

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 list | 9 |
| 5.1.1 Detailed Description | 9 |
| 5.2 ITab | 9 |
| 5.2.1 Detailed Description | 10 |
| 5.2.2 Function Documentation | 10 |
| 5.2.2.1 itab_entry_cmp() | 10 |
| 5.2.2.2 itab_lines() | 10 |
| 5.2.2.3 itab_new() | 10 |
| 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() | 14 |
| 5.3.2.4 readLine() | 14 |
| 5.3.2.5 readStringToken() | 15 |
| 5.3.2.6 src_add() | 15 |
| 5.3.2.7 src_clear() | 16 |
| 5.3.2.8 src_dump() | 16 |
| 5.3.2.9 src_read() | 16 |
| 5.4 Messages | 17 |
| 5.4.1 Detailed Description | 17 |
| 5.5 Syntax Messages | 17 |
| 5.5.1 Detailed Description | 18 |
| 5.6 Internal_structures | 18 |
| 5.6.1 Detailed Description | 18 |

| | |
|---|-----------|
| 6 Data Structure Documentation | 19 |
| 6.1 ast Struct Reference | 19 |
| 6.1.1 Field Documentation | 20 |
| 6.1.1.1 key | 21 |
| 6.1.1.2 next | 21 |
| 6.1.1.3 v | 21 |
| 6.2 classinfo Struct Reference | 21 |
| 6.2.1 Detailed Description | 21 |
| 6.3 gd Struct Reference | 22 |
| 6.4 itab Struct Reference | 23 |
| 6.4.1 Detailed Description | 23 |
| 6.5 itab_entry Struct Reference | 23 |
| 6.5.1 Detailed Description | 24 |
| 6.6 itab_iter Struct Reference | 24 |
| 6.6.1 Detailed Description | 24 |
| 6.7 methodinfo Struct Reference | 25 |
| 6.7.1 Detailed Description | 25 |
| 6.8 s_block Struct Reference | 25 |
| 6.9 s_classdef Struct Reference | 26 |
| 6.10 s_env Struct Reference | 26 |
| 6.11 s_expression Struct Reference | 27 |
| 6.12 s_expression_list Struct Reference | 28 |
| 6.13 s_globals Struct Reference | 28 |
| 6.14 s_message_cascade Struct Reference | 29 |
| 6.15 s_message_pattern Struct Reference | 30 |
| 6.16 s_messages Struct Reference | 30 |
| 6.17 s_methoddef Struct Reference | 31 |
| 6.18 s_namelist Struct Reference | 32 |
| 6.19 s_names Struct Reference | 32 |
| 6.20 s_object Struct Reference | 32 |
| 6.21 s_pattern Struct Reference | 33 |
| 6.22 s_statements Struct Reference | 34 |
| 6.23 stringinfo Struct Reference | 34 |
| 6.23.1 Detailed Description | 34 |
| 6.24 varinfo Struct Reference | 35 |
| 6.24.1 Detailed Description | 35 |
| 6.25 yyParser Struct Reference | 35 |
| 6.26 yyStackEntry Struct Reference | 36 |

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:

| | |
|-------------------------------|----|
| list | 9 |
| ITab | 9 |
| Tokenizer | 11 |
| Messages | 17 |
| Syntax Messages | 17 |
| Internal_structures | 18 |

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

| | |
|-----------------------------------|----|
| ast | 19 |
| classinfo | 21 |
| gd | 22 |
| itab | |
| Structure of itab | 23 |
| itab_entry | |
| Structure of an entry in the itab | 23 |
| itab_iter | |
| Iterator over elements of an itab | 24 |
| methodinfo | 25 |
| s_block | 25 |
| s_classdef | 26 |
| s_env | 26 |
| s_expression | 27 |
| s_expression_list | 28 |
| s_globals | 28 |
| s_message_cascade | 29 |
| s_message_pattern | 30 |
| s_messages | 30 |
| s_methoddef | 31 |
| s_namelist | 32 |
| s_names | 32 |
| s_object | 32 |
| s_pattern | 33 |
| s_statements | 34 |
| stringinfo | 34 |
| varinfo | 35 |
| yyParser | 35 |
| yyStackEntry | 36 |

Chapter 5

Module Documentation

5.1 list

Functions

- void **namelist_init** ([t_namelist](#) *nl)
- void **namelist_add** ([t_namelist](#) *nl, const char *name)
- void **namelist_copy** ([t_namelist](#) *to, [t_namelist](#) *from)

5.1.1 Detailed Description

5.2 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)

5.2.1 Detailed Description

sorted list of structures -> tables with primary index

5.2.2 Function Documentation

5.2.2.1 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

```
00159                                     {
00160     const struct itab_entry *a = aptr;
00161     const struct itab_entry *b = bptr;
00162     return strcmp( a->key, b->key );
00163 }
```

5.2.2.2 itab_lines()

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

returns the number of lines in the table

```
00125                                     {
00126     assert( itab );
00127     return itab->used;
00128 }
```

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

5.2.2.3 itab_new()

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

create a new itab with default parameters.

Returns

reference to an itab structure.

Detailed description follows here.

```
00145     {
00146     struct itab *r = calloc_zero( NULL, struct itab );
00147     r->total = 10;
00148     r->used = 0;
00149     r->rows = calloc_array( r, struct itab_entry, r->total );
00150     return r;
00151 }
```

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

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 [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.

5.3.1 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

| | | |
|-----------------|----------------|------------------------|
| <code>in</code> | <code>c</code> | character to classify. |
|-----------------|----------------|------------------------|

Returns

true if c is an identifier character.

```

00248         {
00249     return isalpha( c ) || isdigit( c ) || c == '_';
00250 }

```

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_ASSIGN;
00446                 result = true;
00447             }
00448             else if( strcmp( "<-", gd.buf ) == 0 ) {
00449                 gd.token = TK_LARROW;
00450                 result = true;
00451             }
00452             else if( strcmp( "|", gd.buf ) == 0 ) {
00453                 gd.token = TK_BAR;
00454                 result = true;
00455             }
00456             else if( 0 == strcmp( "<", gd.buf ) ) {
00457                 gd.token = TK_LT;
00458                 result = true;
00459             }

```



```

00460         else if( 0 == strcmp( ">", gd.buf ) ) {
00461             gd.token = TK_GT;
00462             result = true;
00463         }
00464     }
00465     else if( isdigit( c ) ) {
00466         int idx = 1;
00467         while( isdigit( c ) ) {
00468             gd.buf[idx - 1] = c;
00469             gd.buf[idx] = 0;
00470             readChar( &c );
00471         }
00472         gd.pos--;
00473         gd.token = TK_NUMBER;
00474         result = true;
00475     }
00476     else {
00477         switch ( c ) {
00478             case '\\':
00479                 result = readStringToken( );
00480                 break;
00481             case '.':
00482                 result = true;
00483                 gd.token = TK_DOT;
00484                 break;
00485             case ';':
00486                 result = true;
00487                 gd.token = TK_SEMICOLON;
00488                 break;
00489             case '(':
00490                 result = true;
00491                 gd.token = TK_LPAREN;
00492                 break;
00493             case ')':
00494                 result = true;
00495                 gd.token = TK_RPAREN;
00496                 break;
00497             case '[':
00498                 result = true;
00499                 gd.token = TK_LBRACK;
00500                 break;
00501             case ']':
00502                 result = true;
00503                 gd.token = TK_RBRACK;
00504                 break;
00505             case '{':
00506                 result = true;
00507                 gd.token = TK_LBRACE;
00508                 break;
00509             case '}':
00510                 result = true;
00511                 gd.token = TK_RBRACE;
00512                 break;
00513             case '#':
00514                 readChar( &c );
00515                 for( int idx = 0; is_ident_char( c ) || c == ':'; idx++ ) {
00516                     gd.buf[idx] = c;
00517                     gd.buf[idx + 1] = 0;
00518                     readChar( &c );
00519                 }
00520                 gd.pos--;
00521                 gd.token = TK_SYMBOL;
00522                 result = true;
00523                 break;
00524             case '^':
00525                 result = true;
00526                 gd.token = TK_UARROW;
00527                 break;
00528             case ':':
00529                 result = true;
00530                 gd.token = TK_COLON;
00531                 readChar( &c );
00532                 if( c == '=' ) {
00533                     gd.token = TK_ASSIGN;
00534                 }
00535                 else
00536                     gd.pos--;
00537                 break;
00538             case '$':
00539                 result = true;
00540                 gd.token = TK_CHAR;
00541                 readChar( &c );
00542                 gd.buf[0] = c;
00543                 gd.buf[1] = 0;
00544                 break;
00545             default:
00546                 gd.pos--;

```

```

00547             break;
00548         }
00549     }
00550 }
00551 return result;
00552 }

```

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

```

00356     {
00357     bool result = true;
00358     if( gd.state == 0 ) {
00359         result = readLine( );
00360     }
00361     if( result ) {
00362         *t = gd.line[gd.pos++];
00363         while( *t == 0 ) {
00364             if( readLine( ) ) {
00365                 *t = gd.line[gd.pos++];
00366             }
00367             else {
00368                 result = false;
00369                 break;
00370             }
00371         }
00372     }
00373     return result;
00374 }

```

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.

```

00333     {
00334     if( gd.src_iter == NULL ) {
00335         gd.src_iter = itab_foreach( gd.src );
00336     }

```

```

00337     else {
00338         gd.src_iter = itab_next( gd.src_iter );
00339     }
00340     if( gd.src_iter ) {
00341         gd.line = itab_value( gd.src_iter );
00342         gd.line_count++;
00343         printf( "%2d:%s\n", gd.line_count, gd.line );
00344         gd.pos = 0;
00345         gd.state = 1;
00346         return true;
00347     }
00348     else {
00349         gd.line = "";
00350         gd.state = 2;
00351         return false;
00352     }
00353 }

```

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

5.3.2.5 readStringToken()

```

bool readStringToken (
    void )

```

read string token.

Returns

true if successful

```

00376     {
00377         int idx = 0;
00378         char c;
00379         while( readChar( &c ) && '"' != c ) {
00380             if( c == '\\\\' )
00381                 readChar( &c );
00382             gd.buf[idx++] = c;
00383         }
00384         gd.buf[idx] = 0;
00385         gd.token = TK_STRING;
00386         return true;
00387     }

```

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

5.3.2.6 src_add()

```

bool src_add (
    const char * )

```

adding one line to the source that will be parsed.

```

00289     {
00290         int n = itab_lines( gd.src );
00291         char buf[10];
00292         sprintf( buf, "%09d", n + 1 );
00293         itab_append( gd.src, buf, talloc_strdup( gd.src, line ) );
00294     }

```

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

5.3.2.7 `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.

```
00278     {
00279     if( gd.src ) {
00280         talloc_free( gd.src );
00281     }
00282     gd.src = itab_new( );
00283     if( gd.src_iter ) {
00284         talloc_free( gd.src_iter );
00285     }
00286     gd.src_iter = NULL;
00287 }
```

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

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

5.3.2.8 `src_dump()`

```
bool src_dump ( )
```

dumps all the lines of the current source.

```
00320     {
00321     for( struct itab_iter * x = itab_foreach( gd.src );
00322         x; x = itab_next( x ) ) {
00323         printf( "%s:%s\n", itab_key( x ), itab_value( x ) );
00324     }
00325 }
```

5.3.2.9 `src_read()`

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

read file into itab.

read a file into src itab.

```
00299     {
00300     FILE *f = fopen( name, "r" );
00301     char buf[1000];
00302     char *line;
00303     int line_no = 1;
00304     src_clear( );
00305     for( ;; ) {
00306         line = fgets( buf, sizeof( buf ), f );
00307         if( line == NULL )
00308             break;
00309         int n = strlen( line );
00310         while( n > 0 && isspace( line[--n] ) )
00311             line[n] = 0;
00312         char line_number[10];
00313         sprintf( line_number, "%09d", line_no );
00314         itab_append( gd.src, line_number, talloc_strdup( gd.src, line ) );
00315         line_no++;
00316     }
00317     fclose( f );
00318 }
```

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

5.4 Messages

Data Structures

- struct **s_msgs**

Typedefs

- typedef char **t_msg**[200]

Functions

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

5.4.1 Detailed Description

5.5 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** [20]

5.5.1 Detailed Description

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_block](#)
- struct [s_expression](#)
- struct [s_messages](#)
- struct [s_message_cascade](#)
- struct [s_object](#)
- 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_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_env](#) [t_env](#)

Enumerations

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

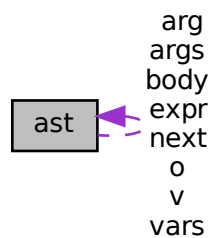
5.6.1 Detailed Description

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

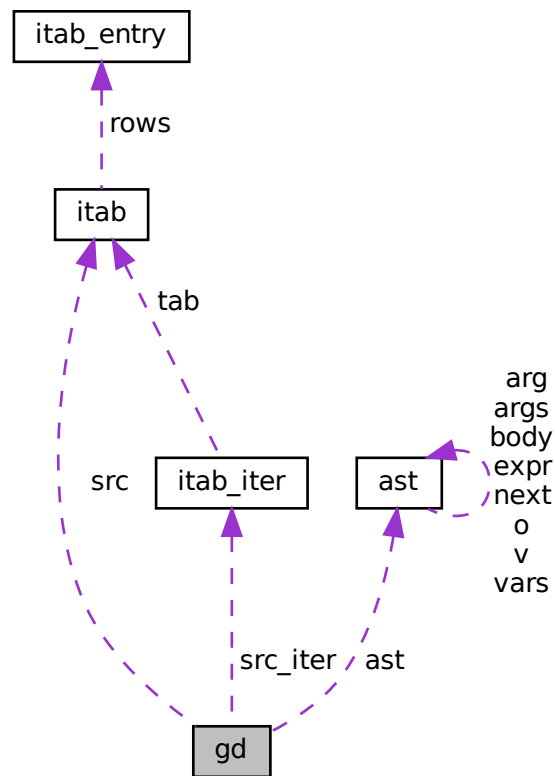
details of a class

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**

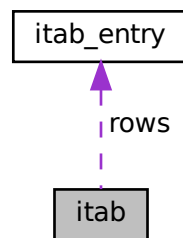
The documentation for this struct was generated from the following file:

- `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.

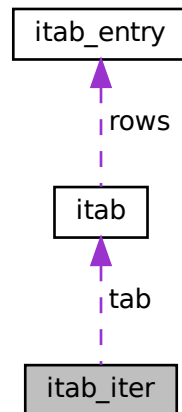
The documentation for this struct was generated from the following file:

- 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

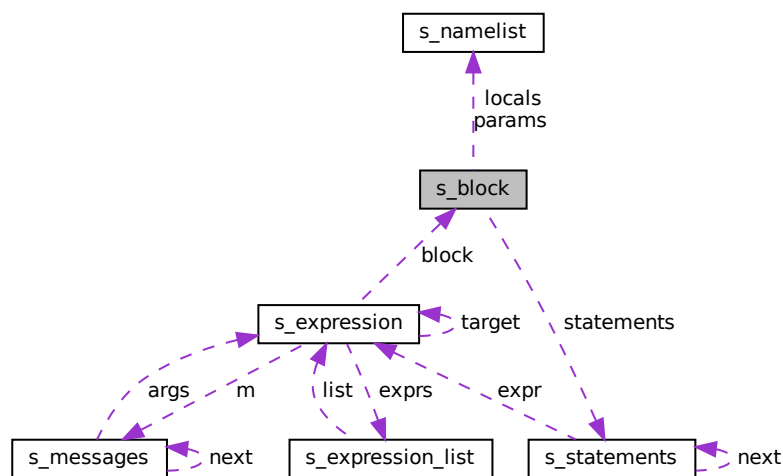
details of a method

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

- lib.c

6.8 s_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

6.9 s_classdef Struct Reference

Data Fields

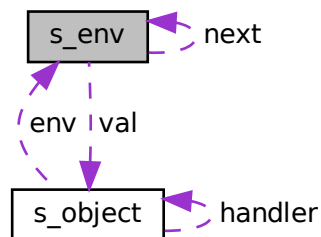
- int **id**
- char * **name**
- char * **meta**
- char * **super**

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

- lib.h

6.10 s_env Struct Reference

Collaboration diagram for s_env:



Data Fields

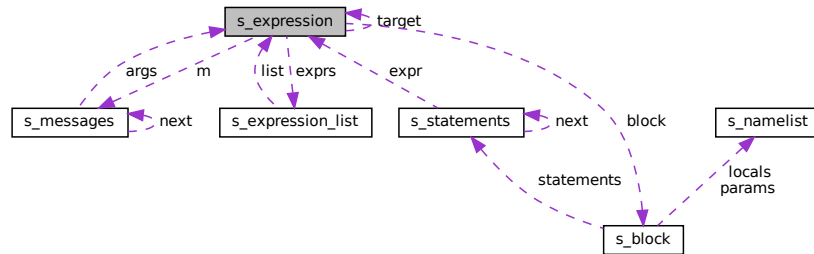
- const char * **name**
- `t_object` * **val**
- struct `s_env` * **next**

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

- lib.h

6.11 s_expression Struct Reference

Collaboration diagram for s_expression:



Data Fields

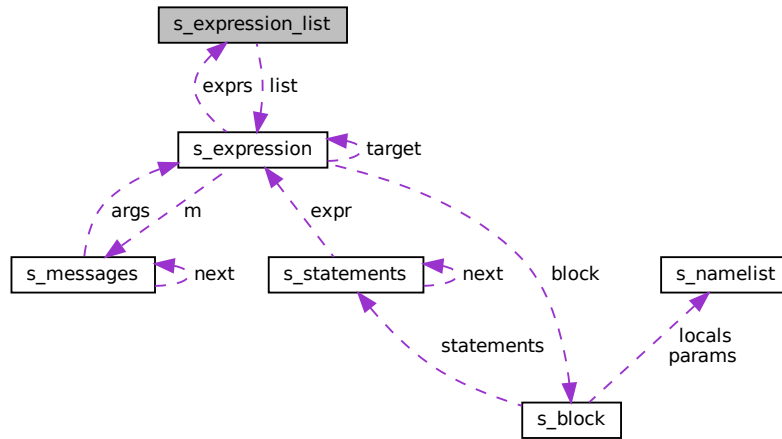
- `t_expression_tag` **tag**
- - union {
 - int **intvalue**
 - const char * **strvalue**
 - const char * **ident**
 - `t_expression_list` **exprs**
 - struct **msg** {
 - struct `s_expression` * **target**
 - struct `s_messages` * **m**
 - msg**
 - `t_block` **block**
 - } **u**

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

- `lib.h`

6.12 s_expression_list Struct Reference

Collaboration diagram for s_expression_list:



Data Fields

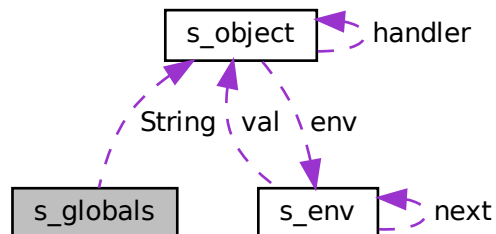
- int **count**
- struct [s_expression](#) ** **list**

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

- lib.h

6.13 s_globals Struct Reference

Collaboration diagram for s_globals:



Data Fields

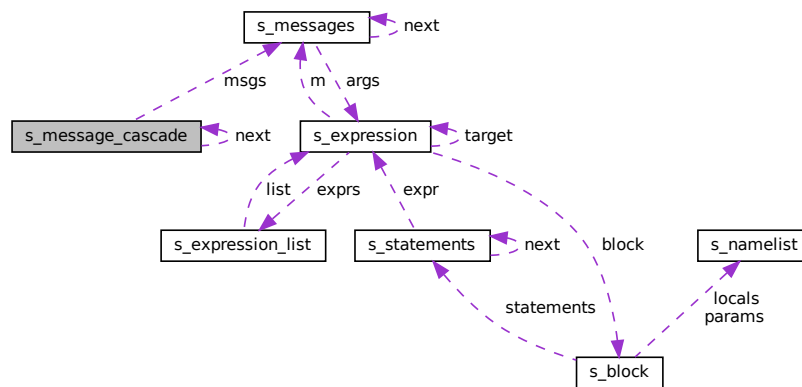
- [t_object](#) * **String**

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

- `tt_test.c`

6.14 s_message_cascade Struct Reference

Collaboration diagram for s_message_cascade:



Data Fields

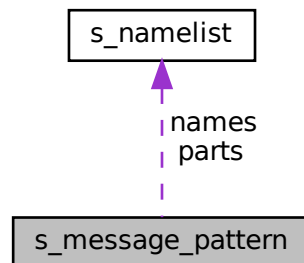
- [t_messages](#) * **msgs**
- struct [s_message_cascade](#) * **next**

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

- `lib.h`

6.15 s_message_pattern Struct Reference

Collaboration diagram for s_message_pattern:



Data Fields

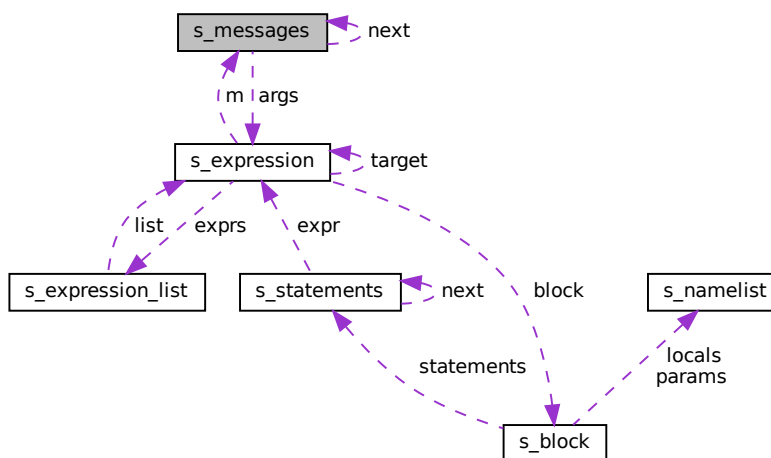
- `t_namelist` **parts**
- `t_namelist` **names**

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

- `lib.h`

6.16 s_messages Struct Reference

Collaboration diagram for s_messages:



Data Fields

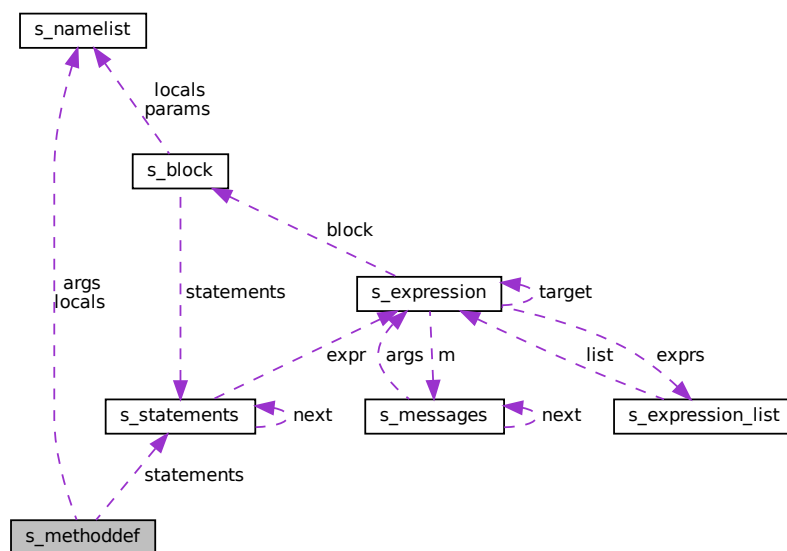
- char * **sel**
- int **argc**
- [t_expression](#) ** **args**
- struct [s_messages](#) * **next**

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

- lib.h

6.17 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

6.18 s_namelist Struct Reference

Data Fields

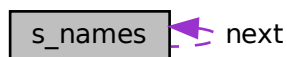
- int **count**
- char ** **names**

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

- lib.h

6.19 s_names Struct Reference

Collaboration diagram for s_names:



Data Fields

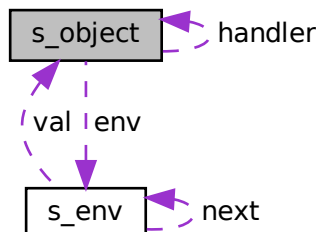
- char * **name**
- **t_names** next

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

- lib.h

6.20 s_object Struct Reference

Collaboration diagram for s_object:



Data Fields

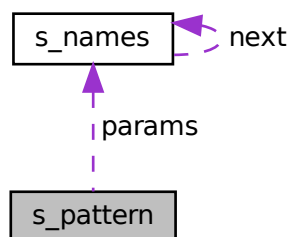
- t_message_handler **handler**
- void * **data**
- struct s_env * **env**

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

- lib.h

6.21 s_pattern Struct Reference

Collaboration diagram for s_pattern:



Data Fields

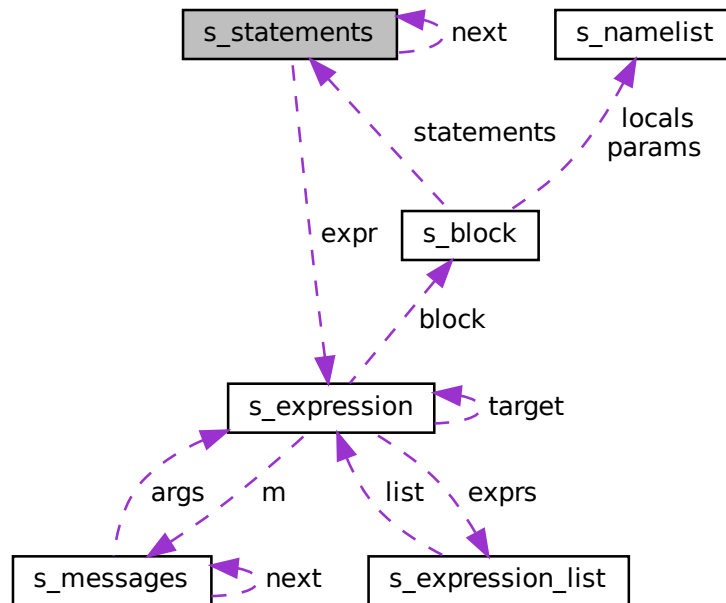
- char * **selector**
- t_names **params**

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

- lib.h

6.22 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`

6.23 stringinfo Struct Reference

Data Fields

- `int` **num**

6.23.1 Detailed Description

details of a string

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

- `lib.c`

6.24 varinfo Struct Reference

Data Fields

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

6.24.1 Detailed Description

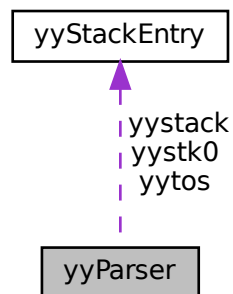
details of a global variable

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

- lib.c

6.25 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

6.26 yyStackEntry Struct Reference

Data Fields

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

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

- lempar.c