# TinyTalk

1.0.0

Generated by Doxygen 1.9.1

**1 TT Language** 1

1.1 Introduction . . . 1

**2 TT Technical Details** 3

2.1 Main Features . . . 3

2.2 Details . . . 3

2.3 Chapter 1: Memory Management . . . 3

2.4 Syntax . . . 3

2.5 Chapter 3: Implementation . . . 4

**3 Module Index** 5

3.1 Modules . . . 5

**4 Data Structure Index** 7

4.1 Data Structures . . . 7

**5 Module Documentation** 9

5.1 ITab . . . 9

5.1.1 Detailed Description . . . 9

5.1.2 Function Documentation . . . 9

5.1.2.1 itab_entry_cmp() . . . 10

5.1.2.2 itab_new() . . . 10

5.2 Runtime Context . . . 10

5.2.1 Detailed Description . . . 11

5.3 Tokenizer . . . 11

5.3.1 Detailed Description . . . 11

5.3.2 Function Documentation . . . 11

5.3.2.1 is_ident_char() . . . 11

5.3.2.2 nextToken() . . . 12

5.3.2.3 readChar() . . . 12

5.3.2.4 readLine() . . . 12

5.3.2.5 readStringToken() . . . 12

5.4 C Code Generator . . . 13

5.4.1 Detailed Description . . . 13

5.4.2 Function Documentation . . . 13

5.4.2.1 c_generate() . . . 13

5.5 Assign . . . 13

5.5.1 Detailed Description . . . 13

**6 Data Structure Documentation** 15

6.1 _Assign Struct Reference . . . 15

6.2 _Assigns Struct Reference . . . 15

6.3 ast Struct Reference . . . 16

6.3.1 Field Documentation . . . 17

6.3.1.1 key . . . 17
6.3.1.2 next . . . 18
6.3.1.3 v . . . 18
6.4 classinfo Struct Reference . . . 18
6.5 context Struct Reference . . . 18
6.6 contextdef Struct Reference . . . 19
6.7 gd Struct Reference . . . 19
6.8 itab Struct Reference . . . 20
6.8.1 Detailed Description . . . 20
6.9 itab_entry Struct Reference . . . 21
6.9.1 Detailed Description . . . 21
6.10 itab_iter Struct Reference . . . 21
6.10.1 Detailed Description . . . 22
6.11 meth Struct Reference . . . 22
6.12 methodinfo Struct Reference . . . 22
6.13 stringinfo Struct Reference . . . 23
6.14 varinfo Struct Reference . . . 23
6.15 yyParser Struct Reference . . . 23
6.16 yyStackEntry Struct Reference . . . 24

# Chapter 1

# TT Language

## 1.1 Introduction

TT Technical Details

# Chapter 2

# TT Technical Details

## 2.1 Main Features

## 2.2 Details

Chapter 1: Memory Management

Syntax

Chapter 3: Implementation

## 2.3 Chapter 1: Memory Management

## 2.4 Syntax

```
object_ident ::= IDENT.
object_ident ::= IDENT IDENT.
unary_pattern ::= IDENT.
binary_pattern ::= BINOP IDENT.
keyword_pattern ::= KEYWORD IDENT.
keyword_pattern ::= keyword_pattern KEYWORD IDENT.
all ::= object_defs.
object_defs ::=.
object_defs ::= object_defs object_ident LBRACK var_list method_defs RBRACK.
object_defs ::= object_defs object_ident LARROW IDENT LBRACK var_list method_defs RBRACK.
var_list ::=.
var_list ::= BAR idents BAR.
idents ::= IDENT.
idents ::= idents IDENT.
method_defs ::=.
method_defs ::= method_defs msg_pattern LBRACK var_list statements RBRACK.
method_defs ::= method_defs msg_pattern VERBATIM.
msg_pattern ::= unary_pattern.
msg_pattern ::= binary_pattern.
msg_pattern ::= keyword_pattern.
statements ::= return_statement.
statements ::= return_statement DOT.
statements ::= expression DOT statements.
statements ::= expression.
statements ::= expression DOT.
```

```
return_statement ::= UARROW expression.
expression ::= IDENT LARROW expr.
expression ::= basic_expression.
basic_expression ::= primary.
basic_expression ::= primary messages cascaded_messages.
basic_expression ::= primary cascaded_messages.
basic_expression ::= primary messages.
primary ::= IDENT.
primary ::= STRING.
primary ::= LBRACK block_body RBRACK.
primary ::= LBRACE expression RBRACE.
block_body ::= block_arguments BAR var_list statements.
block_body ::= var_list statements.
block_body ::= var_list.
block_arguments ::= COLON IDENT.
block_arguments ::= block_arguments COLON IDENT.
messages ::= unary_messages.
messages ::= unary_messages keyword_message.
messages ::= unary_messages binary_messages.
messages ::= unary_messages binary_messages keyword_message.
messages ::= binary_messages.
messages ::= binary_messages keyword_message.
messages ::= keyword_message.
unary_messages ::= IDENT.
binary_messages ::= binary_message.
binary_messages ::= binary_message binary_messages.
binary_message ::= BINOP binary_argument.
binary_argument ::= primary unary_messages.
binary_argument ::= primary.
keyword_message ::= KEYWORD keyword_argument.
keyword_message ::= keyword_message KEYWORD keyword_argument.
keyword_argument ::= primary.
keyword_argument ::= primary unary_messages.
keyword_argument ::= primary unary_messages binary_messages.
cascaded_messages ::= SEMICOLON messages.
cascaded_messages ::= cascaded_messages SEMICOLON messages.
atom ::= IDENT.
atom ::= STRING.
unary_call ::= unary_call IDENT.
binary_call ::= binary_call BINOP unary_call.
unary_call ::= atom.
binary_call ::= unary_call.
expr ::= binary_call.
```

## 2.5 Chapter 3: Implementation

# Chapter 3

# Module Index

## 3.1 Modules

Here is a list of all modules:


ITab . . . . . 9
Runtime Context . . . . . 10
Tokenizer . . . . . 11
Internal Representation . . . . . **??**
    C Code Generator . . . . . 13
Assign . . . . . 13

# Chapter 4

# Data Structure Index

## 4.1 Data Structures

Here are the data structures with brief descriptions:


_Assign . . . 15
_Assigns . . . 15
ast . . . 16
classinfo . . . 18
context . . . 18
contextdef . . . 19
gd . . . 19
itab
    Structure of itab . . . 20
itab_entry
    Structure of an entry in the itab . . . 21
itab_iter
    Iterator over elements of an itab . . . 21
meth . . . 22
methodinfo . . . 22
s_names . . . ??
s_pattern . . . ??
stringinfo . . . 23
varinfo . . . 23
yyParser . . . 23
yyStackEntry . . . 24

# Chapter 5

# Module Documentation

## 5.1 ITab

### Data Structures

- struct itab_entry

  *structure of an entry in the itab.*
- struct itab

  *structure of itab*
- struct itab_iter

  *iterator over elements of an itab.*

### Functions

- struct itab ∗ itab_new ()

  *create a new itab with default parameters.*
- int itab_entry_cmp (const void ∗aptr, const void ∗bptr)

  *compares the keys of two entries*
- void **itab_append** (struct itab ∗itab, const char ∗key, void ∗value)
- void ∗ **itab_read** (struct itab ∗itab, const char ∗key)
- void **itab_dump** (struct itab ∗itab)
- struct itab_iter ∗ **itab_foreach** (struct itab ∗tab)
- struct itab_iter ∗ **itab_next** (struct itab_iter ∗iter)
- void ∗ **itab_value** (struct itab_iter ∗iter)
- const char ∗ **itab_key** (struct itab_iter ∗iter)

### 5.1.1 Detailed Description

sorted list of structures -> tables with primary index

### 5.1.2 Function Documentation

#### 5.1.2.1 itab_entry_cmp()

```
int itab_entry_cmp (
            const void * aptr,
            const void * bptr )
```

compares the keys of two entries

**Returns**

- < 0, when first key is lower
- == 0, when both keys are equal
- > 0, when second key is lower

```
00180                                                           {
00181     const struct itab_entry *a = aptr;
00182     const struct itab_entry *b = bptr;
00183     return strcmp( a->key, b->key );
00184 }
```

#### 5.1.2.2 itab_new()

```
struct itab* itab_new ( )
```

create a new itab with default parameters.

**Returns**

reference to an itab structure.

Detailed description follows here.

```
00166                         {
00167     struct itab *r = talloc( NULL, struct itab );
00168     r->total = 10;
00169     r->used = 0;
00170     r->rows = talloc_array( r, struct itab_entry, r->total );
00171     return r;
00172 }
```

## 5.2 Runtime Context

### Functions

- struct context * **context_new** (struct context *super)
- struct contextdef * **context_lookup** (struct context *ctx, const char *name)

### Variables

- struct contextdef **context_global_def** = {.global = true}
- struct contextdef **context_class_def** = {.instance = true }

### 5.2.1 Detailed Description

## 5.3 Tokenizer

### Functions

- bool is_ident_char (int c)

  *check if character is part of an identifier.*
- bool **is_binary_char** (int c)
- bool readLine ()

  *read one line from stdin stores the result into {gd.line}.*
- bool readChar (char *t)

  *read one character from input and store it somewhere.*
- bool readStringToken (void)

  *read string token.*
- bool nextToken (void)

  *read next token.*

### 5.3.1 Detailed Description

convert stdin into tokens. each token is returned by the call to

**See also**

nextToken.

### 5.3.2 Function Documentation

#### 5.3.2.1 is_ident_char()

```
bool is_ident_char (
            int c )
```

check if character is part of an identifier.

**Parameters**

| | | |
|---|---|---|
| in | *c* | character to classify. |

**Returns**

true if if c is an identifier character.

```
00308                               {
00309     return isalpha( c ) || isdigit( c ) || c == '_';
00310 }
```

Referenced by nextToken().

#### 5.3.2.2 nextToken()

```
bool nextToken ( )
```

read next token.

This is a more detailed description.

**Returns**

true if successful

```
00403                   {
00404     char c;
00405     bool result = false;
00406     while( true ) {
00407         while( readChar( &c ) && isspace( c ) );
00408         if( c == '"' ) {
00409             while( readChar( &c ) && c != '"' );
00410         }
00411         else
00412             break;
00413     }
00414     if( gd.state == 1 ) {
00415         if( isalpha( c ) ) {
00416             int idx = 0;
00417             for( ;; ) {
00418                 gd.buf[idx++] = c;
00419                 readChar( &c );
00420                 if( !is_ident_char( c ) )
00421                     break;
00422             }
00423             if( c == ':' ) {
00424                 gd.buf[idx++] = c;
00425                 gd.token = TK_KEYWORD;
00426             }
00427             else {
00428                 gd.pos--;
00429                 gd.token = TK_IDENT;
00430             }
00431             gd.buf[idx] = 0;
00432             result = true;
00433         }
00434         else if( is_binary_char( c ) ) {
00435             for( int idx = 0; is_binary_char( c ); idx++ ) {
00436                 gd.buf[idx] = c;
00437                 gd.buf[idx + 1] = 0;
00438                 readChar( &c );
00439             }
00440             gd.pos--;
00441             gd.token = 0;
00442             gd.token = TK_BINOP;
00443             result = true;
00444             if( strcmp( "<-", gd.buf ) == 0 ) {
00445                 gd.token = TK_LARROW;
00446                 result = true;
00447             }
00448             else if( strcmp( "|", gd.buf ) == 0 ) {
00449                 gd.token = TK_BAR;
00450                 result = true;
00451             }
00452             else if( 0 == strcmp( "<", gd.buf ) ) {
00453                 gd.token = TK_LT;
00454                 result = true;
00455             }
00456             else if( 0 == strcmp( ">", gd.buf ) ) {
00457                 gd.token = TK_GT;
00458                 result = true;
00459             }
00460         }
00461         else if( isdigit( c ) ) {
00462             int idx = 1;
00463             while( isdigit( c ) ) {
00464                 gd.buf[idx - 1] = c;
```

```
00465                 gd.buf[idx] = 0;
00466                 readChar( &c );
00467             }
00468             gd.pos--;
00469             gd.token = TK_NUMBER;
00470             result = true;
00471         }
00472         else {
00473             switch ( c ) {
00474                 case '\"':
00475                     result = readStringToken(  );
00476                     break;
00477                 case '.':
00478                     result = true;
00479                     gd.token = TK_DOT;
00480                     break;
00481                 case ';':
00482                     result = true;
00483                     gd.token = TK_SEMICOLON;
00484                     break;
00485                 case '(':
00486                     result = true;
00487                     gd.token = TK_LBRACE;
00488                     break;
00489                 case ')':
00490                     result = true;
00491                     gd.token = TK_RBRACE;
00492                     break;
00493                 case '[':
00494                     result = true;
00495                     gd.token = TK_LBRACK;
00496                     break;
00497                 case ']':
00498                     result = true;
00499                     gd.token = TK_RBRACK;
00500                     break;
00501                 case '{':
00502                     {
00503                         int i = 0;
00504                         gd.buf[i] = 0;
00505                         readChar( &c );
00506                         while( c != '}' ) {
00507                             gd.buf[i++] = c;
00508                             gd.buf[i] = 0;
00509                             readChar( &c );
00510                         }
00511                         gd.token = TK_VERBATIM;
00512                     }
00513                     result = true;
00514                     break;
00515                 case '#':
00516                     readChar( &c );
00517                     for( int idx = 0; is_ident_char( c ) || c == ':'; idx++ ) {
00518                         gd.buf[idx] = c;
00519                         gd.buf[idx + 1] = 0;
00520                         readChar( &c );
00521                     }
00522                     gd.pos--;
00523                     gd.token = TK_SYMBOL;
00524                     result = true;
00525                     break;
00526                 case '^':
00527                     result = true;
00528                     gd.token = TK_UARROW;
00529                     break;
00530                 case ':':
00531                     result = true;
00532                     gd.token = TK_COLON;
00533                     break;
00534                 case '$':
00535                     result = true;
00536                     gd.token = TK_CHAR;
00537                     readChar( &c );
00538                     gd.buf[0] = c;
00539                     gd.buf[1] = 0;
00540                     break;
00541                 default:
00542                     gd.pos--;
00543                     break;
00544             }
00545         }
00546     }
00547     return result;
00548 }
```

References is_ident_char(), and readChar().

#### 5.3.2.3 readChar()

```
bool readChar (
            char * t )
```

read one character from input and store it somewhere.

**Parameters**

| | | |
|---|---|---|
| in | *t* | c-string of some sort. |

**Returns**

true if successful

```
00367                           {
00368     bool result = true;
00369     if( gd.state == 0 ) {
00370         result = readLine(  );
00371     }
00372     if( result ) {
00373         *t = gd.line[gd.pos++];
00374         while( *t == 0 ) {
00375             if( readLine(  ) ) {
00376                 *t = gd.line[gd.pos++];
00377             }
00378             else {
00379                 result = false;
00380                 break;
00381             }
00382         }
00383     }
00384     return result;
00385 }
```

References readLine().

Referenced by nextToken(), and readStringToken().

#### 5.3.2.4 readLine()

```
bool readLine ( )
```

read one line from stdin stores the result into {gd.line}.

trailing blanks are removed.

```
00345                     {
00346     static int line_count = 0;
00347
00348     char *line = fgets( gd.line, sizeof( gd.line ), stdin );
00349     line_count++;
00350     printf( "%2d:%s", line_count, line );
00351     if( line ) {
00352         size_t len = strlen( line );
00353         while( len >= 0 && line[len] <= 32 )
00354             line[len--] = 0;
00355         gd.pos = 0;
00356         gd.state = 1;
00357         return true;
00358     }
00359     else {
00360         gd.line[0] = 0;
00361         gd.state = 2;
00362         return false;
00363     }
00364 }
```

Referenced by readChar().

#### 5.3.2.5 readStringToken()

```
bool readStringToken (
             void  )
```

read string token.

**Returns**

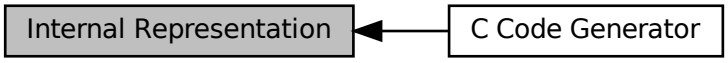
true if successful

```
00387                           {
00388     int idx = 0;
00389     char c;
00390     while( readChar( &c ) && '\"' != c ) {
00391         if( c == '\\' )
00392             readChar( &c );
00393         gd.buf[idx++] = c;
00394     }
00395     gd.buf[idx] = 0;
00396     gd.token = TK_STRING;
00397     return true;
00398 }
```

References readChar().

## 5.4 Internal Representation

Collaboration diagram for Internal Representation:




### Modules

- C Code Generator

### Data Structures

- struct s_names
- struct s_pattern

### Typedefs

- typedef struct s_names * **t_names**
- typedef struct s_pattern * **t_pattern**

### Functions

- void **method_def** (t_pattern pattern, void *locals, void *directive, void *statements)
- void **method_def_verb** (t_pattern pattern, void *coding)

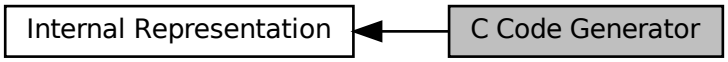
### Variables

- bool **classinfo::meta**
- char * **classinfo::name**
- char * **classinfo::super**
- int **classinfo::num**
- char * **methodinfo::classname**
- char * **methodinfo::name**
- char * **varinfo::classname**
- char * **varinfo::name**
- int **stringinfo::num**
- char * **_Assign::name**
- char * **_Assign::value**
- Assign **_Assigns::assign**
- Assigns **_Assigns::next**
- const char * **itab_entry::key**
- void * **itab_entry::value**
- int **itab::total**
- int **itab::used**
- struct itab_entry * **itab::rows**
- struct itab * **itab_iter::tab**
- int **itab_iter::pos**
- char * **meth::name**
- struct meth * **meth::next**

#### 5.4.1 Detailed Description

## 5.5 C Code Generator

Collaboration diagram for C Code Generator:




### Functions

- void c_generate (FILE *out)

  *generate the C-Code into the file stream.*

### 5.5.1 Detailed Description

|}

### 5.5.2 Function Documentation

#### 5.5.2.1 c_generate()

```
void c_generate (
            FILE * out )
```

generate the C-Code into the file stream.

**Parameters**

| | | |
|---|---|---|
| in | *FILE** | out |

```
01003                                {
01004
01005     itab_dump( variables );
01006     c_generate_structs( out );
01007     c_generate_protos( out );
01008     c_generate_strings( out );
01009     c_generate_blocks( out );
01010     c_generate_dispatchers( out );
01011
01012 }
```

## 5.6 Assign

### Functions

- Assign Assign_new (char *name, char *value)
- Assigns Assigns_new (Assign assign, Assigns next)
- void Assign_add (char *name, char *value)
- Assign Assign_find (char *name)
- char * Assign_value (Assign a)
- char * Assign_name (Assign a)
- void Assigns_dump (Assigns as)

### 5.6.1 Detailed Description

### 5.6.2 Function Documentation

#### 5.6.2.1 Assign_add()

```
void Assign_add (
            char * name,
            char * value )
```

add an assignment pair to the global list. No additional checks are done.

```
00108                                                {
00109     Assign a = Assign_new( name, value );
00110     global_assigns = Assigns_new( a, global_assigns );
00111 }
```

References Assign_new(), and Assigns_new().

#### 5.6.2.2 Assign_find()

```
Assign Assign_find (
            char * name )
```

find a pair in the global assignment list with the given name

```
00114                                       {
00115     for( Assigns as = global_assigns; as; as = as->next ) {
00116         if( strcmp( name, as->assign->name ) == 0 ) {
00117             return as->assign;
00118         }
00119     }
00120     return NULL;
00121 }
```

#### 5.6.2.3 Assign_name()

```
char* Assign_name (
            Assign a )
```

return the name of the assignment pair

```
00095                                {
00096     return a->name;
00097 }
```

#### 5.6.2.4 Assign_new()

```
Assign Assign_new (
            char * name,
            char * value )
```

create new assignment pair

```
00083                                                {
00084     Assign result = malloc( sizeof( *result ) );
00085     result->name = strdup( name );
00086     result->value = strdup( value );
00087     return result;
00088 }
```

Referenced by Assign_add().

#### 5.6.2.5 Assign_value()

```
char* Assign_value (
            Assign a )
```

return the value of the assignment pair

```
00091                                {
00092     return a->value;
00093 }
```

#### 5.6.2.6 Assigns_dump()

```
void Assigns_dump (
            Assigns as )
```

dump the contents of the global assignment list

```
00124                                 {
00125     for( ; as; as = as->next ) {
00126         printf( "%s <- %s\n", as->assign->name, as->assign->value );
00127     }
00128 }
```

#### 5.6.2.7 Assigns_new()

```
Assigns Assigns_new (
            Assign assign,
            Assigns next )
```

create a new assignment list

```
00100                                                  {
00101     Assigns result = malloc( sizeof( *result ) );
00102     result->assign = assign;
00103     result->next = next;
00104     return result;
00105 }
```

Referenced by Assign_add().

# Chapter 6

# Data Structure Documentation

## 6.1 _Assign Struct Reference

### Data Fields

- char * **name**
- char * **value**

### 6.1.1 Detailed Description

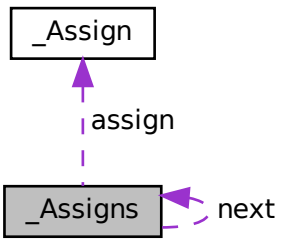
single assignment of a value to string

The documentation for this struct was generated from the following file:

- lib.c

## 6.2 _Assigns Struct Reference

Collaboration diagram for _Assigns:



### Data Fields

- Assign **assign**
- Assigns **next**

### 6.2.1 Detailed Description

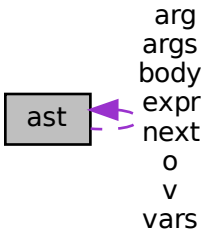
list of assignment of a value to string

The documentation for this struct was generated from the following file:

- lib.c

## 6.3 ast Struct Reference

Collaboration diagram for ast:




### Data Fields

- int tag

  *descriminator for the union, tags start with AST_*

- union {
  struct {
  char * v
  *string value owned by the syntax tree*
  } str
  *string node*
  struct {
  char * v
  *id value owned by the syntax tree*
  } id
  *id node*
  struct {
  struct ast * o

*method target*
char * sel
*selector*
struct ast * arg
*list of arguments*
} unary
*unary method call node*
struct {
struct ast * v
*argument value node*
struct ast * next
*next argument*
} **arg**
struct **argdef** {
const char * key
const char * name
*parameter name*
struct ast * next
*next keyword in the list*
} **argdef**
struct {
struct ast * **v**
struct ast * **next**
} **stmt**
struct {
char * **var**
struct ast * **expr**
} **asgn**
struct {
char * **name**
char * **super**
int **num**
struct ast * **vars**
struct ast * **next**
} **cls**
struct {
char * **v**
struct ast * **next**
} **names**
struct {
const char * **name**
struct ast * **args**
char * **classname**
char * **src**
struct ast * **body**
struct ast * **next**
} **methods**
} **u**

## 6.3.1 Field Documentation

### 6.3.1.1 key

```
const char* ast::key
```

Keyword including the colon at the end if it is no keyword then the plain unary or binary name is here.

#### 6.3.1.2 next

```
struct ast* ast::next
```

next argument

next keyword in the list

#### 6.3.1.3 v

```
char* ast::v
```

string value owned by the syntax tree

id value owned by the syntax tree

The documentation for this struct was generated from the following file:

- global.h

## 6.4 classinfo Struct Reference

### Data Fields

- bool **meta**
- char * **name**
- char * **super**
- int **num**

### 6.4.1 Detailed Description

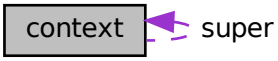
details of a class

The documentation for this struct was generated from the following file:

- lib.c

## 6.5 context Struct Reference

Collaboration diagram for context:



### Data Fields

- struct context * **super**
- bool **ctx_class**
- const char * **name**

The documentation for this struct was generated from the following file:

- lib.h

## 6.6 contextdef Struct Reference

### Data Fields

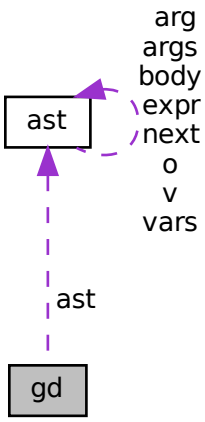
- bool **global**
- bool **instance**
- bool **local**

The documentation for this struct was generated from the following file:

- lib.h

## 6.7 gd Struct Reference

Collaboration diagram for gd:



### Data Fields

- int **state**
- int **paridx**
- int **token**
- int **pos**
- char **buf** [50]
- char **line** [2000]
- struct ast * **ast**
- int **classnum**

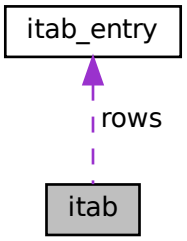
The documentation for this struct was generated from the following file:

- global.h

## 6.8 itab Struct Reference

structure of itab

Collaboration diagram for itab:




### Data Fields

- int **total**
- int **used**
- struct itab_entry * **rows**

### 6.8.1 Detailed Description

structure of itab

The documentation for this struct was generated from the following file:

- lib.c

## 6.9 itab_entry Struct Reference

structure of an entry in the itab.

### Data Fields

- const char * **key**
- void * **value**

### 6.9.1 Detailed Description

structure of an entry in the itab.

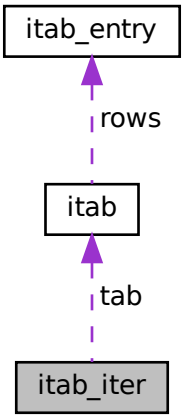
The documentation for this struct was generated from the following file:

- lib.c

## 6.10 itab_iter Struct Reference

iterator over elements of an itab.

Collaboration diagram for itab_iter:




### Data Fields

- struct itab * **tab**
- int **pos**

### 6.10.1 Detailed Description

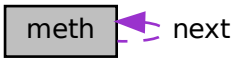
iterator over elements of an itab.

The documentation for this struct was generated from the following file:

- lib.c

## 6.11 meth Struct Reference

Collaboration diagram for meth:




### Data Fields

- char * **name**
- struct meth * **next**

The documentation for this struct was generated from the following file:

- lib.c

## 6.12 methodinfo Struct Reference

### Data Fields

- char * **classname**
- char * **name**

### 6.12.1 Detailed Description

details of a method

The documentation for this struct was generated from the following file:

- lib.c

## 6.13 s_names Struct Reference

Collaboration diagram for s_names:


### Data Fields

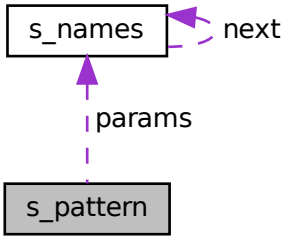
- char * **name**
- t_names **next**

The documentation for this struct was generated from the following file:

- lib.h

## 6.14 s_pattern Struct Reference

Collaboration diagram for s_pattern:




### Data Fields

- char * **selector**
- t_names **params**

The documentation for this struct was generated from the following file:

- lib.h

## 6.15 stringinfo Struct Reference

### Data Fields

- int **num**

### 6.15.1 Detailed Description

details of a string

The documentation for this struct was generated from the following file:

- lib.c

## 6.16 varinfo Struct Reference

### Data Fields

- char * **classname**
- char * **name**

### 6.16.1 Detailed Description

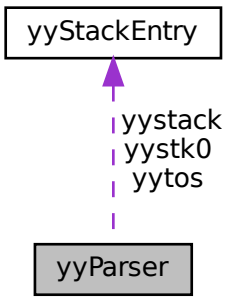
details of a global variable

The documentation for this struct was generated from the following file:

- lib.c

## 6.17 yyParser Struct Reference

Collaboration diagram for yyParser:



### Data Fields

- yyStackEntry * **yytos**
- int **yyerrcnt**
- ParseARG_SDECL ParseCTX_SDECL int **yystksz**
- yyStackEntry * **yystack**
- yyStackEntry **yystk0**

The documentation for this struct was generated from the following file:

- lempar.c

## 6.18 yyStackEntry Struct Reference

### Data Fields

- YYACTIONTYPE **stateno**
- YYCODETYPE **major**
- YYMINORTYPE **minor**

The documentation for this struct was generated from the following file:

- lempar.c