1. Copyright.

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→lr1_k_phrase_ ≠ 0) delete R→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-sufx.
Enum: T_T_lrk_sufx_
Class: T_lrk_sufx 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 simplifiy 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-sufx.
Enum: T_T_terminals_sufx_
Class: T_terminals_sufx 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 fully_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 < refered_T * > *xref_t( );
  void add_T_into_xref(refered_T &T);
  void stbl_idx(yacco2::UINT Idx);
  int stbl_idx( );
private: T_terminal_def * t_def_; std::list < refered_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 < refered_T * > *T_in_stbl::xref_t( )
  {
    return &xref_t_;
    ;
  }
  void T_in_stbl::add_T_into_xref(refered_T &T)
  {
    xref_t_.push_back(&T);
  }
  void T_in_stbl::stbl_idx(yacco2::UINT Idx)
  {
    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 ulgy 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 eosubrule.

Enum: T_T_called_thread_eosubrule_
Class: T_called_thread_eosubrule AB: N AD: N

---

127. called thread eosubrule user-declaration directive.

⟨ called thread eosubrule user-declaration directive 127 ⟩ ≡

public: **T_called_thread_eosubrule**(**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: N
This 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. eosubrule.
Enum: T_T_eosubrule_
Class: T_eosubrule 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::string key(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 TEXparlance.

---

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;
  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;
  }
```

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::string key(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_null_call_thread_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_;
```

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_fnct);
  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_fnct)
{
  pp_funct_ = PP_fnct;
}
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. rc-phrase user-declaration directive.

⟨ rc-phrase user-declaration directive 216 ⟩ ≡

```
public: T_rc_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_;
```

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::string key(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. refered-T user-implementation directive.

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

221. refered-rule.
Enum: T_refered_rule_
Class: refered_rule AB: N AD: N

---

222. refered-rule user-declaration directive.
⟨ refered-rule user-declaration directive 222 ⟩ ≡

```
public: refered_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. refered-rule user-implementation directive.

```
⟨ refered-rule user-implementation directive 223 ⟩ ≡
  refered_rule :: refered_rule(rule_in_stbl &Rule_in_stbl)T_CTOR("refered-rule",
          T_Enum :: T_refered_rule_, 0, false, false)
  {
    rule_in_stbl_ = &Rule_in_stbl;
    its_subrule_def_ = 0;
    tree_node_ = 0;
    element_pos_ = 0;
  }
  int refered_rule :: element_pos()
  {
    return element_pos_;
  }
  void refered_rule :: element_pos(int Pos)
  {
    element_pos_ = Pos;
  }
  rule_def *refered_rule :: its_rule_def()
  {
    return rule_in_stbl_→r_def();
  }
  AST *refered_rule :: tree_node()
  {
    return tree_node_;
  }
  void refered_rule :: tree_node(AST *Tree_node)
  {
    tree_node_ = Tree_node;
  }
  std :: string *refered_rule :: grammar_s_enumerate()
  {
    return &grammar_s_enumerate_;
  }
  void refered_rule :: grammar_s_enumerate(const char *Enumerate)
  {
    grammar_s_enumerate_ += Enumerate;
  }
  T_subrule_def *refered_rule :: its_subrule_def()
  {
    return its_subrule_def_;
  }
  void refered_rule :: its_subrule_def(T_subrule_def *Its_subrule)
  {
    its_subrule_def_ = Its_subrule;
  }
  rule_in_stbl *refered_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 < refered_rule * > *xref_r( );
  void add_R_into_xref(refered_rule &R);
  void stbl_idx(yacco2::UINT Idx);
  int stbl_idx( );
private: rule_def *r_def_; std::list < refered_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 < refered_rule * > *rule_in_stbl::xref_r()
  {
    return &xref_r_;
    ;
  }
  void rule_in_stbl::add_R_into_xref(refered_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;
  string key(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_,
        &dtor_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::string key(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 garantee ascending order, this does.

---

240. subrule-def destructor directive.

⟨ subrule-def destructor directive 240 ⟩ ≡
R→*remove_subrule_directives*( );
R→*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_refered_rule_:
    {
      refered_rule *rrule = (refered_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)
{
  string key(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, "*n*ogood*b*". 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.

⟨ 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