

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_lines()	10
5.1.2.3 itab_new()	10
5.2 Tokenizer	11
5.2.1 Detailed Description	11
5.2.2 Function Documentation	11
5.2.2.1 is_ident_char()	11
5.2.2.2 nextToken()	12
5.2.2.3 readChar()	14
5.2.2.4 readLine()	14
5.2.2.5 readStringToken()	15
5.2.2.6 src_add()	15
5.2.2.7 src_clear()	16
5.2.2.8 src_dump()	16
5.2.2.9 src_read()	16
5.3 Messages	17
5.3.1 Detailed Description	17
5.4 Syntax Messages	17
5.4.1 Detailed Description	18
5.5 list	18
5.5.1 Detailed Description	18
5.5.2 Function Documentation	18
5.5.2.1 namelist_init()	18
5.6 Internal_structures	18

5.6.1 Detailed Description	19
6 Data Structure Documentation	21
6.1 ast Struct Reference	21
6.1.1 Field Documentation	22
6.1.1.1 key	23
6.1.1.2 next	23
6.1.1.3 v	23
6.2 classinfo Struct Reference	23
6.2.1 Detailed Description	23
6.3 gd Struct Reference	24
6.4 itab Struct Reference	25
6.4.1 Detailed Description	25
6.5 itab_entry Struct Reference	25
6.5.1 Detailed Description	26
6.6 itab_iter Struct Reference	26
6.6.1 Detailed Description	26
6.7 methodinfo Struct Reference	27
6.7.1 Detailed Description	27
6.8 s_assignment Struct Reference	27
6.9 s_block Struct Reference	28
6.10 s_classdef Struct Reference	28
6.11 s_env Struct Reference	29
6.12 s_expression Struct Reference	29
6.13 s_expression_list Struct Reference	30
6.14 s_globals Struct Reference	31
6.15 s_message_cascade Struct Reference	32
6.16 s_message_pattern Struct Reference	32
6.17 s_messages Struct Reference	33
6.18 s_methoddef Struct Reference	34
6.19 s_namelist Struct Reference	34
6.20 s_names Struct Reference	35
6.21 s_object Struct Reference	35
6.22 s_pattern Struct Reference	36
6.23 s_slot Struct Reference	37
6.24 s_statements Struct Reference	38
6.25 stringinfo Struct Reference	38
6.25.1 Detailed Description	38
6.26 varinfo Struct Reference	39
6.26.1 Detailed Description	39
6.27 yyParser Struct Reference	39
6.28 yyStackEntry Struct Reference	40

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
Tokenizer	11
Messages	17
Syntax Messages	17
Name List	??
Internal_structures	18

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ast	21
classinfo	23
gd	24
itab	
Structure of itab	25
itab_entry	
Structure of an entry in the itab	25
itab_iter	
Iterator over elements of an itab	26
methodinfo	27
s_assignment	27
s_block	28
s_classdef	28
s_env	29
s_expression	29
s_expression_list	30
s_globals	31
s_message_cascade	32
s_message_pattern	32
s_messages	33
s_methoddef	34
s_namelist	34
s_names	35
s_object	35
s_pattern	36
s_slot	37
s_statements	38
stringinfo	38
varinfo	39
yyParser	39
yyStackEntry	40

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

- int [itab_lines](#) (struct [itab](#) *[itab](#))
- 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

```
00134                                     {
00135     const struct itab_entry *a = aptr;
00136     const struct itab_entry *b = bptr;
00137     return strcmp( a->key, b->key );
00138 }
```

5.1.2.2 itab_lines()

```
int itab_lines (
    struct itab * itab )
```

returns the number of lines in the table

```
00100                                     {
00101     assert( itab );
00102     return itab->used;
00103 }
```

Referenced by [src_add\(\)](#).

5.1.2.3 itab_new()

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

create a new itab with default parameters.

Returns

reference to an itab structure.

Detailed description follows here.

```
00120     {
00121     struct itab *r = calloc_zero( NULL, struct itab );
00122     r->total = 10;
00123     r->used = 0;
00124     r->rows = calloc_array( r, struct itab_entry, r->total );
00125     return r;
00126 }
```

Referenced by [src_clear\(\)](#).

5.2 Tokenizer

Functions

- bool [is_ident_char](#) (int c)
check if character is part of an identifier.
- bool [is_binary_char](#) (int c)
- bool [src_clear](#) ()
- bool [src_add](#) (const char *line)
- bool [src_read](#) (const char *name)
- bool [src_dump](#) ()
- 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.
- void [parse_verbatim](#) (char c)
- bool [nextToken](#) (void)
read next token.

5.2.1 Detailed Description

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

See also

[nextToken](#).

5.2.2 Function Documentation

5.2.2.1 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 c is an identifier character.

```

00223         {
00224         return isalpha( c ) || isdigit( c ) || c == '_';
00225     }

```

Referenced by [nextToken\(\)](#).

5.2.2.2 nextToken()

```
bool nextToken ( )
```

read next token.

This is a more detailed description.

Returns

true if successful

```

00378         {
00379     char c;
00380     bool result = false;
00381     while( true ) {
00382         while( readChar( &c ) && isspace( c ) );
00383         if( c == '"' ) {
00384             while( readChar( &c ) && c != '"' );
00385         }
00386         else
00387             break;
00388     }
00389     if( gd.state == 1 ) {
00390         if( isalpha( c ) ) {
00391             int idx = 0;
00392             for( ;; ) {
00393                 gd.buf[idx++] = c;
00394                 readChar( &c );
00395                 if( !is_ident_char( c ) )
00396                     break;
00397             }
00398             if( c == ':' ) {
00399                 gd.buf[idx++] = c;
00400                 gd.token = TK_KEYWORD;
00401             }
00402             else {
00403                 gd.pos--;
00404                 gd.token = TK_IDENT;
00405             }
00406             gd.buf[idx] = 0;
00407             result = true;
00408         }
00409         else if( is_binary_char( c ) ) {
00410             for( int idx = 0; is_binary_char( c ); idx++ ) {
00411                 gd.buf[idx] = c;
00412                 gd.buf[idx + 1] = 0;
00413                 readChar( &c );
00414             }
00415             gd.pos--;
00416             gd.token = 0;
00417             gd.token = TK_BINOP;
00418             result = true;
00419             if( strcmp( ":", gd.buf ) == 0 ) {
00420                 gd.token = TK_ASSIGN;
00421                 result = true;
00422             }
00423             else if( strcmp( "<-", gd.buf ) == 0 ) {
00424                 gd.token = TK_LARROW;
00425                 result = true;
00426             }
00427             else if( strcmp( "|", gd.buf ) == 0 ) {
00428                 gd.token = TK_BAR;
00429                 result = true;
00430             }
00431             else if( 0 == strcmp( "<", gd.buf ) ) {
00432                 gd.token = TK_LT;
00433                 result = true;
00434             }

```



```

00435         else if( 0 == strcmp( ">", gd.buf ) ) {
00436             gd.token = TK_GT;
00437             result = true;
00438         }
00439     }
00440     else if( isdigit( c ) ) {
00441         int idx = 0;
00442         while( isdigit( c ) ) {
00443             printf("### digit %c\n", c);
00444             gd.buf[idx++] = c;
00445             gd.buf[idx] = 0;
00446             readChar( &c );
00447         }
00448         gd.pos--;
00449         gd.token = TK_NUMBER;
00450         result = true;
00451     }
00452     else {
00453         switch ( c ) {
00454             case '\\':
00455                 result = readStringToken( );
00456                 break;
00457             case '.':
00458                 result = true;
00459                 gd.token = TK_DOT;
00460                 break;
00461             case ';':
00462                 result = true;
00463                 gd.token = TK_SEMICOLON;
00464                 break;
00465             case '(':
00466                 result = true;
00467                 gd.token = TK_LPAREN;
00468                 break;
00469             case ')':
00470                 result = true;
00471                 gd.token = TK_RPAREN;
00472                 break;
00473             case '[':
00474                 result = true;
00475                 gd.token = TK_LBRACK;
00476                 break;
00477             case ']':
00478                 result = true;
00479                 gd.token = TK_RBRACK;
00480                 break;
00481             case '{':
00482                 result = true;
00483                 gd.token = TK_LBRACE;
00484                 break;
00485             case '}':
00486                 result = true;
00487                 gd.token = TK_RBRACE;
00488                 break;
00489             case '#':
00490                 readChar( &c );
00491                 for( int idx = 0; is_ident_char( c ) || c == ':'; idx++ ) {
00492                     gd.buf[idx] = c;
00493                     gd.buf[idx + 1] = 0;
00494                     readChar( &c );
00495                 }
00496                 gd.pos--;
00497                 gd.token = TK_SYMBOL;
00498                 result = true;
00499                 break;
00500             case '^':
00501                 result = true;
00502                 gd.token = TK_UARROW;
00503                 break;
00504             case ':':
00505                 result = true;
00506                 gd.token = TK_COLON;
00507                 readChar( &c );
00508                 if( c == '=' ) {
00509                     gd.token = TK_ASSIGN;
00510                 }
00511                 else
00512                     gd.pos--;
00513                 break;
00514             case '$':
00515                 result = true;
00516                 gd.token = TK_CHAR;
00517                 readChar( &c );
00518                 gd.buf[0] = c;
00519                 gd.buf[1] = 0;
00520                 break;
00521             default:

```

```

00522             gd.pos--;
00523             break;
00524         }
00525     }
00526 }
00527 return result;
00528 }

```

References [is_ident_char\(\)](#), and [readChar\(\)](#).

5.2.2.3 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

```

00331     {
00332     bool result = true;
00333     if( gd.state == 0 ) {
00334         result = readLine( );
00335     }
00336     if( result ) {
00337         *t = gd.line[gd.pos++];
00338         while( *t == 0 ) {
00339             if( readLine( ) ) {
00340                 *t = gd.line[gd.pos++];
00341             }
00342             else {
00343                 result = false;
00344                 break;
00345             }
00346         }
00347     }
00348     return result;
00349 }

```

References [readLine\(\)](#).

Referenced by [nextToken\(\)](#), and [readStringToken\(\)](#).

5.2.2.4 readLine()

```

bool readLine ( )

```

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

trailing blanks are removed.

```

00308     {
00309     if( gd.src_iter == NULL ) {
00310         gd.src_iter = itab_foreach( gd.src );

```

```

00311     }
00312     else {
00313         gd.src_iter = itab_next( gd.src_iter );
00314     }
00315     if( gd.src_iter ) {
00316         gd.line = itab_value( gd.src_iter );
00317         gd.line_count++;
00318         printf( "%2d:%s\n", gd.line_count, gd.line );
00319         gd.pos = 0;
00320         gd.state = 1;
00321         return true;
00322     }
00323     else {
00324         gd.line = "";
00325         gd.state = 2;
00326         return false;
00327     }
00328 }

```

Referenced by [readChar\(\)](#).

5.2.2.5 readStringToken()

```

bool readStringToken (
    void )

```

read string token.

Returns

true if successful

```

00351     {
00352         int idx = 0;
00353         char c;
00354         while( readChar( &c ) && '\n' != c ) {
00355             if( c == '\\\ ' )
00356                 readChar( &c );
00357             gd.buf[idx++] = c;
00358         }
00359         gd.buf[idx] = 0;
00360         gd.token = TK_STRING;
00361         return true;
00362     }

```

References [readChar\(\)](#).

5.2.2.6 src_add()

```

bool src_add (
    const char * )

```

adding one line to the source that will be parsed.

```

00264     {
00265         int n = itab_lines( gd.src );
00266         char buf[10];
00267         sprintf( buf, "%09d", n + 1 );
00268         itab_append( gd.src, buf, talloc_strdup( gd.src, line ) );
00269     }

```

References [itab_lines\(\)](#).

5.2.2.7 src_clear()

```
bool src_clear ( )
```

clear and initialize the source that will alter be parsed.

needs to be called before using *src_add*. *src_read* will do it automatically.

```
00253     {
00254     if( gd.src ) {
00255         talloc_free( gd.src );
00256     }
00257     gd.src = itab_new( );
00258     if( gd.src_iter ) {
00259         talloc_free( gd.src_iter );
00260     }
00261     gd.src_iter = NULL;
00262 }
```

References [itab_new\(\)](#).

Referenced by [src_read\(\)](#).

5.2.2.8 src_dump()

```
bool src_dump ( )
```

dumps all the lines of the current source.

```
00295     {
00296     for( struct itab_iter * x = itab_foreach( gd.src );
00297         x; x = itab_next( x ) ) {
00298         printf( "%s:%s\n", itab_key( x ), itab_value( x ) );
00299     }
00300 }
```

5.2.2.9 src_read()

```
bool src_read (
    const char * name )
```

read file into itab.

read a file into src itab.

```
00274     {
00275     FILE *f = fopen( name, "r" );
00276     char buf[1000];
00277     char *line;
00278     int line_no = 1;
00279     src_clear( );
00280     for( ;; ) {
00281         line = fgets( buf, sizeof( buf ), f );
00282         if( line == NULL )
00283             break;
00284         int n = strlen( line );
00285         while( n > 0 && isspace( line[--n] ) )
00286             line[n] = 0;
00287         char line_number[10];
00288         sprintf( line_number, "%09d", line_no );
00289         itab_append( gd.src, line_number, talloc_strdup( gd.src, line ) );
00290         line_no++;
00291     }
00292     fclose( f );
00293 }
```

References [src_clear\(\)](#).

5.3 Messages

Data Structures

- struct **s_msgs**

Macros

- `#define MSG_LOG_LEN 200`

Typedefs

- typedef char **t_msg**[200]

Functions

- void **msg_init** ()
- void **msg_add** (const char *msg,...)
- void **msg_print_last** ()

5.3.1 Detailed Description

5.4 Syntax Messages

Functions

- void **message_add_msg** ([t_messages](#) *ms, [t_messages](#) *m)

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**
- 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**
- int **s_msgs::size**
- int **s_msgs::pos**
- [t_msg](#) **s_msgs::msgs** [MSG_LOG_LEN]

5.4.1 Detailed Description

5.5 Name List

Functions

- void `namelist_init` (`t_namelist *nl`)
clear the structure for further usage.
- void `namelist_add` (`t_namelist *nl`, `const char *name`)
- void `namelist_copy` (`t_namelist *to`, `t_namelist *from`)

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 `namelist_add()`

```
void namelist_add (
    t_namelist * nl,
    const char * name )
```

adding a name to the name list. Memory will be allocated by the name list and also the name will be copied. The paramter can safely being freed after this call.

Parameters

<i>nl</i>	the modified list
<i>name</i>	the string to be added

```
00023                                     {
00024     nl->count++;
00025     nl->names = talloc_realloc( NULL, nl->names, char *, nl->count );
00026     nl->names[nl->count - 1] = talloc_strdup( nl->names, name );
00027 }
```

5.5.2.2 `namelist_copy()`

```
void namelist_copy (
    t_namelist * to,
    t_namelist * from )
```

make a deep copy of a name list

Parameters

<i>to</i>	the target name list, which doesn't need to be initialized
<i>from</i>	the source to be copied.

```

00032                                     {
00033     to->count = from->count;
00034     to->names = talloc_array( NULL, char *, to->count );
00035     assert(talloc_get_type(from->names, char *));
00036     for( int i = 0; i < to->count; i++ ) {
00037         to->names[i] = talloc_strdup( to->names, from->names[i] );
00038     }
00039     assert(talloc_get_type(to->names, char *));
00040 }

```

5.5.2.3 namelist_init()

```

void namelist_init (
    t_namelist * nl )

```

clear the structure for further usage.

The *namelist* itself is not allocated but could be part of an already allocated structure.

Parameters

<i>nl</i>	reference to an existing structure to be initialized.
-----------	---

```

00013                                     {
00014     nl->count = 0;
00015     nl->names = NULL;
00016 }

```

5.6 Internal_structures

Data Structures

- struct [s_namelist](#)
- struct [s_expression_list](#)
- struct [s_names](#)
- struct [s_pattern](#)
- struct [s_classdef](#)
- struct [s_statements](#)
- struct [s_methoddef](#)
- struct [s_message_pattern](#)
- struct [s_assignment](#)
- struct [s_block](#)
- struct [s_expression](#)
- struct [s_messages](#)
- struct [s_message_cascade](#)
- struct [s_object](#)
- struct [s_slot](#)
- struct [s_env](#)

Typedefs

- typedef struct [s_namelist](#) [t_namelist](#)
- typedef struct [s_names](#) * [t_names](#)
- typedef struct [s_expression_list](#) [t_expression_list](#)
- typedef struct [s_pattern](#) * [t_pattern](#)
- typedef struct [s_classdef](#) [t_classdef](#)
- typedef enum [e_statement_type](#) [t_statement_type](#)
- typedef struct [s_statements](#) [t_statements](#)
- typedef struct [s_methoddef](#) [t_methoddef](#)
- typedef struct [s_message_pattern](#) [t_message_pattern](#)
- typedef enum [e_expression_tag](#) [t_expression_tag](#)
- typedef struct [s_assignment](#) [t_assignment](#)
- typedef struct [s_block](#) [t_block](#)
- typedef struct [s_expression](#) [t_expression](#)
- typedef struct [s_messages](#) [t_messages](#)
- typedef struct [s_message_cascade](#) [t_message_cascade](#)
- typedef struct [s_object](#) *(* [t_message_handler](#)) (struct [s_object](#) *, const char *sel, struct [s_object](#) **args)
- typedef struct [s_object](#) [t_object](#)
- typedef struct [s_slot](#) [t_slot](#)
- typedef struct [s_env](#) [t_env](#)

Enumerations

- enum [e_statement_type](#) { [stmt_return](#) = 100 , [stmt_assign](#) , [stmt_message](#) }
- enum [e_expression_tag](#) {
[tag_string](#) , [tag_char](#) , [tag_message](#) , [tag_number](#) ,
[tag_ident](#) , [tag_block](#) , [tag_array](#) , [tag_assignment](#) }

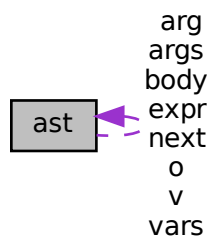
5.6.1 Detailed Description

Chapter 6

Data Structure Documentation

6.1 ast Struct Reference

Collaboration diagram for ast:



Data Fields

- int `tag`
discriminator 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

```

6.1.1 Field Documentation

6.1.1.1 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.

6.1.1.2 next

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

next argument

next keyword in the list

6.1.1.3 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

6.2 classinfo Struct Reference

Data Fields

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

6.2.1 Detailed Description

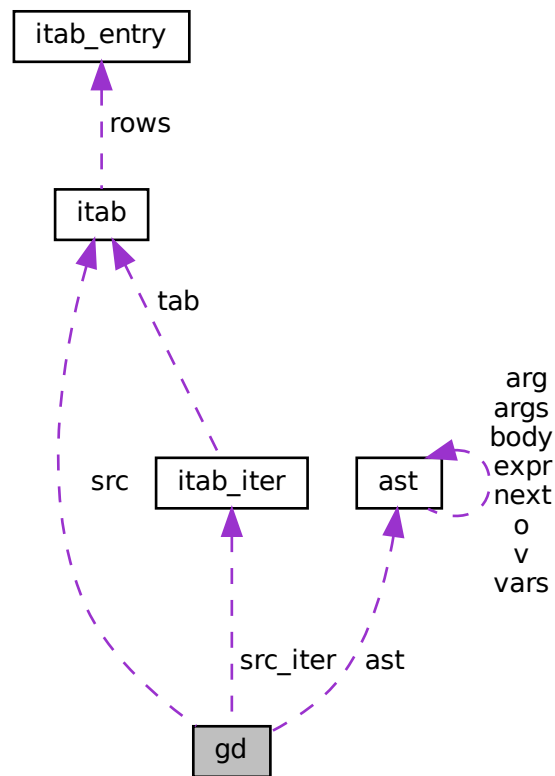
details of a class

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

- lib.c

6.3 gd Struct Reference

Collaboration diagram for gd:



Data Fields

- int **state**
- int **paridx**
- int **token**
- int **pos**
- char **buf** [50]
- char * **line**
- int **line_count**
- struct [ast](#) * **ast**
- int **classnum**
- struct [itab](#) * **src**
- struct [itab_iter](#) * **src_iter**

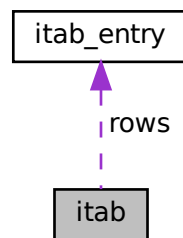
The documentation for this struct was generated from the following file:

- `global.h`

6.4 itab Struct Reference

structure of itab

Collaboration diagram for itab:



Data Fields

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

6.4.1 Detailed Description

structure of itab

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

- `lib.c`

6.5 itab_entry Struct Reference

structure of an entry in the itab.

Data Fields

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

6.5.1 Detailed Description

structure of an entry in the itab.

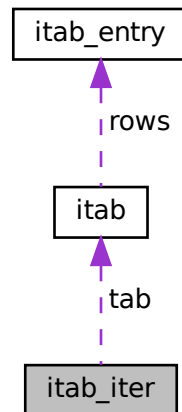
The documentation for this struct was generated from the following file:

- lib.c

6.6 itab_iter Struct Reference

iterator over elements of an itab.

Collaboration diagram for itab_iter:



Data Fields

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

6.6.1 Detailed Description

iterator over elements of an itab.

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

- lib.c

6.7 methodinfo Struct Reference

Data Fields

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

6.7.1 Detailed Description

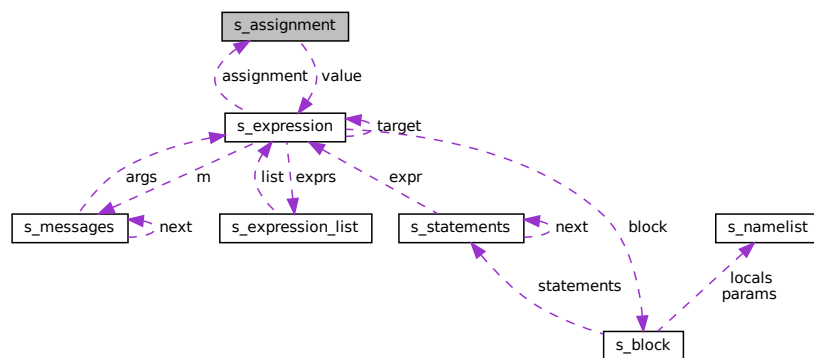
details of a method

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

- lib.c

6.8 s_assignment Struct Reference

Collaboration diagram for s_assignment:



Data Fields

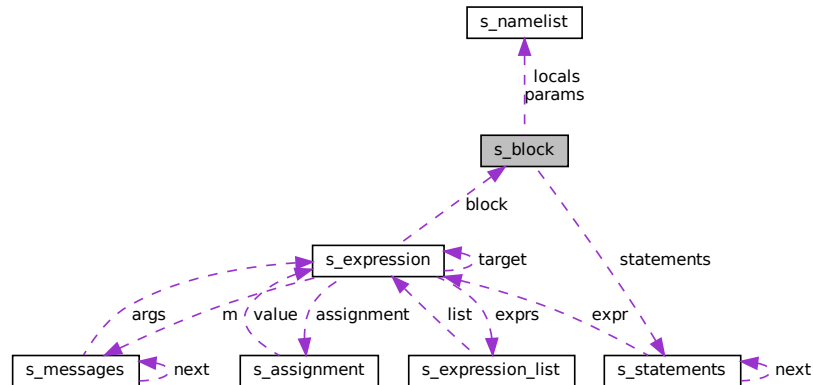
- const char * **target**
- struct `s_expression` * **value**

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

- lib.h

6.9 s_block Struct Reference

Collaboration diagram for s_block:



Data Fields

- `t_namelist` **params**
- `t_namelist` **locals**
- `t_statements` * **statements**

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

- `lib.h`

6.10 s_classdef Struct Reference

Data Fields

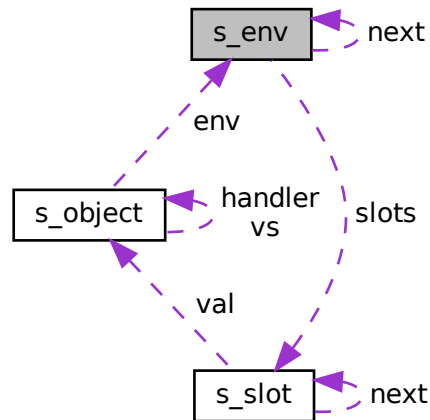
- `int` **id**
- `char *` **name**
- `char *` **meta**
- `char *` **super**

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

- `lib.h`

6.11 s_env Struct Reference

Collaboration diagram for s_env:



Data Fields

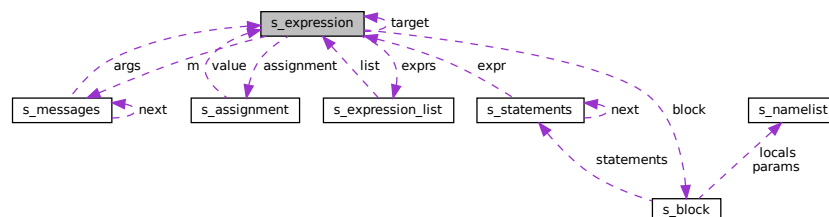
- `t_slot` * `slots`
- struct `s_env` * `next`

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

- `lib.h`

6.12 s_expression Struct Reference

Collaboration diagram for s_expression:



Data Fields

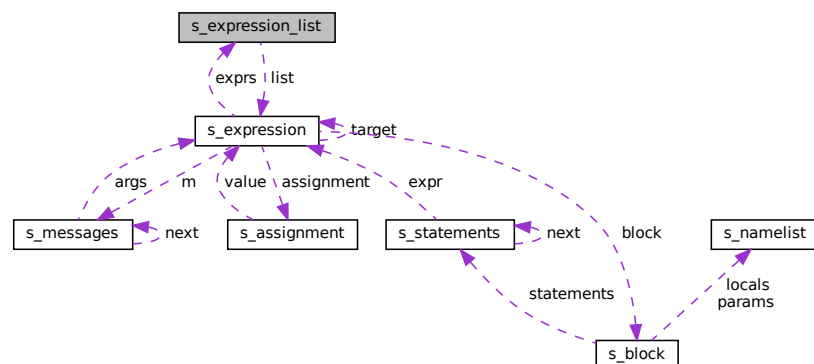
- `t_expression_tag` **tag**
- - union {
 - int **intvalue**
 - const char * **strvalue**
 - const char * **ident**
 - `t_expression_list` **exprs**
 - struct **msg** {
 - struct `s_expression` * **target**
 - struct `s_messages` * **m**
 - msg**
 - `t_assignment` **assignment**
 - `t_block` **block**
 - } **u**

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

- `lib.h`

6.13 `s_expression_list` Struct Reference

Collaboration diagram for `s_expression_list`:



Data Fields

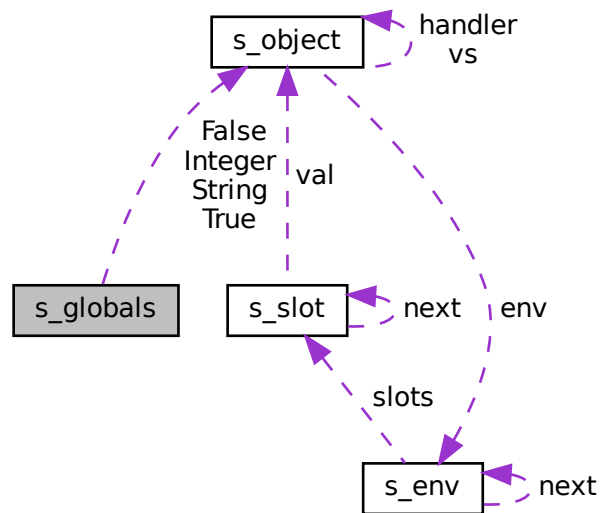
- int **count**
- struct `s_expression` ** **list**

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

- `lib.h`

6.14 s_globals Struct Reference

Collaboration diagram for s_globals:



Data Fields

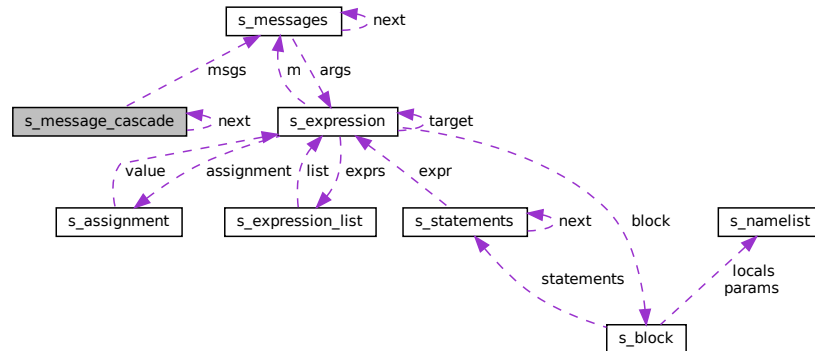
- `t_object` * **String**
- `t_object` * **Integer**
- `t_object` * **True**
- `t_object` * **False**

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

- `tt_test.c`

6.15 s_message_cascade Struct Reference

Collaboration diagram for s_message_cascade:



Data Fields

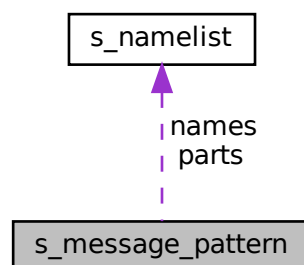
- [t_messages](#) * **msgs**
- struct [s_message_cascade](#) * **next**

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

- lib.h

6.16 s_message_pattern Struct Reference

Collaboration diagram for s_message_pattern:



Data Fields

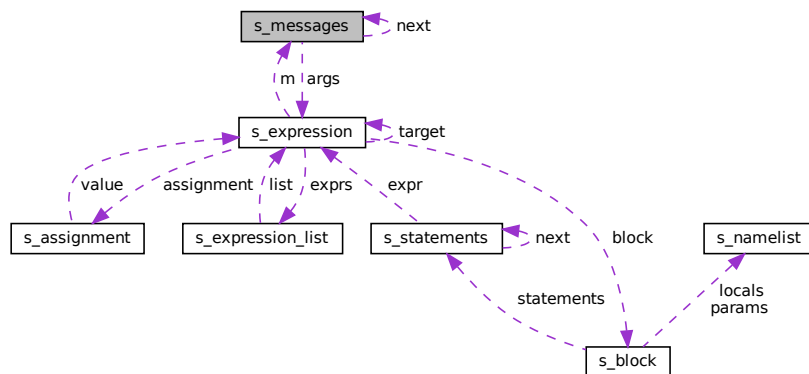
- [t_namelist](#) **parts**
- [t_namelist](#) **names**

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

- lib.h

6.17 s_messages Struct Reference

Collaboration diagram for s_messages:



Data Fields

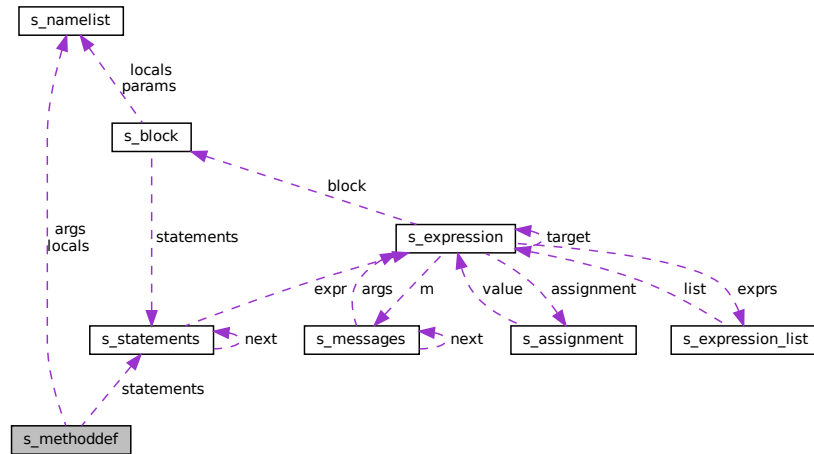
- bool **cascaded**
- char * **sel**
- int **argc**
- [t_expression](#) ** **args**
- struct [s_messages](#) * **next**

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

- lib.h

6.18 s_methoddef Struct Reference

Collaboration diagram for s_methoddef:



Data Fields

- char * **sel**
- [t_namelist](#) **args**
- [t_namelist](#) **locals**
- [t_statements](#) * **statements**

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

- lib.h

6.19 s_namelist Struct Reference

Data Fields

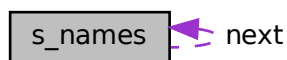
- int **count**
- char ** **names**

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

- lib.h

6.20 s_names Struct Reference

Collaboration diagram for s_names:



Data Fields

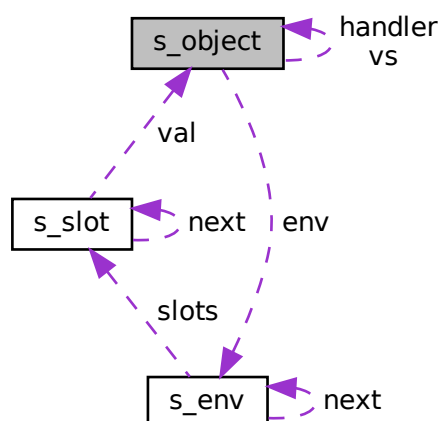
- `char * name`
- `t_names next`

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

- `lib.h`

6.21 s_object Struct Reference

Collaboration diagram for s_object:



Data Fields

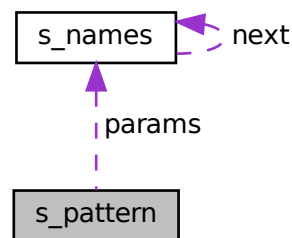
- `t_message_handler` **handler**
- ```
union {
 void * data
 int intval
 struct {
 int i [10]
 void * p [10]
 } vals
 struct {
 struct s_object ** vs
 int cnt
 } vars
} u
```
- `struct s_env * env`

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

- `lib.h`

## 6.22 s\_pattern Struct Reference

Collaboration diagram for `s_pattern`:



## Data Fields

- `char * selector`
- `t_names params`

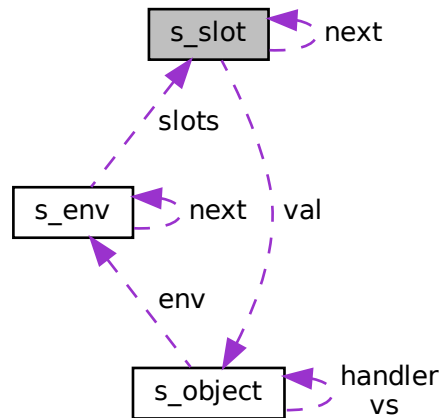
The documentation for this struct was generated from the following file:

- `lib.h`



## 6.23 s\_slot Struct Reference

Collaboration diagram for s\_slot:



### Data Fields

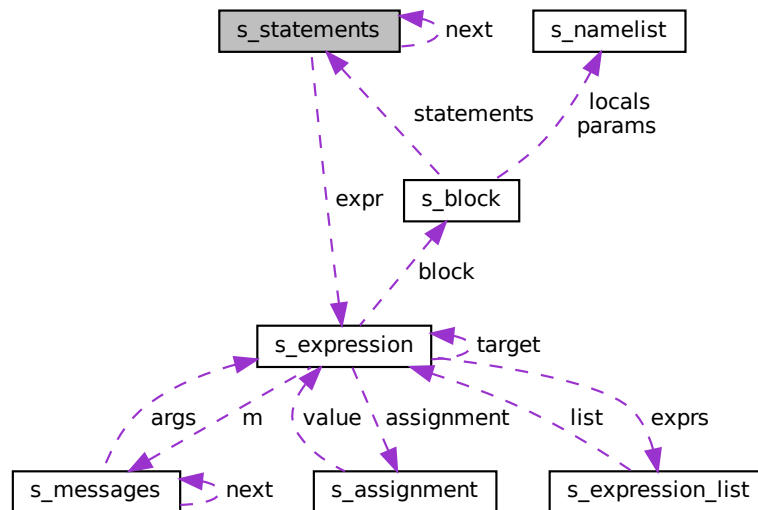
- `const char * name`
- `t_object * val`
- `struct s_slot * next`

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

- `lib.h`

## 6.24 s\_statements Struct Reference

Collaboration diagram for s\_statements:



### Data Fields

- `t_statement_type` **type**
- `struct s_expression *` **expr**
- `struct s_statements *` **next**

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

- `lib.h`

## 6.25 stringinfo Struct Reference

### Data Fields

- `int` **num**

### 6.25.1 Detailed Description

details of a string

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

- `lib.c`

## 6.26 varinfo Struct Reference

## Data Fields

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

### 6.26.1 Detailed Description

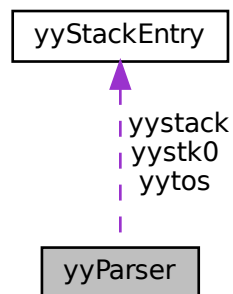
details of a global variable

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

- lib.c

## 6.27 yyParser Struct Reference

Collaboration diagram for yyParser:



## Data Fields

- `yyStackEntry * yytos`
- `int yyerrcnt`
- `ParseARG_SDECL ParseCTX_SDECL int yytsksz`
- `yyStackEntry * yyystack`
- `yyStackEntry yyystk0`

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

- lempar.c

## 6.28 yyStackEntry Struct Reference

### Data Fields

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

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

- lempar.c