TinyTalk

1.0.0

Generated by Doxygen 1.9.1

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:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

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

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

5.1.2.2 itab_lines()

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

returns the number of lines in the table

```
                                    {
    assert( itab );
    return itab->used;
}
```

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.

```
                          {
    struct itab *r = talloc_zero( NULL, struct itab );
    r->total = 10;
    r->used = 0;
    r->rows = talloc_array( r, struct itab_entry, r->total );
    return r;
}
```

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 if c is an identifier character.

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

Referenced by nextToken().

5.2.2.2 nextToken()

```
bool nextToken ( )
```

read next token.

This is a more detailed description.

Returns

true if successful

```
                   {
    char c;
    bool result = false;
    while( true ) {
        while( readChar( &c ) && isspace( c ) );
        if( c == '"' ) {
            while( readChar( &c ) && c != '"' );
        }
        else
            break;
    }
    if( gd.state == 1 ) {
        if( isalpha( c ) ) {
            int idx = 0;
            for( ;; ) {
                gd.buf[idx++] = c;
                readChar( &c );
                if( !is_ident_char( c ) )
                    break;
            }
            if( c == ':' ) {
                gd.buf[idx++] = c;
                gd.token = TK_KEYWORD;
            }
            else {
                gd.pos--;
                gd.token = TK_IDENT;
            }
            gd.buf[idx] = 0;
            result = true;
        }
        else if( is_binary_char( c ) ) {
            for( int idx = 0; is_binary_char( c ); idx++ ) {
                gd.buf[idx] = c;
                gd.buf[idx + 1] = 0;
                readChar( &c );
            }
            gd.pos--;
            gd.token = 0;
            gd.token = TK_BINOP;
            result = true;
            if( strcmp( ":=", gd.buf ) == 0 ) {
                gd.token = TK_ASSIGN;
                result = true;
            }
            else if( strcmp( "<-", gd.buf ) == 0 ) {
                gd.token = TK_LARROW;
                result = true;
            }
            else if( strcmp( "|", gd.buf ) == 0 ) {
                gd.token = TK_BAR;
                result = true;
            }
            else if( 0 == strcmp( "<", gd.buf ) ) {
                gd.token = TK_LT;
                result = true;
            }
            else if( 0 == strcmp( ">", gd.buf ) ) {
                gd.token = TK_GT;
                result = true;
            }
        }
        else if( isdigit( c ) ) {
            int idx = 0;
            while( isdigit( c ) ) {
                printf("### digit %c\n", c);
                gd.buf[idx++] = c;
                gd.buf[idx] = 0;
                readChar( &c );
            }
            gd.pos--;
            gd.token = TK_NUMBER;
            result = true;
        }
        else {
            switch ( c ) {
                case '\"':
                    result = readStringToken(  );
                    break;
                case '.':
                    result = true;
                    gd.token = TK_DOT;
                    break;
                case ';':
                    result = true;
                    gd.token = TK_SEMICOLON;
                    break;
                case '(':
                    result = true;
                    gd.token = TK_LPAREN;
                    break;
                case ')':
                    result = true;
                    gd.token = TK_RPAREN;
                    break;
                case '[':
                    result = true;
                    gd.token = TK_LBRACK;
                    break;
                case ']':
                    result = true;
                    gd.token = TK_RBRACK;
                    break;
                case '{':
                    result = true;
                    gd.token = TK_LBRACE;
                    break;
                case '}':
                    result = true;
                    gd.token = TK_RBRACE;
                    break;
                case '#':
                    readChar( &c );
                    for( int idx = 0; is_ident_char( c ) || c == ':'; idx++ ) {
                        gd.buf[idx] = c;
                        gd.buf[idx + 1] = 0;
                        readChar( &c );
                    }
                    gd.pos--;
                    gd.token = TK_SYMBOL;
                    result = true;
                    break;
                case '^':
                    result = true;
                    gd.token = TK_UARROW;
                    break;
                case ':':
                    result = true;
                    gd.token = TK_COLON;
                    readChar( &c );
                    if( c == '=' ) {
                        gd.token = TK_ASSIGN;
                    }
                    else
                        gd.pos--;
                    break;
                case '$':
                    result = true;
                    gd.token = TK_CHAR;
                    readChar( &c );
                    gd.buf[0] = c;
                    gd.buf[1] = 0;
                    break;
                default:
                        gd.pos--;
                        break;
                }
            }
    }
    return result;
}
```

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

```
                        {
    bool result = true;
    if( gd.state == 0 ) {
        result = readLine(  );
    }
    if( result ) {
        *t = gd.line[gd.pos++];
        while( *t == 0 ) {
            if( readLine(  ) ) {
                *t = gd.line[gd.pos++];
            }
            else {
                result = false;
                break;
            }
        }
    }
    return result;
}
```

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.

```
                 {
    if( gd.src_iter == NULL ) {
        gd.src_iter = itab_foreach( gd.src );
    }
    else {
        gd.src_iter = itab_next( gd.src_iter );
    }
    if( gd.src_iter ) {
        gd.line = itab_value( gd.src_iter );
        gd.line_count++;
        printf( "%2d:%s\n", gd.line_count, gd.line );
        gd.pos = 0;
        gd.state = 1;
        return true;
    }
    else {
        gd.line = "";
        gd.state = 2;
        return false;
    }
}
```

Referenced by readChar().

5.2.2.5 readStringToken()

```
bool readStringToken (
               void   )
```

read string token.

Returns

true if successful

```
                          {
    int idx = 0;
    char c;
    while( readChar( &c ) && '\"' != c ) {
        if( c == '\\' )
            readChar( &c );
        gd.buf[idx++] = c;
    }
    gd.buf[idx] = 0;
    gd.token = TK_STRING;
    return true;
}
```

References readChar().

5.2.2.6 src_add()

```
bool src_add (
               const char *   )
```

adding one line to the source that will be parsed.

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

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.

```
                  {
    if( gd.src ) {
        talloc_free( gd.src );
    }
    gd.src = itab_new(  );
    if( gd.src_iter ) {
        talloc_free( gd.src_iter );
    }
    gd.src_iter = NULL;
}
```

References itab_new().

Referenced by src_read().

5.2.2.8 src_dump()

```
bool src_dump ( )
```

dumps all the lines of the current source.

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

5.2.2.9 src_read()

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

read file into itab.

read a file into src itab.

```
                                 {
    FILE *f = fopen( name, "r" );
    char buf[1000];
    char *line;
    int line_no = 1;
    src_clear(  );
    for( ;; ) {
        line = fgets( buf, sizeof( buf ), f );
        if( line == NULL )
            break;
        int n = strlen( line );
        while( n > 0 && isspace( line[--n] ) )
            line[n] = 0;
        char line_number[10];
        sprintf( line_number, "%09d", line_no );
        itab_append( gd.src, line_number, talloc_strdup( gd.src, line ) );
        line_no++;
    }
    fclose( f );
}
```

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

nl	the modified list
name	the string to be added

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

5.5.2.2 namelist_copy()

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

make a deep copy of a name list

Parameters

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

```
                                                          {
    to->count = from->count;
    to->names = talloc_array( NULL, char *, to->count );
    assert(talloc_get_type(from->names, char *));
    for( int i = 0; i < to->count; i++ ) {
        to->names[i] = talloc_strdup( to->names, from->names[i] );
    }
    assert(talloc_get_type(to->names, char *));
}
```

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

nl	reference to an existing structure to be initialized.

```
                                     {
    nl->count = 0;
    nl->names = NULL;
}
```

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** }

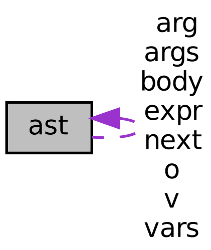
5.6.1 Detailed Description

Chapter 6

Data Structure Documentation

6.1 ast Struct Reference

Collaboration diagram for ast:



Data Fields

- int tag

    *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
```

6.1.1 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

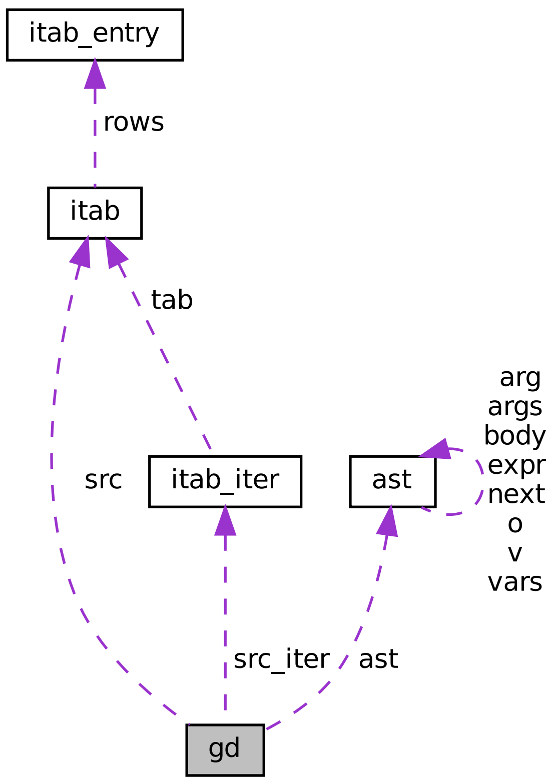
details of a class

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**

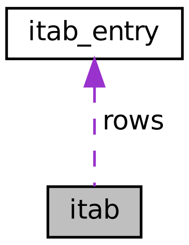
The documentation for this struct was generated from the following file:

- 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.

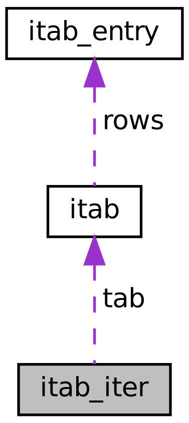
The documentation for this struct was generated from the following file:

- 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

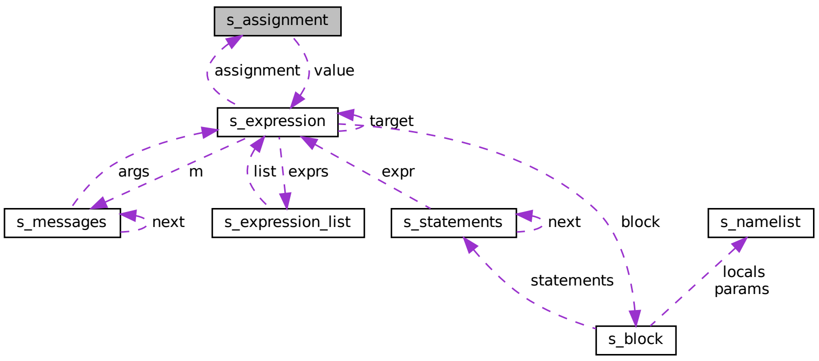
details of a method

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

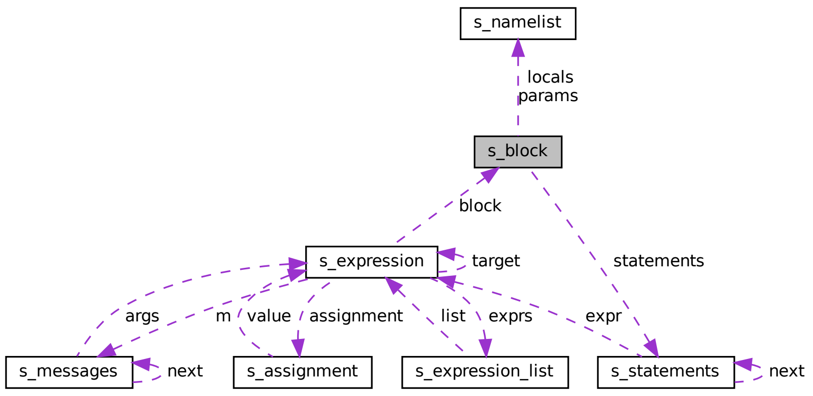
- const char ∗ **target**
- struct s_expression ∗ **value**

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

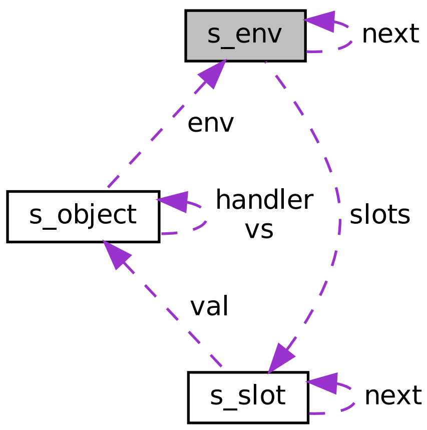
- int **id**
- char ∗ **name**
- char ∗ **meta**
- char ∗ **super**

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

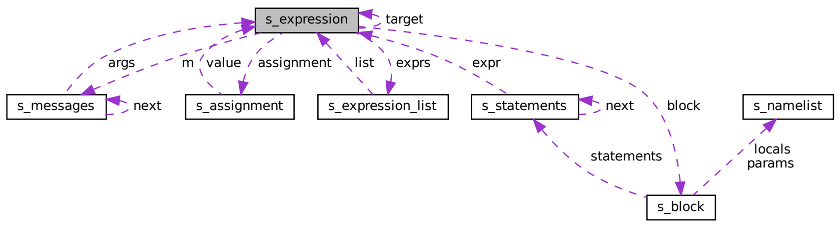
- t_slot ∗ **slots**
- struct s_env ∗ **next**

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**
-

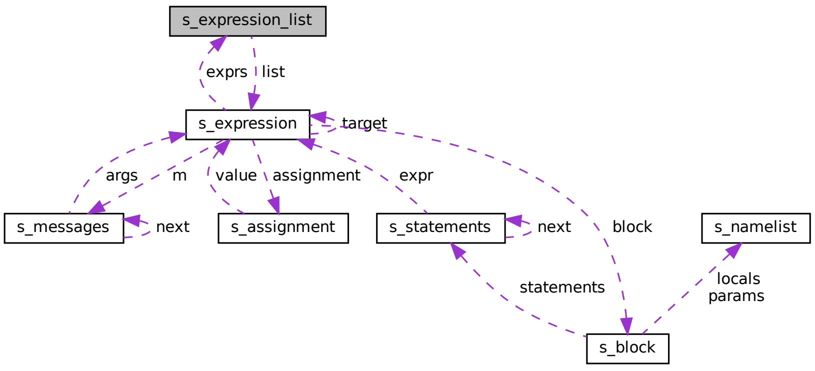
  union {
    int **intvalue**
    const char ∗ **strvalue**
    const char ∗ **ident**
    t_expression_list **exprs**
    struct **msg** {
      struct s_expression ∗ **target**
      struct s_messages ∗ **m**
    } **msg**
    t_assignment **assignment**
    t_block **block**
  } **u**

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

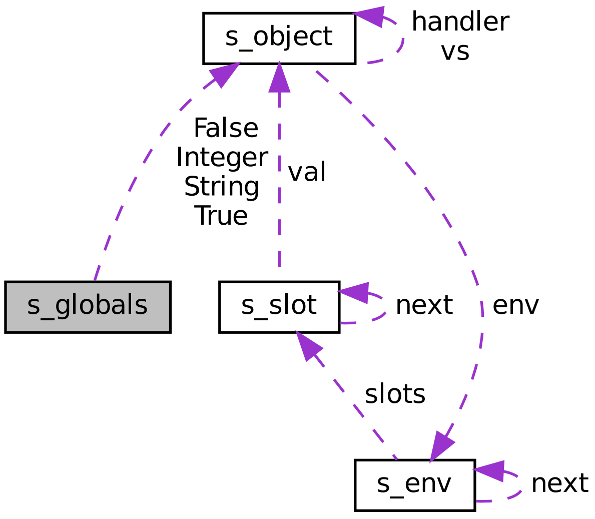
- int **count**
- struct s_expression ∗∗ **list**

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

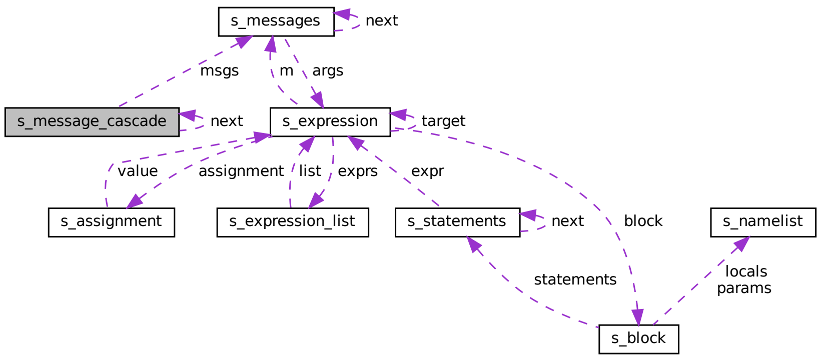
- t_object ∗ **String**
- t_object ∗ **Integer**
- t_object ∗ **True**
- t_object ∗ **False**

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

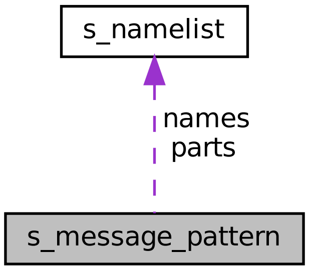
- t_messages ∗ **msgs**
- struct s_message_cascade ∗ **next**

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

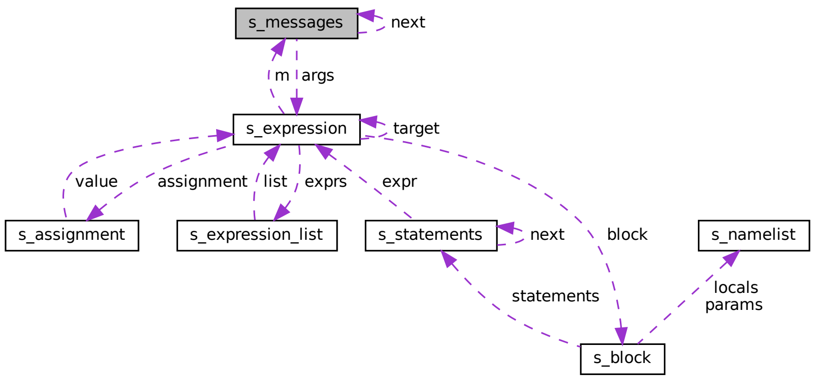
- t_namelist **parts**
- t_namelist **names**

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

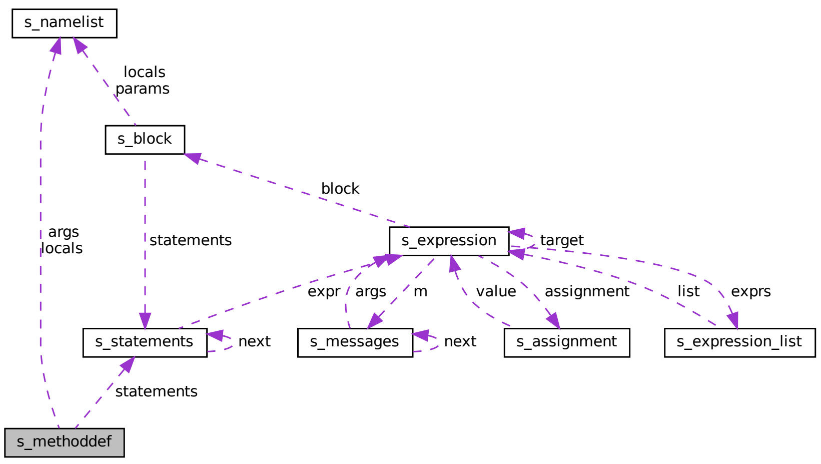
- bool **cascaded**
- char ∗ **sel**
- int **argc**
- t_expression ∗∗ **args**
- struct s_messages ∗ **next**

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

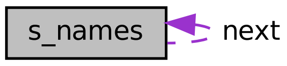
- int **count**
- char ∗∗ **names**

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

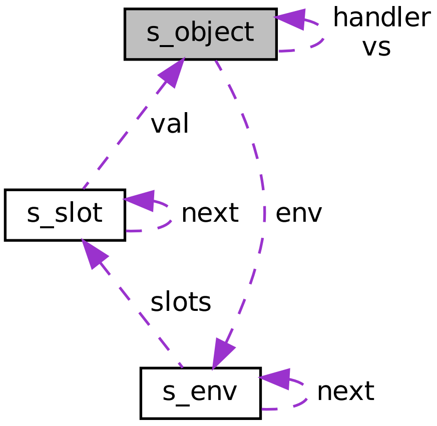
- char ∗ **name**
- t_names **next**

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
```

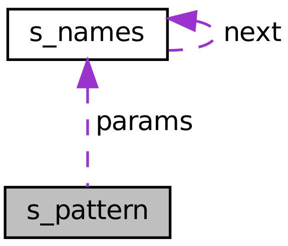
- struct s_env ∗ **env**

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

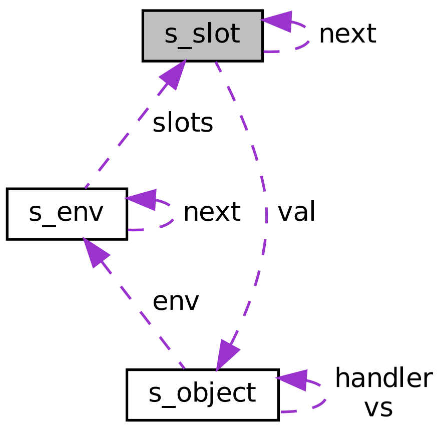
- char ∗ **selector**
- t_names **params**

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

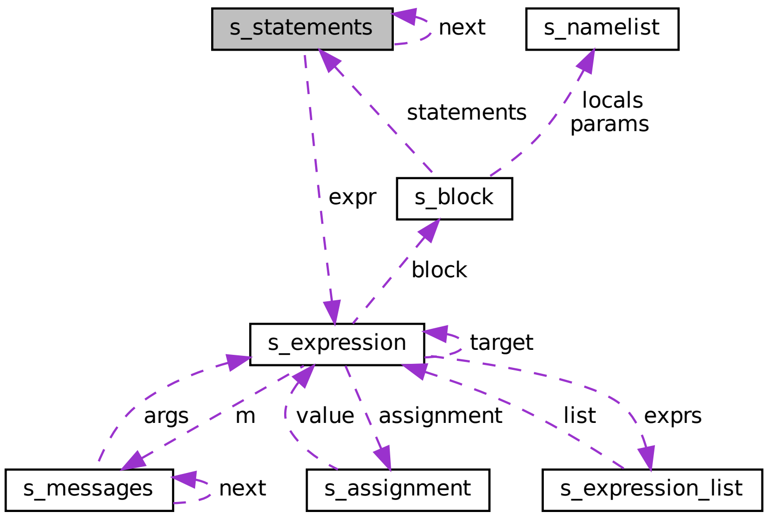
- const char ∗ **name**
- t_object ∗ **val**
- struct s_slot ∗ **next**

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

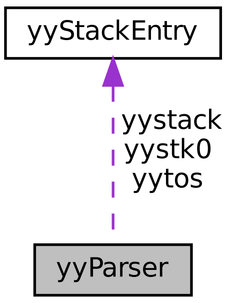
details of a global variable

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 **yystksz**
- yyStackEntry ∗ **yystack**
- yyStackEntry **yystk0**

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