: [3](#), [280](#).  
*push\_back*: [108](#), [153](#), [192](#), [217](#), [230](#), [238](#), [242](#),  
[246](#), [263](#).  
*R*: [229](#), [230](#), [237](#), [238](#).  
*r\_def*: [223](#), [229](#), [230](#).  
*r\_def\_*: [229](#), [230](#).  
*raw*: [210](#).  
*Raw\_char*: [212](#), [213](#).  
*raw\_char*: [211](#), [212](#), [213](#).  
*raw\_char\_*: [211](#), [212](#), [213](#).  
*rc*: [258](#).  
*Rc*: [107](#), [108](#), [273](#), [274](#).  
*rc\_phrase*: [65](#), [66](#).  
*rc\_phrase\_*: [64](#), [65](#), [66](#).  
*recursive*: [226](#), [227](#).  
*Recursive*: [226](#), [227](#).  
*recursive\_*: [226](#), [227](#).  
**referred\_rule**: [3](#), [222](#), [223](#), [229](#), [230](#), [242](#).  
**referred\_T**: [3](#), [107](#), [108](#), [219](#), [220](#), [242](#).  
*remove\_directives\_from\_map*: [162](#), [163](#), [164](#), [257](#),  
[258](#), [259](#).  
*remove\_lhs\_directives*: [232](#), [233](#), [234](#).  
*remove\_mntr\_directives*: [202](#), [203](#), [204](#).  
*remove\_subrule\_directives*: [240](#), [241](#), [242](#).  
*remove\_subrule\_elems\_from\_vector*: [240](#), [241](#), [242](#).  
*rhs\_use\_cnt*: [226](#), [227](#).  
*rhs\_use\_cnt\_*: [226](#), [227](#).  
*rrule*: [242](#).  
*rt*: [242](#).  
*rule*: [254](#).  
*Rule*: [226](#), [227](#).  
*rule\_adding\_T\_in\_first\_set*: [226](#), [227](#).  
*Rule\_def*: [229](#), [230](#).  
**rule\_def**: [3](#), [127](#), [128](#), [148](#), [149](#), [196](#), [197](#), [219](#), [220](#),  
[222](#), [223](#), [226](#), [227](#), [229](#), [230](#), [237](#), [238](#), [241](#), [242](#).  
*Rule\_enumerate*: [226](#), [227](#).  
*Rule\_in\_stbl*: [222](#), [223](#).  
**rule\_in\_stbl**: [3](#), [222](#), [223](#), [226](#), [227](#), [229](#), [230](#).  
*rule\_in\_stbl\_*: [222](#), [223](#).  
*rule\_lhs*: [226](#), [227](#).  
*rule\_lhs\_*: [225](#), [226](#), [227](#).  
*rule\_name*: [226](#), [227](#), [238](#).  
*rule\_name\_*: [226](#), [227](#).  
*Rule\_no*: [226](#), [227](#).  
*rule\_no*: [226](#), [227](#).  
*rule\_no\_*: [226](#), [227](#).  
*rule\_s\_tree*: [226](#), [227](#).  
*rule\_use\_skeleton*: [226](#), [227](#).

- rule\_use\_skeleton\_*: [226](#), [227](#).  
*rules\_adding\_T\_in\_first\_set\_*: [226](#), [227](#).  
*rules\_alphabet*: [237](#), [238](#).  
*rules\_alphabet\_*: [237](#), [238](#).  
*rules\_phrase*: [69](#), [70](#).  
*rules\_phrase\_*: [68](#), [69](#), [70](#).  
*rules\_tree*: [237](#).  
*S\_no*: [175](#), [176](#).  
*second*: [164](#), [204](#), [227](#), [234](#), [242](#), [259](#).  
*set*: [54](#), [55](#), [226](#), [227](#), [268](#).  
*set\_rc*: [108](#), [153](#), [164](#), [192](#), [204](#), [217](#), [234](#), [238](#),  
[242](#), [259](#), [263](#), [274](#).  
*Srce\_expr*: [54](#), [55](#).  
*srdef*: [227](#).  
*sri*: [227](#).  
*srie*: [227](#).  
*srt*: [227](#).  
*start\_err\_enumerate*: [104](#), [105](#).  
*start\_err\_enumerate\_*: [104](#), [105](#).  
*start\_lrk\_enumerate*: [104](#), [105](#).  
*start\_lrk\_enumerate\_*: [104](#), [105](#).  
*start\_rc\_enumerate*: [104](#), [105](#).  
*start\_rc\_enumerate\_*: [104](#), [105](#).  
*start\_T\_enumerate*: [104](#), [105](#).  
*start\_T\_enumerate\_*: [104](#), [105](#).  
**status**: [248](#).  
*status\_*: [248](#), [249](#).  
*stbl\_idx*: [107](#), [108](#), [178](#), [179](#), [229](#), [230](#), [265](#),  
[266](#), [273](#), [274](#).  
*stbl\_idx\_*: [107](#), [108](#), [178](#), [179](#), [229](#), [230](#), [265](#),  
[266](#), [273](#), [274](#).  
*Stc*: [13](#), [14](#), [17](#), [18](#), [20](#), [21](#), [27](#), [28](#), [50](#), [51](#), [79](#), [80](#),  
[82](#), [83](#), [85](#), [86](#), [88](#), [89](#), [91](#), [92](#), [94](#), [95](#).  
**std**: [3](#), [54](#), [55](#), [100](#), [107](#), [108](#), [110](#), [111](#), [121](#), [122](#),  
[124](#), [125](#), [127](#), [128](#), [130](#), [131](#), [133](#), [134](#), [148](#), [149](#),  
[152](#), [153](#), [155](#), [156](#), [163](#), [164](#), [172](#), [173](#), [191](#), [192](#),  
[196](#), [197](#), [203](#), [204](#), [216](#), [217](#), [219](#), [220](#), [222](#), [223](#),  
[226](#), [227](#), [229](#), [230](#), [233](#), [234](#), [237](#), [238](#), [241](#), [242](#),  
[245](#), [246](#), [251](#), [252](#), [258](#), [259](#), [262](#), [263](#), [268](#),  
[269](#), [276](#), [277](#), [279](#), [280](#), [282](#), [283](#).  
*stop\_err\_enumerate*: [104](#), [105](#).  
*stop\_err\_enumerate\_*: [104](#), [105](#).  
*stop\_lrk\_enumerate*: [104](#), [105](#).  
*stop\_lrk\_enumerate\_*: [104](#), [105](#).  
*stop\_rc\_enumerate*: [104](#), [105](#).  
*stop\_rc\_enumerate\_*: [104](#), [105](#).  
*stop\_T\_enumerate*: [104](#), [105](#).  
*stop\_T\_enumerate\_*: [104](#), [105](#).  
*string*: [54](#), [55](#), [100](#), [110](#), [111](#), [121](#), [122](#), [124](#), [125](#),  
[127](#), [128](#), [130](#), [131](#), [133](#), [134](#), [148](#), [149](#), [152](#), [153](#),  
[155](#), [156](#), [163](#), [164](#), [167](#), [172](#), [173](#), [191](#), [192](#), [196](#),  
[197](#), [203](#), [204](#), [216](#), [217](#), [219](#), [220](#), [222](#), [223](#), [226](#),  
[227](#), [233](#), [234](#), [237](#), [238](#), [241](#), [242](#),  
[242](#), [251](#), [252](#), [258](#), [259](#), [262](#), [263](#), [268](#),  
[269](#), [276](#), [277](#), [279](#), [280](#), [282](#), [283](#).  
*String*: [276](#), [277](#).  
*subrule\_directives*: [241](#), [242](#).  
*subrule\_directives\_*: [241](#), [242](#).  
*subrule\_elems*: [241](#), [242](#).  
*subrule\_elems\_*: [241](#), [242](#).  
*subrule\_list*: [227](#).  
*subrule\_lvl*: [227](#).  
*Subrule\_no*: [241](#), [242](#).  
*subrule\_no\_of\_rule*: [241](#), [242](#).  
*subrule\_no\_of\_rule\_*: [241](#), [242](#).  
*subrule\_s\_tree*: [227](#), [241](#), [242](#).  
*Subrules*: [226](#), [227](#).  
*subrules*: [226](#), [227](#), [245](#), [246](#).  
*subrules\_*: [225](#), [226](#), [227](#), [245](#), [246](#).  
*subrules\_ph*: [227](#).  
*Supplier*: [54](#), [55](#).  
*sym*: [153](#), [164](#), [192](#), [204](#), [217](#), [234](#), [238](#), [242](#),  
[246](#), [259](#), [263](#).  
*symbol\_*: [254](#), [255](#).  
*syntax\_code*: [13](#), [14](#), [17](#), [18](#), [20](#), [21](#), [27](#), [28](#), [50](#),  
[51](#), [79](#), [80](#), [82](#), [83](#), [85](#), [86](#), [88](#), [89](#), [91](#), [92](#),  
[94](#), [95](#), [251](#), [252](#).  
*Syntax\_code*: [251](#), [252](#).  
*syntax\_code\_*: [13](#), [14](#), [17](#), [18](#), [20](#), [21](#), [27](#), [28](#), [50](#),  
[51](#), [79](#), [80](#), [82](#), [83](#), [85](#), [86](#), [88](#), [89](#), [91](#), [92](#),  
[94](#), [95](#), [251](#), [252](#).  
*T*: [107](#), [108](#), [226](#), [227](#).  
*t*: [227](#).  
*T\_angled\_string*: [110](#), [111](#).  
*T\_arbitrator\_code*: [13](#), [14](#).  
*T\_attributes*: [100](#), [101](#), [273](#), [274](#).  
*T\_basic\_char*: [114](#), [115](#).  
*T\_block*: [118](#), [119](#).  
**T\_c\_literal**: [3](#), [121](#), [122](#).  
*T\_c\_string*: [124](#), [125](#), [158](#), [159](#), [167](#), [168](#), [268](#), [269](#).  
**T\_called\_thread\_eosubrule**: [3](#), [127](#), [128](#),  
[226](#), [227](#).  
*T\_comment*: [130](#), [131](#).  
*T\_constructor*: [17](#), [18](#).  
**T\_CTOR**: [11](#), [14](#), [18](#), [21](#), [25](#), [28](#), [33](#), [46](#), [51](#), [55](#), [59](#),  
[62](#), [66](#), [70](#), [75](#), [80](#), [83](#), [86](#), [89](#), [92](#), [95](#), [101](#), [105](#),  
[108](#), [111](#), [115](#), [119](#), [122](#), [125](#), [128](#), [131](#), [134](#), [137](#),  
[146](#), [149](#), [153](#), [156](#), [159](#), [164](#), [168](#), [173](#), [176](#), [179](#),  
[182](#), [185](#), [192](#), [197](#), [204](#), [208](#), [213](#), [217](#), [220](#), [223](#),  
[227](#), [230](#), [234](#), [238](#), [242](#), [246](#), [249](#), [252](#), [255](#), [259](#),  
[263](#), [266](#), [269](#), [274](#), [277](#), [280](#), [283](#).  
*T\_cweb\_comment*: [133](#), [134](#).  
*T\_cweb\_marker*: [136](#), [137](#).  
*t\_def*: [107](#), [108](#), [220](#).  
*t\_def\_*: [107](#), [108](#).

- T\_destructor*: 20, [21](#).  
*T\_Enum*: 11, 14, 18, 21, 25, 28, 33, 46, 51, 55, 59, 62, 66, 70, 75, 80, 83, 86, 89, 92, 95, 101, 105, 108, 111, 115, 119, 122, 125, 128, 131, 134, 137, 146, 149, 153, 156, 159, 164, 168, 173, 176, 179, 182, 185, 192, 197, 204, 208, 213, 217, 220, 223, 227, 230, 234, 238, 242, 246, 249, 252, 255, 259, 263, 266, 269, 274, 277, 280, 283.  
*T\_enum*\_: [100](#), [101](#).  
**T\_enum\_phrase**: [3](#), 10, 11, 104, [105](#).  
*T\_enumeration*: 10, [11](#).  
*T\_eol*: [145](#), [146](#).  
*T\_eosubrule*: [148](#), [149](#), [242](#).  
*T\_error\_symbols*: [24](#), [25](#).  
**T\_error\_symbols\_phrase**: [3](#), [24](#), [25](#), [152](#), [153](#).  
*T\_esc\_seq*: [155](#), [156](#).  
*T\_failed*: [27](#), [28](#).  
*T\_file\_inclusion*: [158](#), [159](#).  
*T\_fsm*: [32](#), [33](#).  
**T\_fsm\_class**: [3](#).  
**T\_fsm\_class\_phrase**: [3](#), [163](#), [164](#), [167](#), [168](#).  
**T\_fsm\_phrase**: [3](#), [32](#), [33](#), [167](#), [168](#).  
**T\_identifier**: [3](#), 61, 62, 104, 105, 127, 128, 152, 153, 163, 164, 167, 168, 172, [173](#), 191, 192, 216, 217, 262, 263.  
*t\_in\_stbl*: [219](#), [220](#), [273](#), [274](#).  
**T\_in\_stbl**: [3](#), 54, 55, 107, [108](#), 219, 220, 226, [227](#).  
*t\_in\_stbl*\_: [219](#), [220](#), [273](#), [274](#).  
*T\_int\_no*: [175](#), [176](#).  
*T\_kw\_in\_stbl*: [179](#).  
*T\_la\_expr\_src*: [181](#), [182](#).  
*T\_lint*: [184](#), [185](#).  
*T\_lr1\_constant\_symbols*: [45](#), [46](#).  
**T\_lr1\_k\_phrase**: [3](#), [45](#), [46](#), [191](#), [192](#).  
*t\_name*: [153](#), [192](#), [217](#), [258](#), [259](#), 263.  
*t\_name*\_: [258](#), [259](#).  
*T\_null\_call\_thread\_eosubrule*: [196](#), [197](#).  
*T\_op*: [50](#), [51](#).  
*T\_parallel\_la\_boundary*: [54](#), [55](#), 207, [208](#).  
*T\_parallel\_monitor\_phrase*: [203](#), [204](#), [226](#), [227](#).  
*T\_parallel\_parser*: [58](#), [59](#).  
**T\_parallel\_parser\_phrase**: [3](#), [58](#), [59](#), 207, [208](#).  
*T\_parallel\_thread\_function*: [61](#), [62](#), 207, [208](#).  
*T\_raw\_char*: [212](#), [213](#).  
*T\_raw\_characters*: [65](#), [66](#).  
**T\_rc\_phrase**: [3](#), [65](#), [66](#), [216](#), [217](#).  
*T\_referred\_rule*\_: [223](#), [242](#).  
*T\_referred\_T*\_: [220](#), [242](#).  
**T\_rhs\_bnd**: [3](#).  
*T\_rule\_def*\_: [227](#).  
*T\_rule\_in\_stbl*: [230](#).  
*T\_rule\_lhs\_phrase*: [226](#), [227](#), [233](#), [234](#).  
*T\_rules*: [69](#), [70](#).  
**T\_rules\_phrase**: [3](#), 69, 70, 237, [238](#).  
*T\_subrule\_def*: [127](#), [128](#), 148, [149](#), 196, [197](#), 219, [220](#), [222](#), [223](#), [227](#), 239, 241, [242](#), 245, 246.  
**T\_subrules\_phrase**: [3](#), 226, [227](#), 245, [246](#).  
*T\_sym\_tbl\_report\_card*: [248](#), [249](#).  
**T\_syntax\_code**: [3](#), 13, 14, 17, 18, 20, 21, 27, 28, 50, 51, 79, 80, 82, 83, 85, 86, 88, 89, 91, 92, 94, 95, 104, 105, 191, 192, 251, [252](#), 262, 263.  
*T\_T\_angled\_string*\_: [111](#).  
*T\_T\_arbitrator\_code*\_: [14](#).  
*T\_T\_attributes*\_: [101](#).  
*T\_T\_basic\_char*\_: [115](#).  
*T\_T\_block*\_: [119](#).  
*T\_T\_c\_literal*\_: [122](#).  
*T\_T\_c\_string*\_: [125](#).  
*T\_T\_called\_thread\_eosubrule*\_: [128](#).  
*T\_T\_comment*\_: [131](#).  
*T\_T\_constructor*\_: [18](#).  
*T\_T\_cweb\_comment*\_: [134](#).  
*T\_T\_cweb\_marker*\_: [137](#).  
*T\_T\_destructor*\_: [21](#).  
*T\_T\_enum\_phrase*\_: [105](#).  
*T\_T\_enumeration*\_: [11](#).  
*T\_T\_eol*\_: [146](#).  
*T\_T\_eosubrule*\_: [149](#), [242](#).  
*T\_T\_error\_symbols*\_: [25](#).  
*T\_T\_error\_symbols\_phrase*\_: [153](#).  
*T\_T\_esc\_seq*\_: [156](#).  
*T\_T\_failed*\_: [28](#).  
*T\_T\_file\_inclusion*\_: [159](#).  
*T\_T\_fsm*\_: [33](#).  
*T\_T\_fsm\_class\_phrase*\_: [164](#).  
*T\_T\_fsm\_phrase*\_: [168](#).  
*T\_T\_identifier*\_: [173](#).  
*T\_T\_in\_stbl*\_: [108](#).  
*T\_T\_int\_no*\_: [176](#).  
*T\_T\_la\_expr\_src*\_: [182](#).  
*T\_T\_lint*\_: [185](#).  
*T\_T\_lr1\_constant\_symbols*\_: [46](#).  
*T\_T\_lr1\_k\_phrase*\_: [192](#).  
*T\_T\_null\_call\_thread\_eosubrule*\_: [197](#).  
*T\_T\_op*\_: [51](#).  
*T\_T\_parallel\_la\_boundary*\_: [55](#).  
*T\_T\_parallel\_monitor\_phrase*\_: [204](#).  
*T\_T\_parallel\_parser*\_: [59](#).  
*T\_T\_parallel\_parser\_phrase*\_: [208](#).  
*T\_T\_parallel\_thread\_function*\_: [62](#).  
*T\_T\_raw\_char*\_: [213](#).  
*T\_T\_raw\_characters*\_: [66](#).  
*T\_T\_rc\_phrase*\_: [217](#).  
*T\_T\_rule\_lhs\_phrase*\_: [234](#).

- T\_T\_rules\_*: 70.  
*T\_T\_rules\_phrase\_*: 238.  
*T\_T\_subrule\_def\_*: 242.  
*T\_T\_subrules\_phrase\_*: 246.  
*T\_T\_sym\_tbl\_report\_card\_*: 249.  
*T\_T\_syntax\_code\_*: 252.  
*T\_T\_terminal\_def\_*: 259.  
*T\_T\_terminals\_*: 75.  
*T\_T\_terminals\_phrase\_*: 263.  
*T\_T\_unquoted\_string\_*: 277.  
*T\_T\_user\_declaration\_*: 80.  
*T\_T\_user\_imp\_sym\_*: 83.  
*T\_T\_user\_imp\_tbl\_*: 86.  
*T\_T\_user\_implementation\_*: 89.  
*T\_T\_user\_prefix\_declaration\_*: 92.  
*T\_T\_user\_suffix\_declaration\_*: 95.  
*T\_T\_ws\_*: 280.  
*T\_T\_xc\_str\_*: 283.  
*T\_table\_entry\_*: 255.  
*T\_terminal\_def\_*: 107, [108](#), 152, 153, 191, 192, 216, 217, 219, [220](#), 258, [259](#), 262, 263.  
*T\_terminals\_*: [74](#), [75](#).  
**T\_terminals\_phrase**: [3](#), 74, 75, 262, [263](#).  
*T\_th\_in\_stbl\_*: 266.  
*T\_thread\_attributes\_*: 269.  
*T\_tth\_in\_stbl\_*: 274.  
*T\_unquoted\_string\_*: 276, [277](#).  
*T\_user\_declaration\_*: 79, [80](#), 256.  
*T\_user\_imp\_sym\_*: 82, [83](#).  
*T\_user\_imp\_tbl\_*: 85, [86](#).  
*T\_user\_implementation\_*: 88, [89](#).  
*T\_user\_prefix\_declaration\_*: 91, [92](#).  
*T\_user\_suffix\_declaration\_*: 94, [95](#).  
*T\_ws\_*: [279](#), [280](#).  
*T\_xc\_str\_*: 282, [283](#).  
*table\_entry\_*: 248, 254, [255](#).  
*tbl\_entry\_*: 248, 249.  
*Term\_in\_stbl\_*: [219](#), [220](#).  
*terminal\_*: 254.  
*terminals\_phrase\_*: [74](#), [75](#).  
*terminals\_phrase\_*: 73, [74](#), 75.  
*terminals\_refs\_code\_*: [262](#), [263](#).  
*terminals\_refs\_code\_*: 261, [262](#), 263.  
*terminals\_sufx\_code\_*: [262](#), [263](#).  
*terminals\_sufx\_code\_*: 261, [262](#), 263.  
*th\_enum\_*: [268](#), 269.  
*th\_in\_stbl\_*: 265, [266](#).  
*th\_in\_stbl\_*: 265, 266.  
*Thd\_name\_*: [128](#).  
*thread\_*: 254.  
*thread\_attributes\_*: 265, [266](#), 268, [269](#).  
*thread\_in\_stbl\_*: 265, 266.  
*Thread\_name\_*: [127](#), 268, 269.  
*thread\_name\_*: 268, 269.  
**TOKEN\_GAGGLE**: 54, [55](#), 181, [182](#).  
*Ton\_*: [226](#), [227](#), [242](#), [258](#), [259](#).  
*total\_enumerate\_*: [104](#), [105](#).  
*total\_enumerate\_*: [104](#), [105](#).  
*total\_err\_enumerate\_*: [104](#), [105](#).  
*total\_err\_enumerate\_*: [104](#), [105](#).  
*total\_lrk\_enumerate\_*: [104](#), [105](#).  
*total\_lrk\_enumerate\_*: [104](#), [105](#).  
*total\_no\_subrules\_*: [104](#), [105](#).  
*total\_no\_subrules\_*: [104](#), [105](#).  
*total\_rc\_enumerate\_*: [104](#), [105](#).  
*total\_rc\_enumerate\_*: [104](#), [105](#).  
*Total\_subrules\_*: [104](#), [105](#).  
*total\_T\_enumerate\_*: [104](#), [105](#).  
*total\_T\_enumerate\_*: [104](#), [105](#).  
*Transitive\_*: [268](#), [269](#).  
*transitive\_*: [268](#), [269](#).  
*Tree\_*: 104, 105, 152, 153, 163, 164, 167, 168, 191, 192, 203, 204, 207, 208, 216, 217, 226, 227, 233, 234, 237, 238, 245, 246, 262, 263.  
*tree\_node\_*: [127](#), [128](#), [148](#), [149](#), [196](#), [197](#), [219](#), [220](#), [222](#), [223](#), 242.  
*Tree\_node\_*: 127, 128, 148, 149, 196, 197, 219, 220, 222, 223.  
*tree\_node\_*: 127, 128, 148, 149, 196, 197, 219, 220, 222, 223.  
*true\_*: 11, 70, 75, 117, 119, 156, 255.  
*tth\_in\_stbl\_*: 273, [274](#).  
*type\_*: [254](#), 255.  
**UINT**: 107, 108, 178, 179, 229, 230, 265, 266, 273, 274.  
*unknown\_*: 248, 249, 254, 255.  
*unquoted\_string\_*: 276, 277.  
*unquoted\_string\_*: 276, 277.  
*Use\_cnt\_*: [226](#), [227](#).  
*used\_*: 254.  
*used\_*: [254](#), 255.  
*vacant\_*: [254](#), 255.  
*vector\_*: 152, [153](#), 191, [192](#), 216, [217](#), 227, 237, 238, 241, [242](#), 245, [246](#), 262, [263](#), [268](#), [269](#).  
*version\_*: [167](#), [168](#).  
*version\_*: 166, 167, 168.  
*White\_space\_data\_*: [279](#).  
*ws\_*: 280.  
*Ws\_data\_*: 280.  
*ws\_data\_*: 279, 280.  
*ws\_data\_*: 279, 280.  
*xc\_str\_*: 281.  
*xref\_r\_*: 229, [230](#).  
*xref\_r\_*: 229, 230.

*xref\_t*: [107](#), [108](#).

*xref\_t\_*: [107](#), [108](#).

**yacco2**: [3](#), [54](#), [55](#), [107](#), [108](#), [117](#), [118](#), [119](#), [146](#),  
[152](#), [153](#), [163](#), [164](#), [172](#), [173](#), [178](#), [179](#), [181](#),  
[182](#), [185](#), [191](#), [192](#), [203](#), [204](#), [216](#), [217](#), [229](#),  
[230](#), [233](#), [234](#), [237](#), [238](#), [241](#), [242](#), [258](#), [259](#),  
[262](#), [263](#), [265](#), [266](#), [273](#), [274](#), [280](#).

*yacco2\_lcl\_option*: [198](#), [199](#), [200](#).

*zero\_content*: [117](#).

*zero\_la\_tok\_can*: [181](#), [182](#).

*zero\_out\_basic\_char*: [114](#), [115](#).

*zero\_out\_raw\_char*: [212](#), [213](#).

< T-attributes user-declaration directive 100 >  
< T-attributes user-implementation directive 101 >  
< T-enum-phrase destructor directive 103 >  
< T-enum-phrase user-declaration directive 104 >  
< T-enum-phrase user-implementation directive 105 >  
< T-in-stbl user-declaration directive 107 >  
< T-in-stbl user-implementation directive 108 >  
< # T-enumeration destructor directive 9 >  
< # T-enumeration user-declaration directive 10 >  
< # T-enumeration user-implementation directive 11 >  
< # arbitrator-code user-declaration directive 13 >  
< # arbitrator-code user-implementation directive 14 >  
< # constructor user-declaration directive 17 >  
< # constructor user-implementation directive 18 >  
< # destructor user-declaration directive 20 >  
< # destructor user-implementation directive 21 >  
< # error-symbols destructor directive 23 >  
< # error-symbols user-declaration directive 24 >  
< # error-symbols user-implementation directive 25 >  
< # failed user-declaration directive 27 >  
< # failed user-implementation directive 28 >  
< # fsm destructor directive 31 >  
< # fsm user-declaration directive 32 >  
< # fsm user-implementation directive 33 >  
< # lr1-constant-symbols destructor directive 44 >  
< # lr1-constant-symbols user-declaration directive 45 >  
< # lr1-constant-symbols user-implementation directive 46 >  
< # op user-declaration directive 50 >  
< # op user-implementation directive 51 >  
< # parallel-la-boundary user-declaration directive 54 >  
< # parallel-la-boundary user-implementation directive 55 >  
< # parallel-parser destructor directive 57 >  
< # parallel-parser user-declaration directive 58 >  
< # parallel-parser user-implementation directive 59 >  
< # parallel-thread-function user-declaration directive 61 >  
< # parallel-thread-function user-implementation directive 62 >  
< # raw-characters destructor directive 64 >  
< # raw-characters user-declaration directive 65 >  
< # raw-characters user-implementation directive 66 >  
< # rules destructor directive 68 >  
< # rules user-declaration directive 69 >  
< # rules user-implementation directive 70 >  
< # terminals destructor directive 73 >  
< # terminals user-declaration directive 74 >  
< # terminals user-implementation directive 75 >  
< # user-declaration user-declaration directive 79 >  
< # user-declaration user-implementation directive 80 >  
< # user-imp-sym user-declaration directive 82 >  
< # user-imp-sym user-implementation directive 83 >  
< # user-imp-tbl user-declaration directive 85 >  
< # user-imp-tbl user-implementation directive 86 >  
< # user-implementation user-declaration directive 88 >



< # user-implementation user-implementation directive 89 >  
< # user-prefix-declaration user-declaration directive 91 >  
< # user-prefix-declaration user-implementation directive 92 >  
< # user-suffix-declaration user-declaration directive 94 >  
< # user-suffix-declaration user-implementation directive 95 >  
< angled-string user-declaration directive 110 >  
< angled-string user-implementation directive 111 >  
< basic-char destructor directive 113 >  
< basic-char user-declaration directive 114 >  
< basic-char user-implementation directive 115 >  
< block destructor directive 117 >  
< block user-declaration directive 118 >  
< block user-implementation directive 119 >  
< c-literal user-declaration directive 121 >  
< c-literal user-implementation directive 122 >  
< c-string user-declaration directive 124 >  
< c-string user-implementation directive 125 >  
< called thread eosubrle user-declaration directive 127 >  
< called thread eosubrle user-implementation directive 128 >  
< comment user-declaration directive 130 >  
< comment user-implementation directive 131 >  
< cweb-comment user-declaration directive 133 >  
< cweb-comment user-implementation directive 134 >  
< cweb-marker user-declaration directive 136 >  
< cweb-marker user-implementation directive 137 >  
< eol user-declaration directive 145 >  
< eol user-implementation directive 146 >  
< eosubrle user-declaration directive 148 >  
< eosubrle user-implementation directive 149 >  
< error-symbols-phrase destructor directive 151 >  
< error-symbols-phrase user-declaration directive 152 >  
< error-symbols-phrase user-implementation directive 153 >  
< esc-seq user-declaration directive 155 >  
< esc-seq user-implementation directive 156 >  
< file-inclusion user-declaration directive 158 >  
< file-inclusion user-implementation directive 159 >  
< fsm-class-phrase destructor directive 162 >  
< fsm-class-phrase user-declaration directive 163 >  
< fsm-class-phrase user-implementation directive 164 >  
< fsm-phrase destructor directive 166 >  
< fsm-phrase user-declaration directive 167 >  
< fsm-phrase user-implementation directive 168 >  
< identifier user-declaration directive 172 >  
< identifier user-implementation directive 173 >  
< int-no user-declaration directive 175 >  
< int-no user-implementation directive 176 >  
< kw-in-stbl user-declaration directive 178 >  
< kw-in-stbl user-implementation directive 179 >  
< la-express-source user-declaration directive 181 >  
< la-express-source user-implementation directive 182 >  
< lint user-declaration directive 184 >  
< lint user-implementation directive 185 >

`<lr1-k-phrase destructor directive 190 >`  
`<lr1-k-phrase user-declaration directive 191 >`  
`<lr1-k-phrase user-implementation directive 192 >`  
`<null call thread eosubrule user-declaration directive 196 >`  
`<null call thread eosubrule user-implementation directive 197 >`  
`<parallel-monitor-phrase destructor directive 202 >`  
`<parallel-monitor-phrase user-declaration directive 203 >`  
`<parallel-monitor-phrase user-implementation directive 204 >`  
`<parallel-parser-phrase destructor directive 206 >`  
`<parallel-parser-phrase user-declaration directive 207 >`  
`<parallel-parser-phrase user-implementation directive 208 >`  
`<raw-char destructor directive 211 >`  
`<raw-char user-declaration directive 212 >`  
`<raw-char user-implementation directive 213 >`  
`<rc-phrase destructor directive 215 >`  
`<rc-phrase user-declaration directive 216 >`  
`<rc-phrase user-implementation directive 217 >`  
`<referred-T user-declaration directive 219 >`  
`<referred-T user-implementation directive 220 >`  
`<referred-rule user-declaration directive 222 >`  
`<referred-rule user-implementation directive 223 >`  
`<rule-def destructor directive 225 >`  
`<rule-def user-declaration directive 226 >`  
`<rule-def user-implementation directive 227 >`  
`<rule-in-stbl user-declaration directive 229 >`  
`<rule-in-stbl user-implementation directive 230 >`  
`<rule-lhs-phrase destructor directive 232 >`  
`<rule-lhs-phrase user-declaration directive 233 >`  
`<rule-lhs-phrase user-implementation directive 234 >`  
`<rules-phrase destructor directive 236 >`  
`<rules-phrase user-declaration directive 237 >`  
`<rules-phrase user-implementation directive 238 >`  
`<subrule-def destructor directive 240 >`  
`<subrule-def user-declaration directive 241 >`  
`<subrule-def user-implementation directive 242 >`  
`<subrules-phrase destructor directive 244 >`  
`<subrules-phrase user-declaration directive 245 >`  
`<subrules-phrase user-implementation directive 246 >`  
`<sym-tbl-report-card user-declaration directive 248 >`  
`<sym-tbl-report-card user-implementation directive 249 >`  
`<syntax-code user-declaration directive 251 >`  
`<syntax-code user-implementation directive 252 >`  
`<table-entry user-declaration directive 254 >`  
`<table-entry user-implementation directive 255 >`  
`<terminal-def destructor directive 257 >`  
`<terminal-def user-declaration directive 258 >`  
`<terminal-def user-implementation directive 259 >`  
`<terminals-phrase destructor directive 261 >`  
`<terminals-phrase user-declaration directive 262 >`  
`<terminals-phrase user-implementation directive 263 >`  
`<terminals-refs directive 3 >`  
`<th-in-stbl user-declaration directive 265 >`

`< th-in-stbl user-implementation directive 266 >`  
`< thread-attributes user-declaration directive 268 >`  
`< thread-attributes user-implementation directive 269 >`  
`< tth-in-stbl user-declaration directive 273 >`  
`< tth-in-stbl user-implementation directive 274 >`  
`< unquoted-string user-declaration directive 276 >`  
`< unquoted-string user-implementation directive 277 >`  
`< ws user-declaration directive 279 >`  
`< ws user-implementation directive 280 >`  
`< xc-str user-declaration directive 282 >`  
`< xc-str user-implementation directive 283 >`

Terminal Vocabulary

Date: January 2, 2015 at 16:31

File: yacco2\_terminals Namespace: NS\_yacco2\_terminals

Number of terminals: 114

<b>Terminal vocabulary</b> .....	2	2
<b>Terminals-refs</b> directive .....	3	2
# <b>***</b> .....	4	2
# <b>AB</b> .....	5	3
# <b>AD</b> .....	6	3
# <b>NULL</b> .....	7	3
# <b>T-enumeration</b> .....	8	3
# T-enumeration destructor directive .....	9	3
# T-enumeration user-declaration directive .....	10	3
# T-enumeration user-implementation directive .....	11	4
# <b>arbitrator-code</b> .....	12	4
# arbitrator-code user-declaration directive .....	13	4
# arbitrator-code user-implementation directive .....	14	5
# <b>constant-defs</b> .....	15	5
# <b>constructor</b> .....	16	5
# constructor user-declaration directive .....	17	5
# constructor user-implementation directive .....	18	6
# <b>destructor</b> .....	19	6
# destructor user-declaration directive .....	20	6
# destructor user-implementation directive .....	21	6
# <b>error-symbols</b> .....	22	6
# error-symbols destructor directive .....	23	7
# error-symbols user-declaration directive .....	24	7
# error-symbols user-implementation directive .....	25	7
# <b>failed</b> .....	26	7
# failed user-declaration directive .....	27	7
# failed user-implementation directive .....	28	8
# <b>file-name</b> .....	29	8
# <b>fsm</b> .....	30	8
# fsm destructor directive .....	31	8
# fsm user-declaration directive .....	32	8
# fsm user-implementation directive .....	33	9
# <b>fsm-class</b> .....	34	9
# <b>fsm-comments</b> .....	35	9
# <b>fsm-date</b> .....	36	9
# <b>fsm-debug</b> .....	37	9
# <b>fsm-filename</b> .....	38	9
# <b>fsm-id</b> .....	39	9
# <b>fsm-namespace</b> .....	40	10
# <b>fsm-version</b> .....	41	10
# <b>lhs</b> .....	42	10
# <b>lr1-constant-symbols</b> .....	43	10
# lr1-constant-symbols destructor directive .....	44	10
# lr1-constant-symbols user-declaration directive .....	45	10
# lr1-constant-symbols user-implementation directive .....	46	10
# <b>lrk-suffix</b> .....	47	11
# <b>name-space</b> .....	48	11
# <b>op</b> .....	49	11
# op user-declaration directive .....	50	11
# op user-implementation directive .....	51	11
# <b>parallel-control-monitor</b> .....	52	11
# <b>parallel-la-boundary</b> .....	53	12

# parallel-la-boundary user-declaration directive .....	54	12
# parallel-la-boundary user-implementation directive .....	55	13
# <b>parallel-parser</b> .....	56	13
# parallel-parser destructor directive .....	57	14
# parallel-parser user-declaration directive .....	58	14
# parallel-parser user-implementation directive .....	59	14
# <b>parallel-thread-function</b> .....	60	14
# parallel-thread-function user-declaration directive .....	61	14
# parallel-thread-function user-implementation directive .....	62	15
# <b>raw-characters</b> .....	63	15
# raw-characters destructor directive .....	64	15
# raw-characters user-declaration directive .....	65	15
# raw-characters user-implementation directive .....	66	16
# <b>rules</b> .....	67	16
# rules destructor directive .....	68	16
# rules user-declaration directive .....	69	16
# rules user-implementation directive .....	70	16
# <b>sym-class</b> .....	71	17
# <b>terminals</b> .....	72	17
# terminals destructor directive .....	73	17
# terminals user-declaration directive .....	74	17
# terminals user-implementation directive .....	75	17
# <b>terminals-refs</b> .....	76	17
# <b>terminals-suffix</b> .....	77	17
# <b>user-declaration</b> .....	78	18
# user-declaration user-declaration directive .....	79	18
# user-declaration user-implementation directive .....	80	18
# <b>user-imp-sym</b> .....	81	18
# user-imp-sym user-declaration directive .....	82	18
# user-imp-sym user-implementation directive .....	83	19
# <b>user-imp-tbl</b> .....	84	19
# user-imp-tbl user-declaration directive .....	85	19
# user-imp-tbl user-implementation directive .....	86	19
# <b>user-implementation</b> .....	87	19
# user-implementation user-declaration directive .....	88	20
# user-implementation user-implementation directive .....	89	20
# <b>user-prefix-declaration</b> .....	90	20
# user-prefix-declaration user-declaration directive .....	91	20
# user-prefix-declaration user-implementation directive .....	92	21
# <b>user-suffix-declaration</b> .....	93	21
# user-suffix-declaration user-declaration directive .....	94	21
# user-suffix-declaration user-implementation directive .....	95	21
- > .....	96	21
:: .....	97	22
<b>T-alphabet</b> .....	98	22
<b>T-attributes</b> .....	99	22
T-attributes user-declaration directive .....	100	22
T-attributes user-implementation directive .....	101	22
<b>T-enum-phrase</b> .....	102	22
T-enum-phrase destructor directive .....	103	22
T-enum-phrase user-declaration directive .....	104	23
T-enum-phrase user-implementation directive .....	105	25

<b>T-in-stbl</b> .....	106	28
T-in-stbl user-declaration directive .....	107	28
T-in-stbl user-implementation directive .....	108	29
<b>angled-string</b> .....	109	29
angled-string user-declaration directive .....	110	29
angled-string user-implementation directive .....	111	30
<b>basic-char</b> .....	112	30
basic-char destructor directive .....	113	30
basic-char user-declaration directive .....	114	30
basic-char user-implementation directive .....	115	30
<b>block</b> .....	116	31
block destructor directive .....	117	31
block user-declaration directive .....	118	31
block user-implementation directive .....	119	31
<b>c-literal</b> .....	120	31
c-literal user-declaration directive .....	121	31
c-literal user-implementation directive .....	122	32
<b>c-string</b> .....	123	32
c-string user-declaration directive .....	124	32
c-string user-implementation directive .....	125	32
<b>called thread eosubrle</b> .....	126	32
called thread eosubrle user-declaration directive .....	127	33
called thread eosubrle user-implementation directive .....	128	34
<b>comment</b> .....	129	35
comment user-declaration directive .....	130	35
comment user-implementation directive .....	131	35
<b>cweb-comment</b> .....	132	35
cweb-comment user-declaration directive .....	133	35
cweb-comment user-implementation directive .....	134	36
<b>cweb-marker</b> .....	135	36
cweb-marker user-declaration directive .....	136	36
cweb-marker user-implementation directive .....	137	36
<b>emitfile</b> .....	138	36
<b>end-T-alphabet</b> .....	139	36
<b>end-list-of-native-first-set-terminals</b> .....	140	37
<b>end-list-of-transitive-threads</b> .....	141	37
<b>end-list-of-used-threads</b> .....	142	37
<b>end-preamble</b> .....	143	37
<b>eol</b> .....	144	37
eol user-declaration directive .....	145	37
eol user-implementation directive .....	146	37
<b>eosubrle</b> .....	147	37
eosubrle user-declaration directive .....	148	38
eosubrle user-implementation directive .....	149	39
<b>error-symbols-phrase</b> .....	150	39
error-symbols-phrase destructor directive .....	151	40
error-symbols-phrase user-declaration directive .....	152	40
error-symbols-phrase user-implementation directive .....	153	41
<b>esc-seq</b> .....	154	42
esc-seq user-declaration directive .....	155	42
esc-seq user-implementation directive .....	156	42
<b>file-inclusion</b> .....	157	43

file-inclusion user-declaration directive	158	43
file-inclusion user-implementation directive	159	43
<b>file-of-T-alphabet</b>	160	43
<b>fsm-class-phrase</b>	161	44
fsm-class-phrase destructor directive	162	44
fsm-class-phrase user-declaration directive	163	44
fsm-class-phrase user-implementation directive	164	45
<b>fsm-phrase</b>	165	46
fsm-phrase destructor directive	166	46
fsm-phrase user-declaration directive	167	47
fsm-phrase user-implementation directive	168	48
<b>grammar-name</b>	169	50
<b>grammar-phrase</b>	170	50
<b>identifier</b>	171	50
identifier user-declaration directive	172	50
identifier user-implementation directive	173	50
<b>int-no</b>	174	50
int-no user-declaration directive	175	50
int-no user-implementation directive	176	51
<b>kw-in-stbl</b>	177	51
kw-in-stbl user-declaration directive	178	51
kw-in-stbl user-implementation directive	179	51
<b>la-express-source</b>	180	52
la-express-source user-declaration directive	181	52
la-express-source user-implementation directive	182	52
<b>lint</b>	183	52
lint user-declaration directive	184	52
lint user-implementation directive	185	52
<b>list-of-native-first-set-terminals</b>	186	52
<b>list-of-transitive-threads</b>	187	53
<b>list-of-used-threads</b>	188	53
<b>lr1-k-phrase</b>	189	53
lr1-k-phrase destructor directive	190	53
lr1-k-phrase user-declaration directive	191	53
lr1-k-phrase user-implementation directive	192	54
<b>monolithic</b>	193	55
<b>no-of-T</b>	194	55
<b>null call thread eosubrule</b>	195	55
null call thread eosubrule user-declaration directive	196	56
null call thread eosubrule user-implementation directive	197	57
<b>option-err</b>	198	58
<b>option-p</b>	199	58
<b>option-t</b>	200	58
<b>parallel-monitor-phrase</b>	201	58
parallel-monitor-phrase destructor directive	202	58
parallel-monitor-phrase user-declaration directive	203	58
parallel-monitor-phrase user-implementation directive	204	59
<b>parallel-parser-phrase</b>	205	59
parallel-parser-phrase destructor directive	206	60
parallel-parser-phrase user-declaration directive	207	60
parallel-parser-phrase user-implementation directive	208	61
<b>preamble</b>	209	61



<b>raw-char</b> .....	210	62
raw-char destructor directive .....	211	62
raw-char user-declaration directive .....	212	62
raw-char user-implementation directive .....	213	62
<b>rc-phrase</b> .....	214	62
rc-phrase destructor directive .....	215	62
rc-phrase user-declaration directive .....	216	63
rc-phrase user-implementation directive .....	217	64
<b>referred-T</b> .....	218	65
referred-T user-declaration directive .....	219	65
referred-T user-implementation directive .....	220	66
<b>referred-rule</b> .....	221	67
referred-rule user-declaration directive .....	222	67
referred-rule user-implementation directive .....	223	68
<b>rule-def</b> .....	224	69
rule-def destructor directive .....	225	69
rule-def user-declaration directive .....	226	70
rule-def user-implementation directive .....	227	72
<b>rule-in-stbl</b> .....	228	76
rule-in-stbl user-declaration directive .....	229	76
rule-in-stbl user-implementation directive .....	230	77
<b>rule-lhs-phrase</b> .....	231	77
rule-lhs-phrase destructor directive .....	232	77
rule-lhs-phrase user-declaration directive .....	233	78
rule-lhs-phrase user-implementation directive .....	234	79
<b>rules-phrase</b> .....	235	80
rules-phrase destructor directive .....	236	80
rules-phrase user-declaration directive .....	237	80
rules-phrase user-implementation directive .....	238	81
<b>subrule-def</b> .....	239	82
subrule-def destructor directive .....	240	82
subrule-def user-declaration directive .....	241	83
subrule-def user-implementation directive .....	242	84
<b>subrules-phrase</b> .....	243	87
subrules-phrase destructor directive .....	244	87
subrules-phrase user-declaration directive .....	245	87
subrules-phrase user-implementation directive .....	246	88
<b>sym-tbl-report-card</b> .....	247	88
sym-tbl-report-card user-declaration directive .....	248	89
sym-tbl-report-card user-implementation directive .....	249	89
<b>syntax-code</b> .....	250	89
syntax-code user-declaration directive .....	251	89
syntax-code user-implementation directive .....	252	90
<b>table-entry</b> .....	253	90
table-entry user-declaration directive .....	254	90
table-entry user-implementation directive .....	255	91
<b>terminal-def</b> .....	256	91
terminal-def destructor directive .....	257	91
terminal-def user-declaration directive .....	258	92
terminal-def user-implementation directive .....	259	93
<b>terminals-phrase</b> .....	260	95
terminals-phrase destructor directive .....	261	95

terminals-phrase user-declaration directive .....	262	95
terminals-phrase user-implementation directive .....	263	96
<b>th-in-stbl</b> .....	264	97
th-in-stbl user-declaration directive .....	265	98
th-in-stbl user-implementation directive .....	266	98
<b>thread-attributes</b> .....	267	98
thread-attributes user-declaration directive .....	268	99
thread-attributes user-implementation directive .....	269	100
<b>thread-name</b> .....	270	100
<b>transitive</b> .....	271	100
<b>tth-in-stbl</b> .....	272	101
tth-in-stbl user-declaration directive .....	273	101
tth-in-stbl user-implementation directive .....	274	101
<b>unquoted-string</b> .....	275	101
unquoted-string user-declaration directive .....	276	101
unquoted-string user-implementation directive .....	277	102
<b>ws</b> .....	278	102
ws user-declaration directive .....	279	102
ws user-implementation directive .....	280	102
<b>xc-str</b> .....	281	102
xc-str user-declaration directive .....	282	103
xc-str user-implementation directive .....	283	103
<b>Index</b> .....	284	104