42h

v0.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 node_prefix::prefix::assigment_word Struct Reference

`#include <ast.h>`

Data Fields

- char * variable_name
- char * value

3.1.1 Field Documentation

3.1.1.1 value

`char* value`

3.1.1.2 variable_name

`char* variable_name`

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

- src/ast/ast.h

3.2 buffer Struct Reference

`#include <buffer.h>`

Data Fields

- char * buf
- int index

3.2.1 Field Documentation

3.2.1.1 buf

```
char* buf
```

3.2.1.2 index

```
int index
```

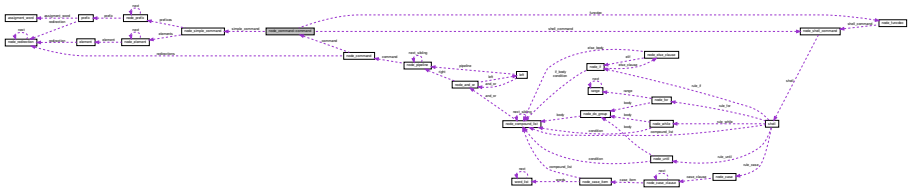
The documentation for this struct was generated from the following file:

- src/utils/buffer.h

3.3 node_command::command Union Reference

```
#include <ast.h>
```

Collaboration diagram for node_command::command:



Data Fields

- struct node_simple_command * simple_command
- struct node_shell_command * shell_command
- struct node_funcdec * funcdec

3.3.1 Field Documentation

3.3.1.1 funcdec

```
struct node_funcdec* funcdec
```

3.3.1.2 shell_command

```
struct node_shell_command* shell_command
```

3.3.1.3 simple_command

```
struct node_simple_command* simple_command
```

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

- src/ast/ast.h

3.4 commands Struct Reference

```
#include <exec.h>
```

Data Fields

- const char * name
- void(* function)(char **args)

3.4.1 Field Documentation

3.4.1.1 function

```
void(* function) (char **args)
```

3.4.1.2 name

```
const char* name
```

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

- src/exec/exec.h

3.5 echo_tab Struct Reference

`#include <commands.h>`

Data Fields

- char name
- char corresp

3.5.1 Field Documentation

3.5.1.1 corresp

`char corresp`

3.5.1.2 name

`char name`

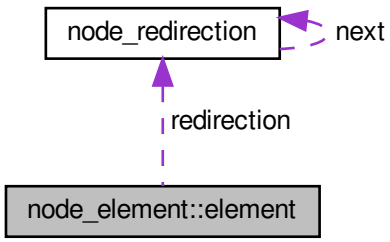
The documentation for this struct was generated from the following file:

- src/exec/commands.h

3.6 node_element::element Union Reference

`#include <ast.h>`

Collaboration diagram for node_element::element:



Data Fields

- char * word
- struct node_redirection * redirection

3.6.1 Field Documentation

3.6.1.1 redirection

```
struct node_redirection* redirection
```

3.6.1.2 word

```
char* word
```

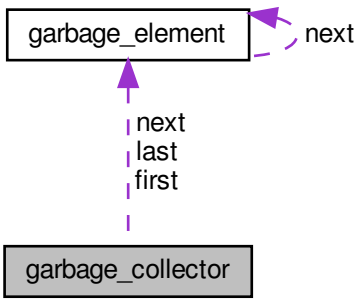
The documentation for this union was generated from the following file:

- src/ast/ast.h

3.7 garbage_collector Struct Reference

```
#include <garbage_collector.h>
```

Collaboration diagram for garbage_collector:



Data Fields

- struct garbage_element * first
- struct garbage_element * next
- struct garbage_element * last

3.7.1 Field Documentation

3.7.1.1 first

```
struct garbage_element* first
```

3.7.1.2 last

```
struct garbage_element* last
```

3.7.1.3 next

```
struct garbage_element* next
```

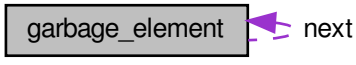
The documentation for this struct was generated from the following file:

- src/garbage_collector/garbage_collector.h

3.8 garbage_element Struct Reference

```
#include <garbage_collector.h>
```

Collaboration diagram for garbage_element:



Data Fields

- struct garbage_element * next
- void * addr

3.8.1 Field Documentation

3.8.1.1 addr

```
void* addr
```

3.8.1.2 next

```
struct garbage_element* next
```

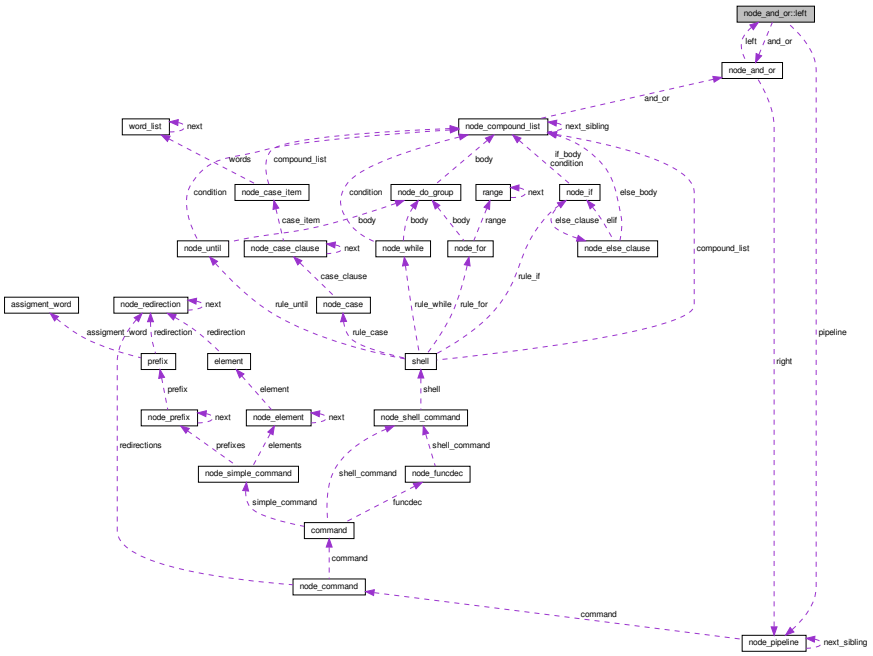
The documentation for this struct was generated from the following file:

- src/garbage_collector/garbage_collector.h

3.9 node_and_or::left Union Reference

```
#include <ast.h>
```

Collaboration diagram for node_and_or::left:



Data Fields

- struct node_pipeline * pipeline
- struct node_and_or * and_or

3.9.1 Field Documentation

3.9.1.1 and_or

```
struct node_and_or* and_or
```

3.9.1.2 pipeline

```
struct node_pipeline* pipeline
```

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

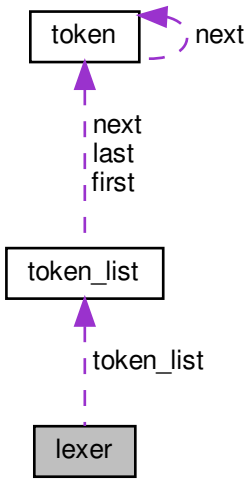
- src/ast/ast.h

3.10 lexer Struct Reference

Lexer architecture and methods.

```
#include <lexer.h>
```

Collaboration diagram for lexer:



Data Fields

- char * input
- struct token_list * token_list

3.10.1 Detailed Description

Lexer architecture and methods.

Parameters

input	the full input string.
token_list	the linked-list of tokens.

3.10.2 Field Documentation

3.10.2.1 input

```
char* input
```

3.10.2.2 token_list

```
struct token_list* token_list
```

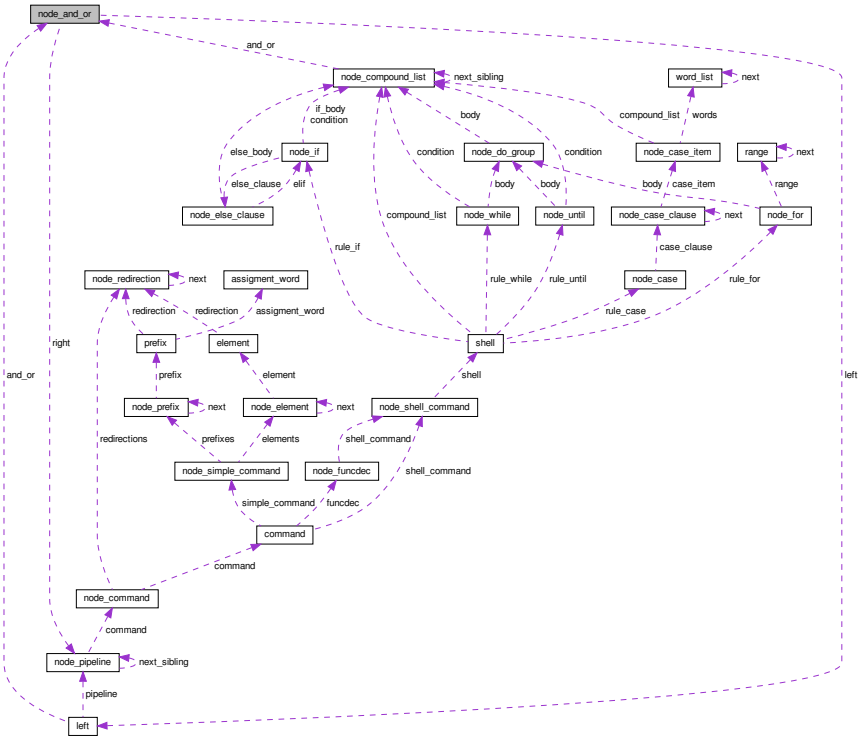
The documentation for this struct was generated from the following file:

- src/lexer/lexer.h

3.11 node_and_or Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_and_or:



Data Structures

- union left

Public Types

- enum type_logical { AND, OR }

Data Fields

- bool is_final
- union node_and_or::left left
- struct node_pipeline * right
- enum node_and_or::type_logical type

3.11.1 Member Enumeration Documentation

3.11.1.1 type_logical

```
enum type_logical
```

Enumerator

AND	
OR	

3.11.2 Field Documentation

3.11.2.1 is_final

```
bool is_final
```

3.11.2.2 left

```
union node_and_or::left left
```

3.11.2.3 right

```
struct node_pipeline* right
```

3.11.2.4 type

```
enum node_and_or::type_logical type
```

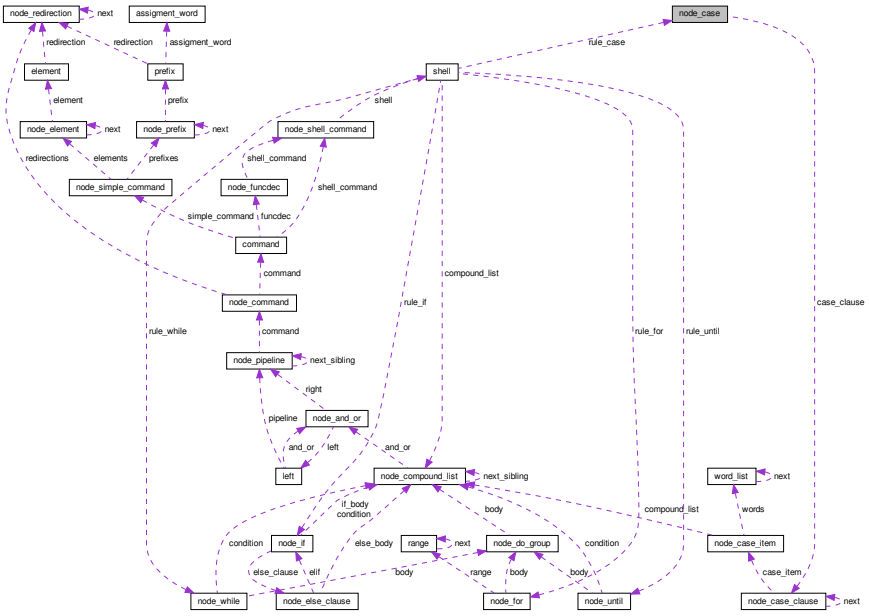
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.12 node_case Struct Reference

`#include <ast.h>`

Collaboration diagram for node_case:



Data Fields

- bool is_case_clause
- char * word
- struct node_case_clause * case_clause

3.12.1 Field Documentation

3.12.1.1 case_clause

`struct node_case_clause* case_clause`

3.12.1.2 is_case_clause

`bool is_case_clause`

3.12.1.3 word

```
char* word
```

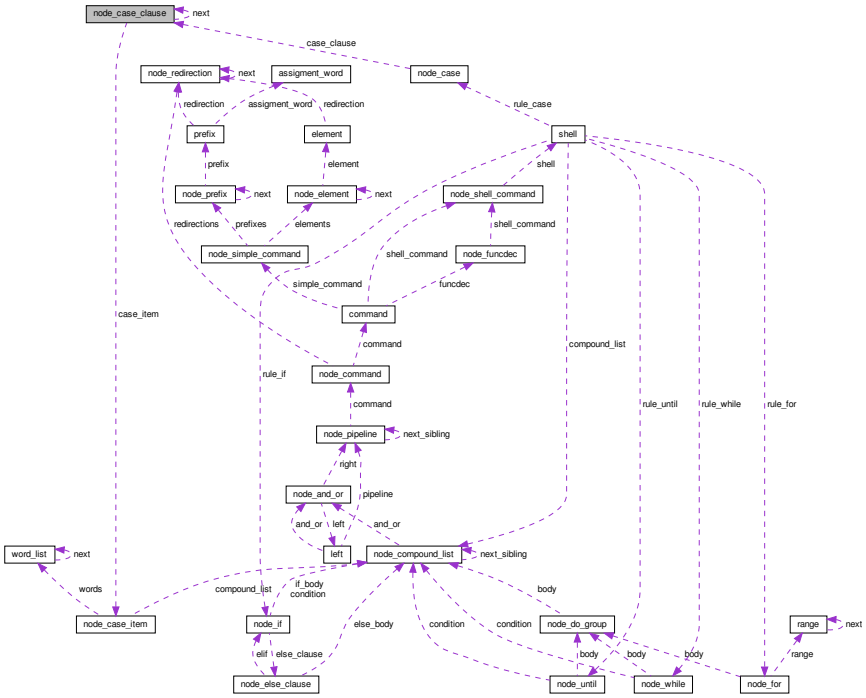
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.13 node_case_clause Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_case_clause:



Data Fields

- struct node_case_item * case_item
- struct node_case_clause * next

3.13.1 Field Documentation

3.13.1.1 case_item

```
struct node_case_item* case_item
```

3.13.1.2 next

```
struct node_case_clause* next
```

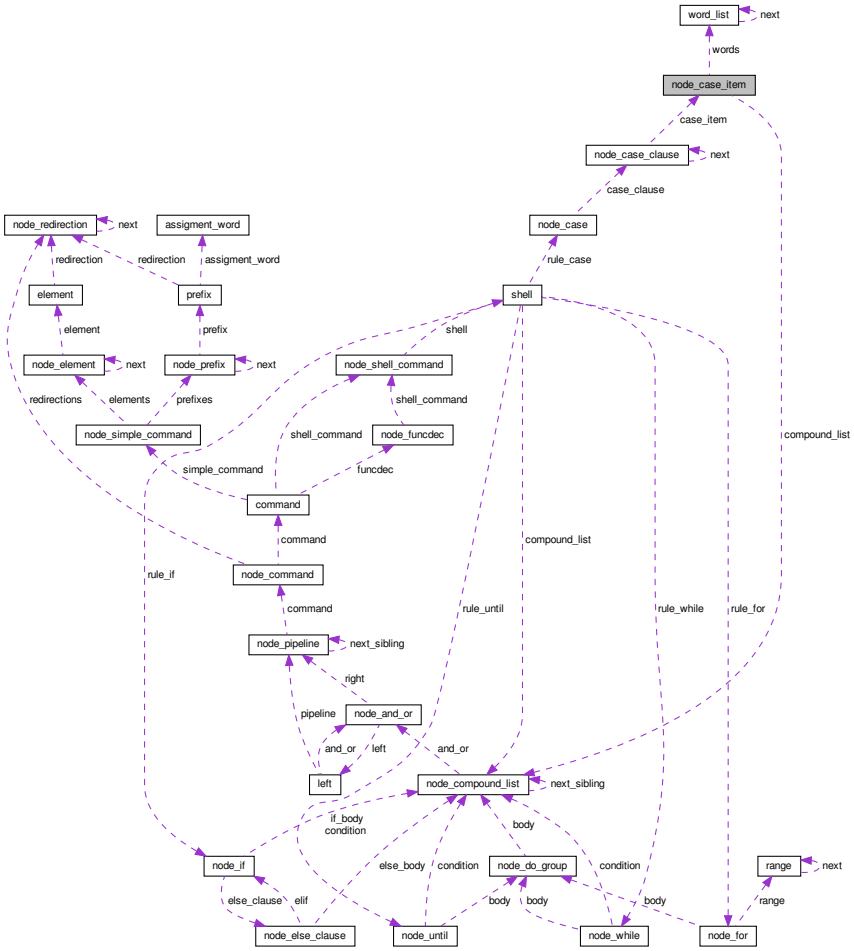
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.14 node_case_item Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_case_item:



Data Fields

- struct word_list * words
- struct node_compound_list * compound_list

3.14.1 Field Documentation

3.14.1.1 compound_list

```
struct node_compound_list* compound_list
```

3.14.1.2 words

```
struct word_list* words
```

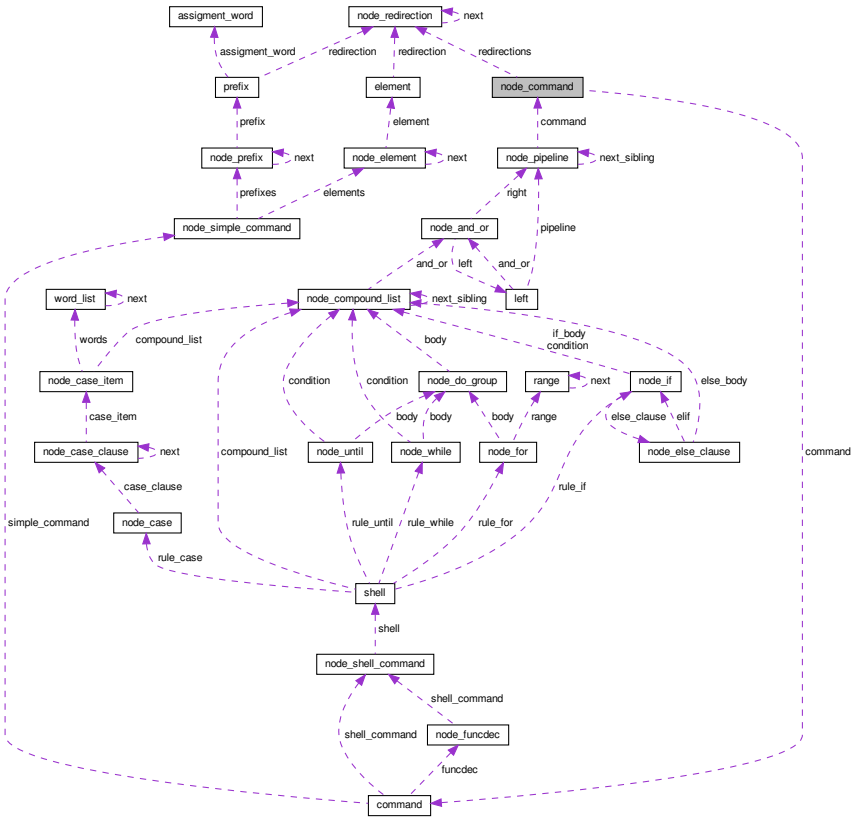
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.15 node_command Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_command:



Data Structures

- union command

Public Types

- enum command_token { SIMPLE_COMMAND, SHELL_COMMAND, FUNCDEC }

Data Fields

- enum node_command::command_token type
- union node_command::command command
- struct node_redirection * redirections

3.15.1 Member Enumeration Documentation

3.15.1.1 command_token

```
enum command_token
```

Enumerator

SIMPLE_COMMAND	
SHELL_COMMAND	
FUNCDEC	

3.15.2 Field Documentation

3.15.2.1 command

```
union node_command::command command
```

3.15.2.2 redirections

```
struct node_redirection* redirections
```

3.15.2.3 type

```
enum node_command::command_token type
```

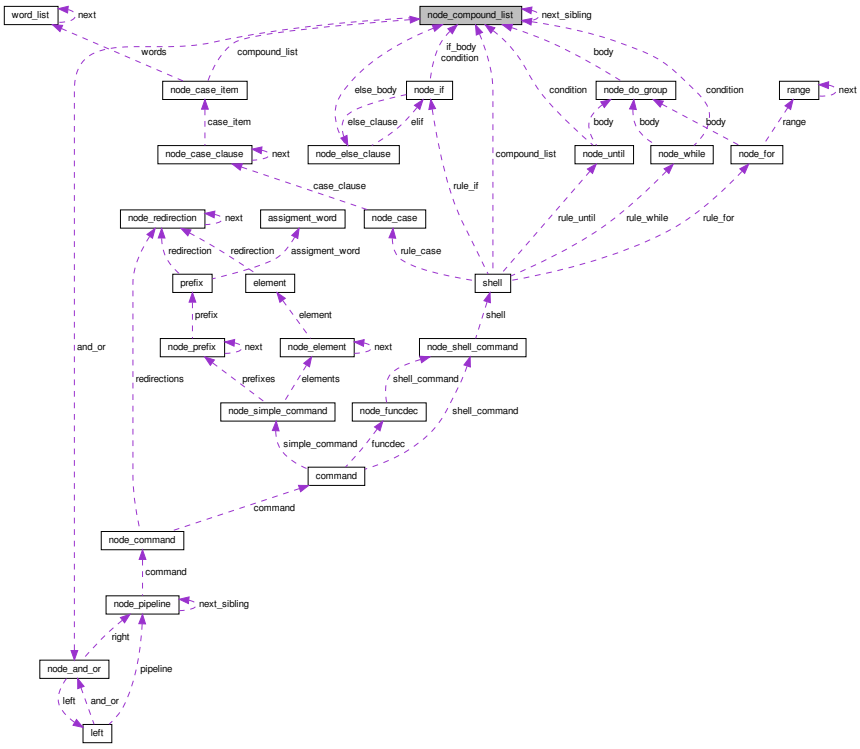
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.16 node_compound_list Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_compound_list:



Data Fields

- struct node_and_or * and_or
- struct node_compound_list * next_sibling

3.16.1 Field Documentation

3.16.1.1 and_or

```
struct node_and_or* and_or
```

3.16.1.2 next_sibling

```
struct node_compound_list* next_sibling
```

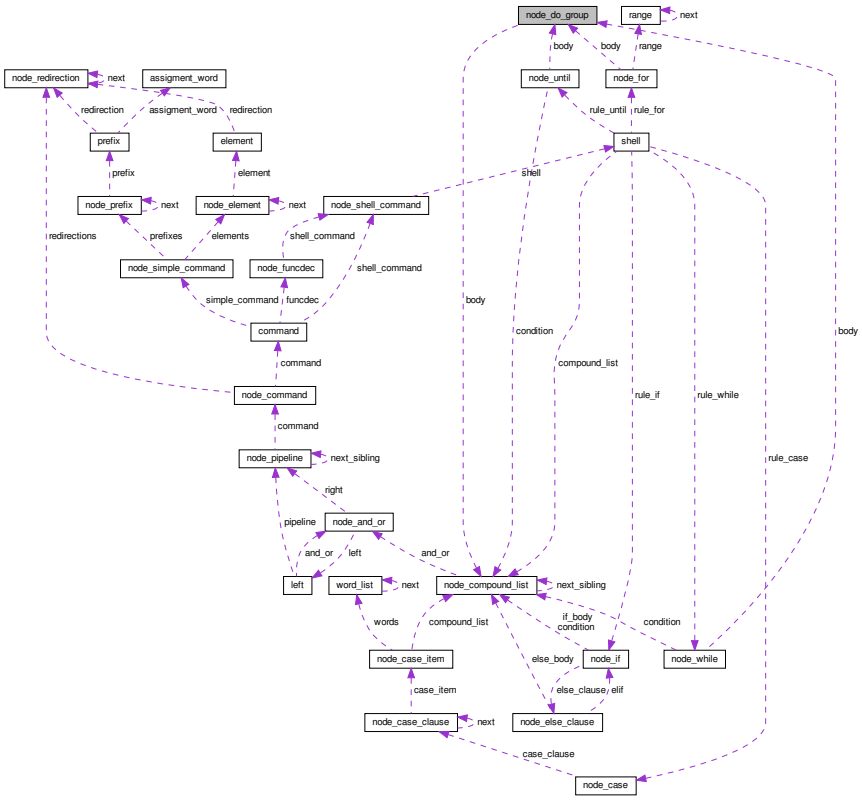
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.17 node_do_group Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_do_group:



Data Fields

- struct node_compound_list * body

3.17.1 Field Documentation

3.17.1.1 body

```
struct node_compound_list* body
```

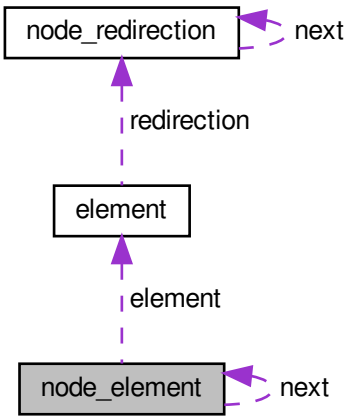
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.18 node_element Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_element:



Data Structures

- union element

Public Types

- enum type_element { TOKEN_REDIRECTION, WORD }

Data Fields

- struct node_element * next
- enum node_element::type_element type
- union node_element::element element

3.18.1 Member Enumeration Documentation

3.18.1.1 type_element

```
enum type_element
```

Enumerator

TOKEN_REDIRECTION	
WORD	

3.18.2 Field Documentation

3.18.2.1 element

```
union node_element::element element
```

3.18.2.2 next

```
struct node_element* next
```

3.18.2.3 type

```
enum node_element::type_element type
```

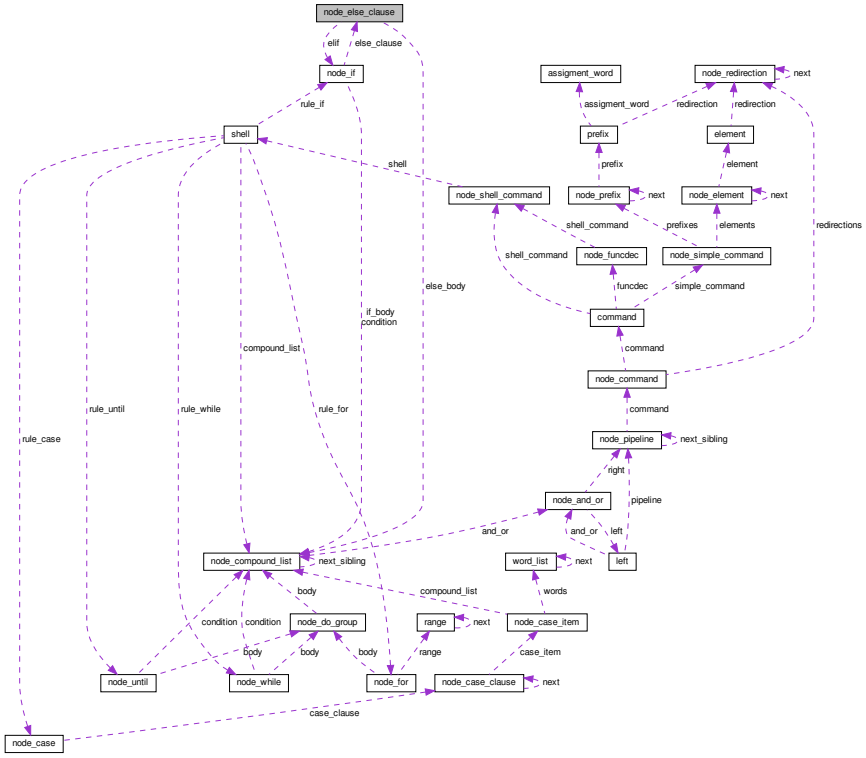
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.19 node_else_clause Struct Reference

`#include <ast.h>`

Collaboration diagram for node_else_clause:



Public Types

- enum else_clause { ELIF, ELSE }

Data Fields

- enum node_else_clause::else_clause type
- union {
  struct node_if * elif
  struct node_compound_list * else_body
  } clause

3.19.1 Member Enumeration Documentation

3.19.1.1 else_clause

`enum else_clause`

Enumerator

ELIF	
ELSE	

3.19.2 Field Documentation

3.19.2.1 clause

```
union { ... } clause
```

3.19.2.2 elif

```
struct node_if* elif
```

3.19.2.3 else_body

```
struct node_compound_list* else_body
```

3.19.2.4 type

```
enum node_else_clause::else_clause type
```

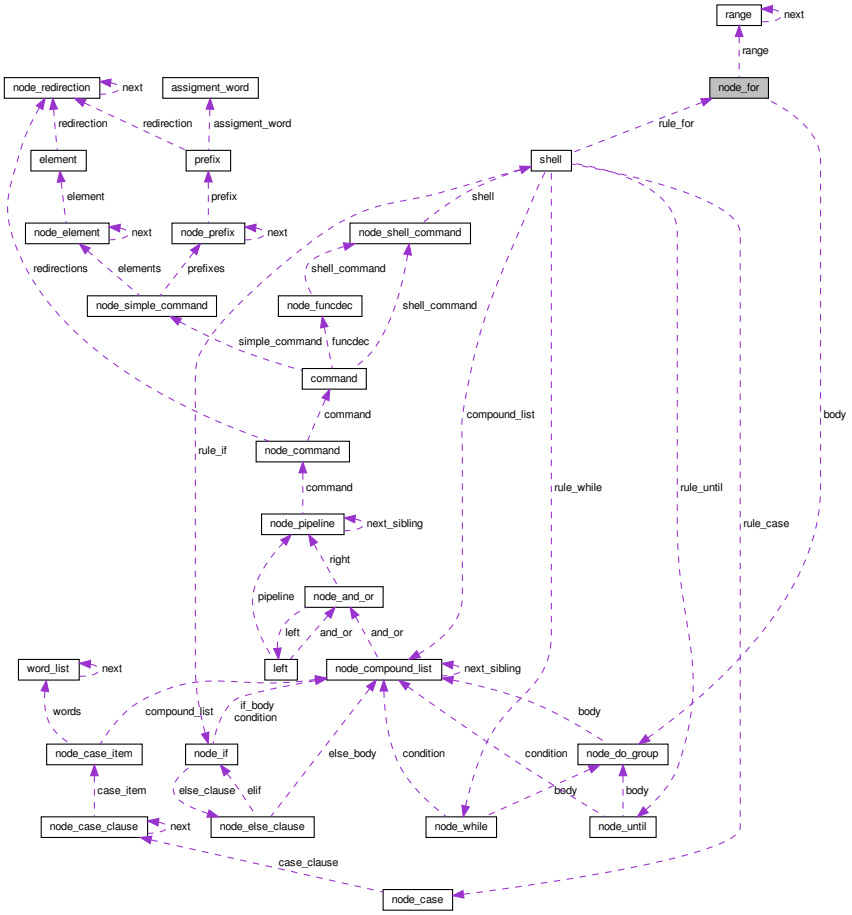
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.20 node_for Struct Reference

`#include <ast.h>`

Collaboration diagram for node_for:



Data Fields

- char * variable_name
- struct range * range
- struct node_do_group * body

3.20.1 Field Documentation

3.20.1.1 body

`struct node_do_group* body`

3.20.1.2 range

```
struct range* range
```

3.20.1.3 variable_name

```
char* variable_name
```

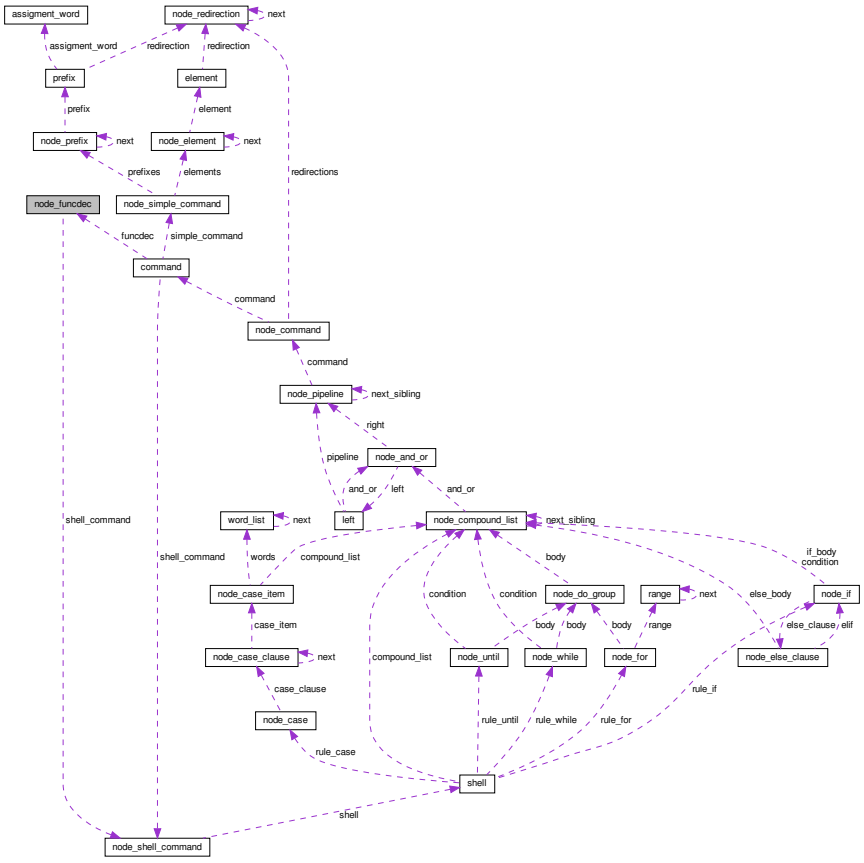
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.21 node_funcdec Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_funcdec:



Data Fields

- bool is_function
- char * function_name
- struct node_shell_command * shell_command

3.21.1 Field Documentation

3.21.1.1 function_name

```
char* function_name
```

3.21.1.2 is_function

```
bool is_function
```

3.21.1.3 shell_command

```
struct node_shell_command* shell_command
```

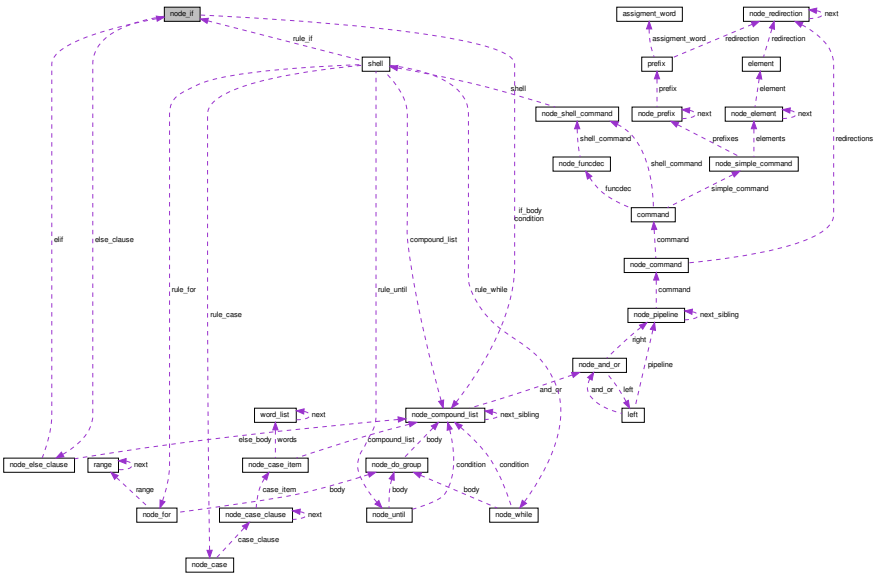
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.22 node_if Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_if:



Data Fields

- struct node_compound_list * condition
- struct node_compound_list * if_body
- struct node_else_clause * else_clause

3.22.1 Field Documentation

3.22.1.1 condition

```
struct node_compound_list* condition
```

3.22.1.2 else_clause

```
struct node_else_clause* else_clause
```

3.22.1.3 if_body

```
struct node_compound_list* if_body
```

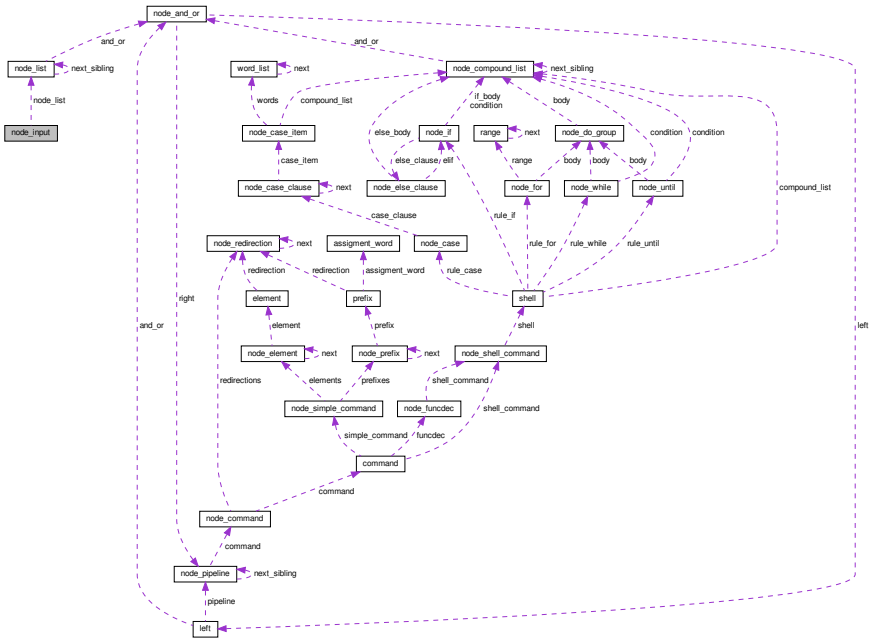
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.23 node_input Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_input:



Data Fields

- struct node_list * node_list

3.23.1 Field Documentation

3.23.1.1 node_list

```
struct node_list* node_list
```

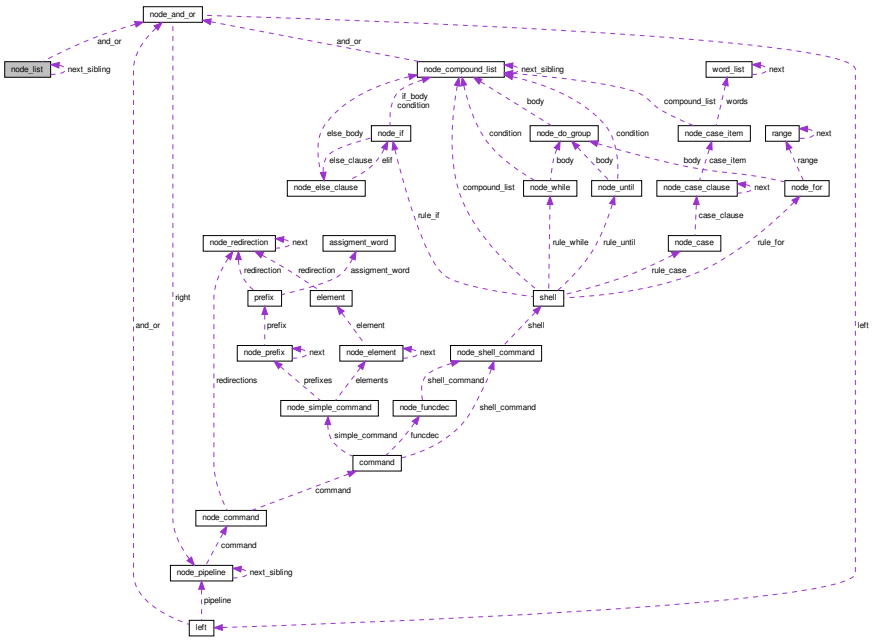
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.24 node_list Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_list:



Public Types

- enum type { SEMI, SEPAND, NONE }

Data Fields

- struct node_and_or * and_or
- struct node_list * next_sibling
- enum node_list::type type

3.24.1 Member Enumeration Documentation

3.24.1.1 type

```
enum type
```

Enumerator

SEMI	
SEPAND	
NONE	

3.24.2 Field Documentation

3.24.2.1 and_or

```
struct node_and_or* and_or
```

3.24.2.2 next_sibling

```
struct node_list* next_sibling
```

3.24.2.3 type

```
enum node_list::type type
```

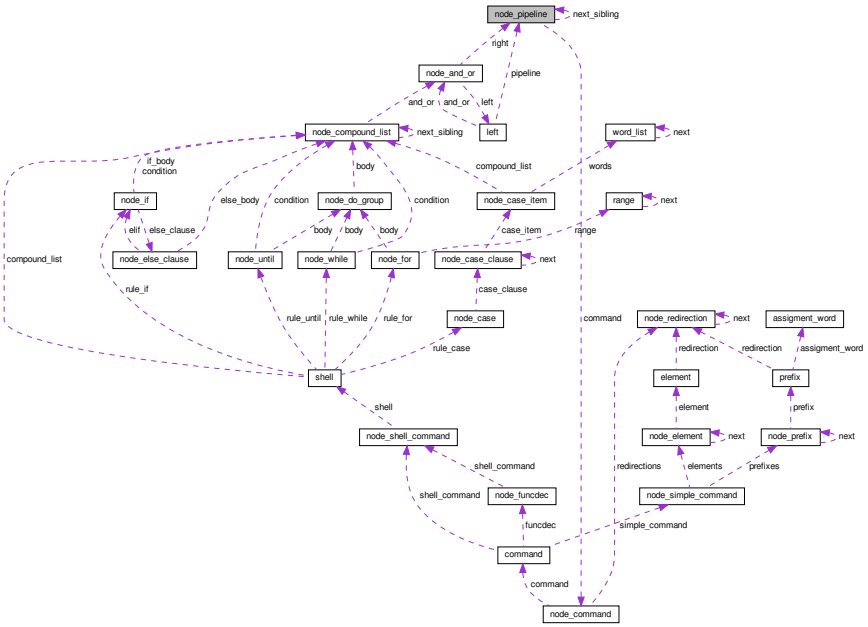
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.25 node_pipeline Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_pipeline:



Data Fields

- bool is_not
- struct node_command * command
- struct node_pipeline * next_sibling

3.25.1 Field Documentation

3.25.1.1 command

```
struct node_command* command
```

3.25.1.2 is_not

```
bool is_not
```

3.25.1.3 next_sibling

```
struct node_pipeline* next_sibling
```

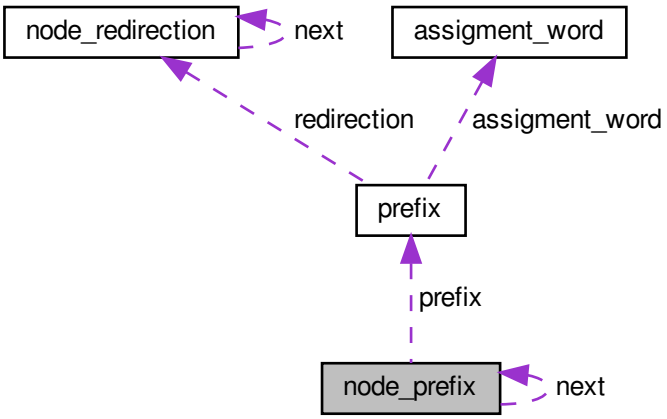
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.26 node_prefix Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_prefix:



Data Structures

- union prefix

Public Types

- enum type_prefix { REDIRECTION, ASSIGMENT_WORD }

Data Fields

- struct node_prefix * next
- enum node_prefix::type_prefix type
- union node_prefix::prefix prefix

3.26.1 Member Enumeration Documentation

3.26.1.1 type_prefix

```
enum type_prefix
```

Enumerator

REDIRECTION	
ASSIGMENT_WORD	

3.26.2 Field Documentation

3.26.2.1 next

```
struct node_prefix* next
```

3.26.2.2 prefix

```
union node_prefix::prefix prefix
```

3.26.2.3 type

```
enum node_prefix::type_prefix type
```

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

- src/ast/ast.h

3.27 node_redirection Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_redirection:

Data Fields

- unsigned int type
- char * left
- char * right
- struct node_redirection * next

3.27.1 Field Documentation

3.27.1.1 left

```
char* left
```

3.27.1.2 next

```
struct node_redirection* next
```

3.27.1.3 right

```
char* right
```

3.27.1.4 type

```
unsigned int type
```

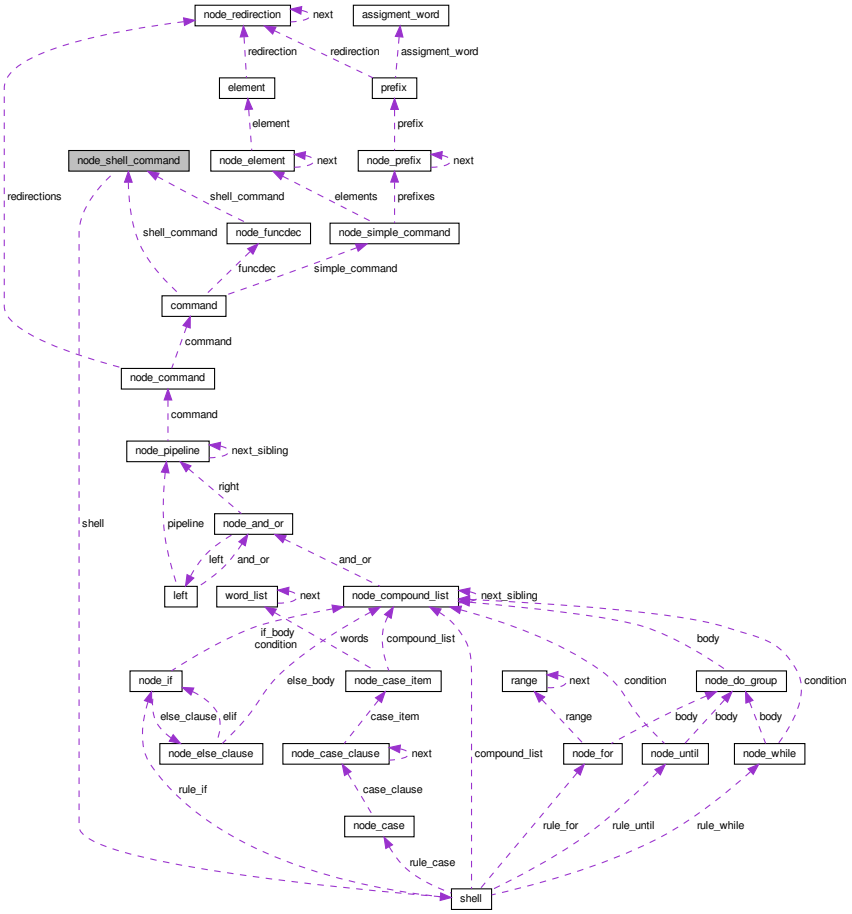
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.28 node_shell_command Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_shell_command:



Data Structures

- union shell

Public Types

- enum type_clause { C_BRACKETS, PARENTHESIS, RULE }
- enum shell_type {
  FOR, WHILE, UNTIL, CASE,
  IF }

Data Fields

- enum node_shell_command::type_clause type
- union node_shell_command::shell shell
- enum node_shell_command::shell_type shell_type

3.28.1 Member Enumeration Documentation

3.28.1.1 shell_type

```
enum shell_type
```

Enumerator

FOR	
WHILE	
UNTIL	
CASE	
IF	

3.28.1.2 type_clause

```
enum type_clause
```

Enumerator

C_BRACKETS	
PARENTHESIS	
RULE	

3.28.2 Field Documentation

3.28.2.1 shell

```
union node_shell_command::shell shell
```

3.28.2.2 shell_type

```
enum node_shell_command::shell_type shell_type
```

3.28.2.3 type

```
enum node_shell_command::type_clause type
```

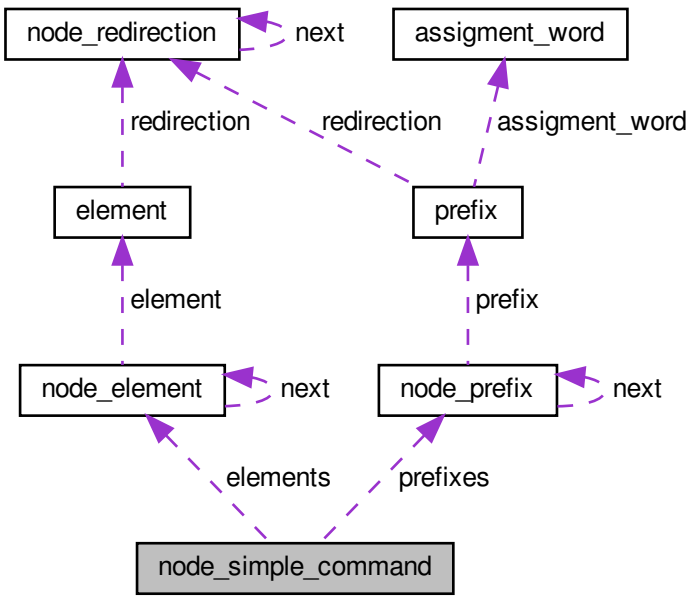
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.29 node_simple_command Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_simple_command:



Data Fields

- bool to_export
- struct node_prefix * prefixes
- struct node_element * elements

3.29.1 Field Documentation

3.29.1.1 elements

```
struct node_element* elements
```

3.29.1.2 prefixes

```
struct node_prefix* prefixes
```

3.29.1.3 to_export

```
bool to_export
```

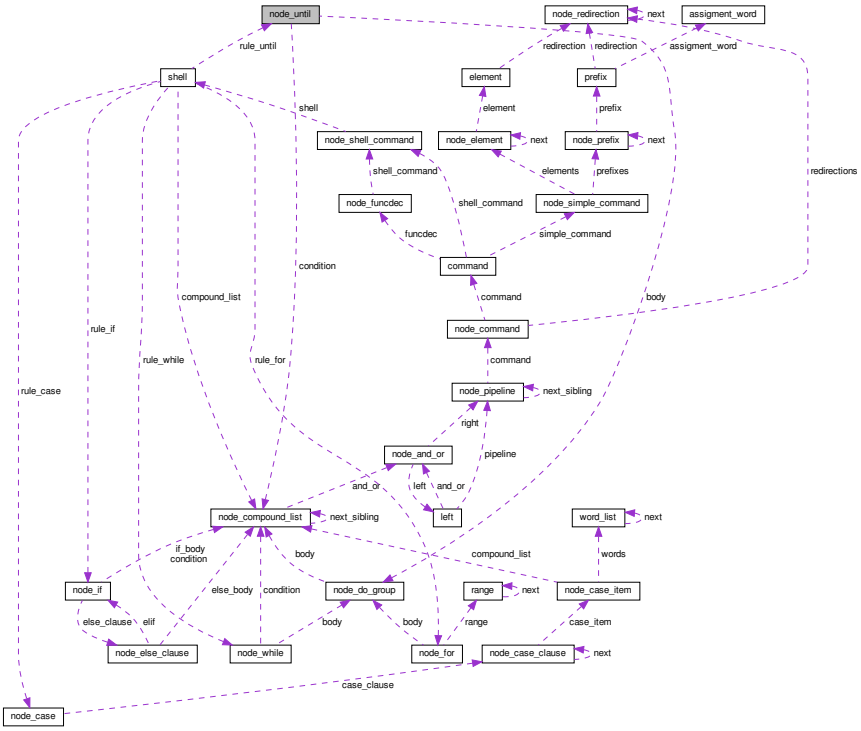
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.30 node_until Struct Reference

```
#include <ast.h>
```

Collaboration diagram for node_until:



Data Fields

- struct node_compound_list * condition
- struct node_do_group * body

3.30.1 Field Documentation

3.30.1.1 body

```
struct node_do_group* body
```

3.30.1.2 condition

```
struct node_compound_list* condition
```

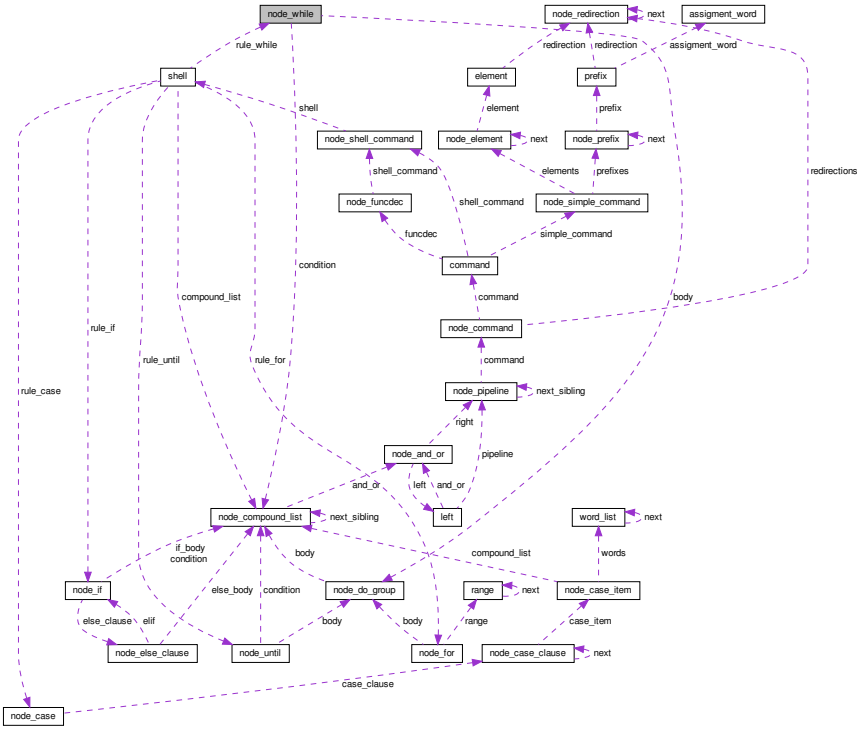
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.31 node_while Struct Reference

`#include <ast.h>`

Collaboration diagram for node_while:



Data Fields

- struct node_compound_list * condition
- struct node_do_group * body

3.31.1 Field Documentation

3.31.1.1 body

`struct node_do_group* body`

3.31.1.2 condition

```
struct node_compound_list* condition
```

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

- src/ast/ast.h

3.32 option_sh Struct Reference

```
#include <main.h>
```

Data Fields

- bool norc_flag
- bool print_ast_flag
- char * cmd

3.32.1 Field Documentation

3.32.1.1 cmd

```
char* cmd
```

3.32.1.2 norc_flag

```
bool norc_flag
```

3.32.1.3 print_ast_flag

```
bool print_ast_flag
```

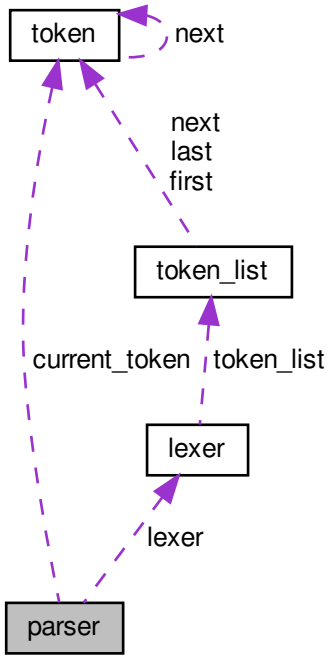
The documentation for this struct was generated from the following file:

- src/main.h

3.33 parser Struct Reference

`#include <ast.h>`

Collaboration diagram for parser:



Data Fields

- struct lexer * lexer
- struct token * current_token

3.33.1 Field Documentation

3.33.1.1 current_token

`struct token* current_token`

3.33.1.2 lexer

```
struct lexer* lexer
```

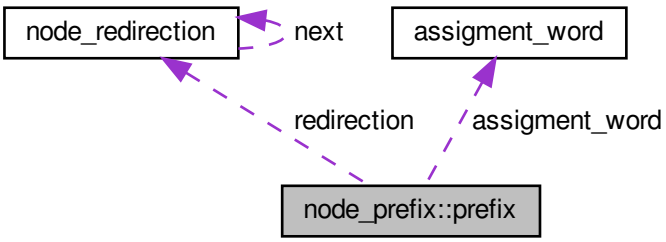
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.34 node_prefix::prefix Union Reference

```
#include <ast.h>
```

Collaboration diagram for node_prefix::prefix:



Data Structures

- struct assigment_word

Data Fields

- struct node_prefix::prefix::assigment_word * assigment_word
- struct node_redirection * redirection

3.34.1 Field Documentation

3.34.1.1 assigment_word

```
struct node_prefix::prefix::assigment_word * assigment_word
```

3.34.1.2 redirection

```
struct node_redirection* redirection
```

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

- src/ast/ast.h

3.35 program_data_storage Struct Reference

```
#include <expansion.h>
```

Data Fields

- char * binary_name
- char ** argv
- int argc
- char * last_cmd_status

3.35.1 Field Documentation

3.35.1.1 argc

```
int argc
```

3.35.1.2 argv

```
char** argv
```

3.35.1.3 binary_name

```
char* binary_name
```

3.35.1.4 last_cmd_status

```
char* last_cmd_status
```

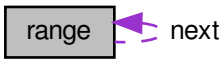
The documentation for this struct was generated from the following file:

- src/expansion/expansion.h

3.36 range Struct Reference

```
#include <ast.h>
```

Collaboration diagram for range:



Data Fields

- char * value
- struct range * next

3.36.1 Field Documentation

3.36.1.1 next

```
struct range* next
```

3.36.1.2 value

```
char* value
```

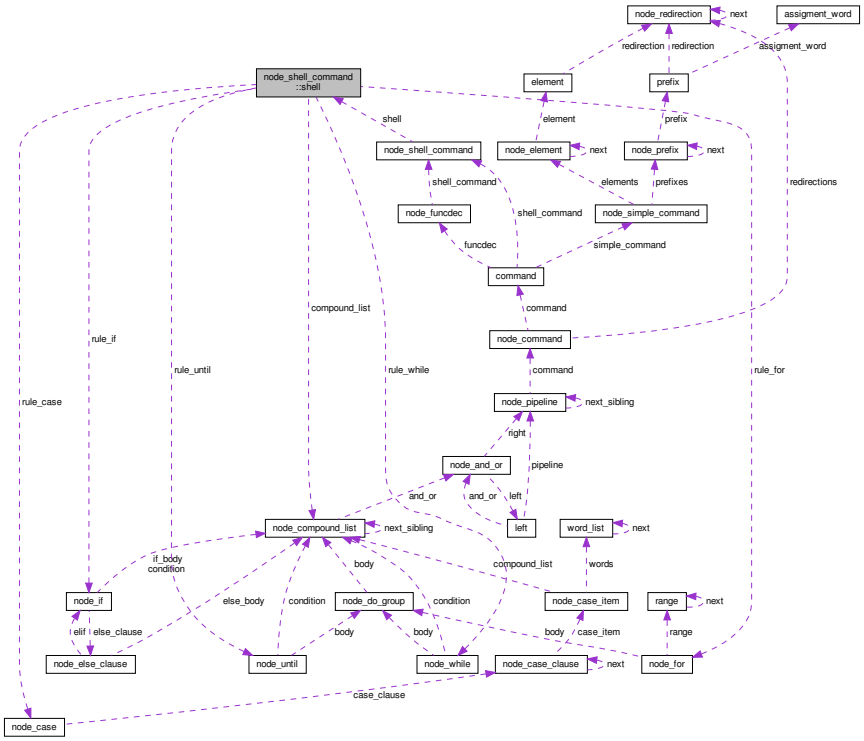
The documentation for this struct was generated from the following file:

- src/ast/ast.h

3.37 node_shell_command::shell Union Reference

`#include <ast.h>`

Collaboration diagram for node_shell_command::shell:



Data Fields

- struct node_compound_list * compound_list
- struct node_for * rule_for
- struct node_while * rule_while
- struct node_until * rule_until
- struct node_case * rule_case
- struct node_if * rule_if

3.37.1 Field Documentation

3.37.1.1 compound_list

`struct node_compound_list* compound_list`

3.37.1.2 rule_case

```
struct node_case* rule_case
```

3.37.1.3 rule_for

```
struct node_for* rule_for
```

3.37.1.4 rule_if

```
struct node_if* rule_if
```

3.37.1.5 rule_until

```
struct node_until* rule_until
```

3.37.1.6 rule_while

```
struct node_while* rule_while
```

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

- src/ast/ast.h

3.38 std Struct Reference

```
#include <exec.h>
```

Data Fields

- char * in
- char * out
- char * err

3.38.1 Field Documentation

3.38.1.1 err

```
char* err
```

3.38.1.2 in

```
char* in
```

3.38.1.3 out

```
char* out
```

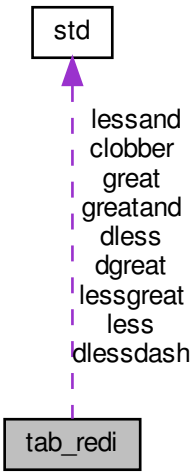
The documentation for this struct was generated from the following file:

- src/exec/exec.h

3.39 tab_redi Struct Reference

```
#include <exec.h>
```

Collaboration diagram for tab_redi:



Data Fields

- struct std dless
- struct std lessgreat
- struct std lessand
- struct std less
- struct std dgreat
- struct std greatand
- struct std clobber
- struct std great
- struct std dlessdash

3.39.1 Field Documentation

3.39.1.1 clobber

```
struct std clobber
```

3.39.1.2 dgreat

```
struct std dgreat
```

3.39.1.3 dless

```
struct std dless
```

3.39.1.4 dlessdash

```
struct std dlessdash
```

3.39.1.5 great

```
struct std great
```

3.39.1.6 greatand

```
struct std greatand
```

3.39.1.7 less

```
struct std less
```

3.39.1.8 lessand

```
struct std lessand
```

3.39.1.9 lessgreat

```
struct std lessgreat
```

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

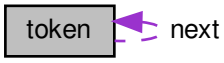
- src/exec/exec.h

3.40 token Struct Reference

Token struct declaration.

```
#include <token.h>
```

Collaboration diagram for token:



Data Fields

- enum token_type type
- char * value
- struct token * next

3.40.1 Detailed Description

Token struct declaration.

Parameters

type	the enum associated to the string.
value	of a token (string) if this token is a word.
next	pointer to the next token in the list.

3.40.2 Field Documentation

3.40.2.1 next

```
struct token* next
```

3.40.2.2 type

```
enum token_type type
```

3.40.2.3 value

```
char* value
```

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

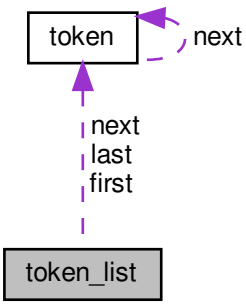
- src/lexer/token.h

3.41 token_list Struct Reference

Basically a lined-list of tokens.

```
#include <token.h>
```

Collaboration diagram for token_list:



Data Fields

- struct token * last
- struct token * first
- struct token * next

3.41.1 Detailed Description

Basically a lined-list of tokens.

Parameters

first	token of the list (used as start point for parsing).
last	token of the list.
next	pointer to the next token in the list.

3.41.2 Field Documentation

3.41.2.1 first

```
struct token* first
```

3.41.2.2 last

```
struct token* last
```

3.41.2.3 next

```
struct token* next
```

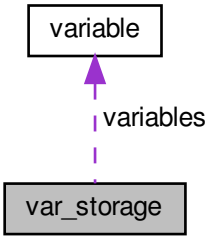
The documentation for this struct was generated from the following file:

- src/lexer/token.h

3.42 var_storage Struct Reference

`#include <var_storage.h>`

Collaboration diagram for var_storage:



Data Fields

- struct variable ** variables

3.42.1 Field Documentation

3.42.1.1 variables

`struct variable** variables`

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

- src/var_storage/var_storage.h

3.43 variable Struct Reference

`#include <var_storage.h>`

Data Fields

- char * key
- char * value
- enum var_type type

3.43.1 Field Documentation

3.43.1.1 key

```
char* key
```

3.43.1.2 type

```
enum var_type type
```

3.43.1.3 value

```
char* value
```

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

- src/var_storage/var_storage.h

3.44 word_list Struct Reference

```
#include <ast.h>
```

Collaboration diagram for word_list:

Data Fields

- char * word
- struct word_list * next

3.44.1 Field Documentation

3.44.1.1 next

```
struct word_list* next
```

3.44.1.2 word

```
char* word
```

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

- src/ast/ast.h

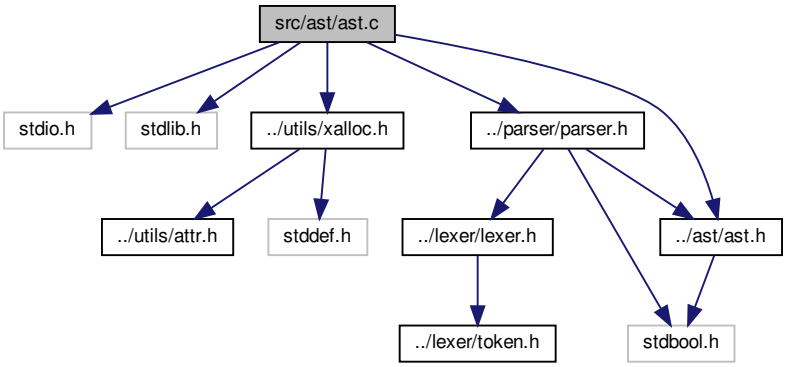
Chapter 4

File Documentation

4.1 src/ast/ast.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../utils/xalloc.h"
#include "../parser/parser.h"
#include "../ast/ast.h"
```

Include dependency graph for ast.c:



Functions

- struct node_input * build_input (void)

  build node input
- struct node_list * build_list (void)

  build node list
- struct node_and_or * build_and_or_final (bool is_and, struct node_pipeline *left, struct node_pipeline *right)

  build node and_or_final
- struct node_and_or * build_and_or_merge (bool is_and, struct node_and_or *left, struct node_pipeline *right)

build node_and_or_merge

- struct node_pipeline * build_pipeline (bool is_not)

  build node pipeline
- struct node_command * build_command (void)

  build command
- struct node_simple_command * build_simple_command (void)

  build simple command
- struct node_shell_command * build_shell_command (struct parser *parser)

  build shell command
- struct node_funcdec * build_funcdec ()

  build node funcdec
- struct node_redirection * build_redirection (struct parser *parser)

  build node redirection
- struct node_prefix * build_prefix (struct parser *parser)

  build node prefix
- struct node_element * build_element (struct parser *parser)

  build node element
- struct node_compound_list * build_compound_list (void)

  build node compound list
- struct node_while * build_while (void)

  build node while
- struct node_until * build_until (void)

  build node until
- struct node_case * build_case (struct parser *parser)

  build node case
- struct node_if * build_if (void)

  build node if
- struct node_for * build_for (void)

  build node for
- struct node_else_clause * build_else_clause (struct parser *parser)

  build node else clause
- struct node_do_group * build_do_group (void)

  build do group
- struct node_case_clause * build_case_clause (void)

  build node case clause
- struct node_case_item * build_case_item (void)

  build node case item

4.1.1 Function Documentation

4.1.1.1 build_and_or_final()

```
struct node_and_or* build_and_or_final (
            bool is_and,
            struct node_pipeline * left,
            struct node_pipeline * right )
```

build node and_or_final

Parameters

is_and	
left	
right	

Returns

struct node_and_or*

4.1.1.2 build_and_or_merge()

```
struct node_and_or* build_and_or_merge (
            bool is_and,
            struct node_and_or * left,
            struct node_pipeline * right )
```

build node_and_or_merge

Parameters

is_and	
left	
right	

Returns

struct node_and_or*

4.1.1.3 build_case()

```
struct node_case* build_case (
            struct parser * parser )
```

build node case

Parameters

parser	

Returns

struct node_case*

4.1.1.4 build_case_clause()

```
struct node_case_clause* build_case_clause (
            void  )
```

build node case clause

Returns

struct node_case_clause*

4.1.1.5 build_case_item()

```
struct node_case_item* build_case_item (
            void  )
```

build node case item

Returns

struct node_case_item*

4.1.1.6 build_command()

```
struct node_command* build_command (
            void  )
```

build command

Returns

struct node_command*

4.1.1.7 build_compound_list()

```
struct node_compound_list* build_compound_list (
            void  )
```

build node compound list

Returns

struct node_compound_list*

4.1.1.8 build_do_group()

```
struct node_do_group* build_do_group (
            void  )
```

build do group

Returns

struct node_do_group*

4.1.1.9 build_element()

```
struct node_element* build_element (
            struct parser * parser )
```

build node element

Parameters

parser	

Returns

struct node_element*

4.1.1.10 build_else_clause()

```
struct node_else_clause* build_else_clause (
            struct parser * parser )
```

build node else clause

Parameters

parser	

Returns

struct node_else_clause*

4.1.1.11 build_for()

```
struct node_for* build_for (
            void  )
```

build node for

Returns

struct node_for*

4.1.1.12 build_funcdec()

```
struct node_funcdec* build_funcdec ( )
```

build node funcdec

Returns

struct node_funcdec*

4.1.1.13 build_if()

```
struct node_if* build_if (
            void  )
```

build node if

Returns

struct node_if*

4.1.1.14 build_input()

```
struct node_input* build_input (
            void  )
```

build node input

Returns

struct node_input*

4.1.1.15 build_list()

```
struct node_list* build_list (
            void  )
```

build node list

Returns

struct node_list*

4.1.1.16 build_pipeline()

```
struct node_pipeline* build_pipeline (
            bool is_not )
```

build node pipeline

Parameters

is_not	

Returns

struct node_pipeline*

4.1.1.17 build_prefix()

```
struct node_prefix* build_prefix (
            struct parser * parser )
```

build node prefix

Parameters

parser	

Returns

struct node_prefix*

4.1.1.18 build_redirection()

```
struct node_redirection* build_redirection (
            struct parser * parser )
```

build node redirection

Parameters

parser	

Returns

struct node_redirection*

4.1.1.19 build_shell_command()

```
struct node_shell_command* build_shell_command (
            struct parser * parser )
```

build shell command

Parameters

parser	

Returns

struct node_shell_command*

4.1.1.20 build_simple_command()

```
struct node_simple_command* build_simple_command (
            void  )
```

build simple command

Returns

struct node_simple_command*

4.1.1.21 build_until()

```
struct node_until* build_until (
            void  )
```

build node until

Returns

struct node_until*

4.1.1.22 build_while()

```
struct node_while* build_while (
            void  )
```

build node while

Returns

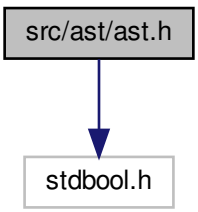
struct node_while*

4.2 src/ast/ast.h File Reference

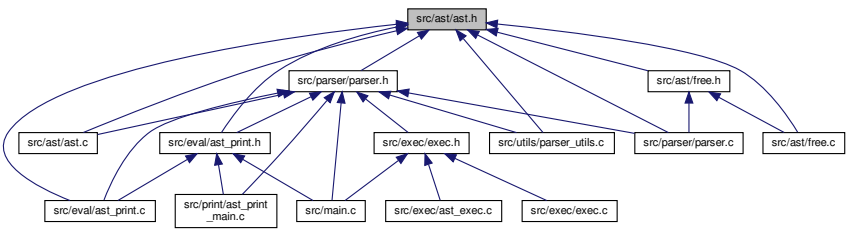
Define ast and parser structures.

```
#include <stdbool.h>
```

Include dependency graph for ast.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct parser
- struct node_input
- struct node_list
- struct node_and_or
- union node_and_or::left
- struct node_pipeline
- struct node_command
- union node_command::command
- struct node_simple_command
- struct node_shell_command
- union node_shell_command::shell
- struct node_funcdec
- struct node_redirection
- struct node_prefix
- union node_prefix::prefix
- struct node_prefix::prefix::assigment_word
- struct node_element
- union node_element::element
- struct node_compound_list
- struct node_while
- struct node_until
- struct node_case
- struct node_if
- struct range
- struct node_for
- struct node_else_clause
- struct node_do_group
- struct node_case_clause
- struct word_list
- struct node_case_item

Functions

- struct node_input * build_input (void)

  build node input
- struct node_list * build_list (void)

  build node list
- struct node_and_or * build_and_or_final (bool is_and, struct node_pipeline *left, struct node_pipeline *right)

  build node and_or_final
- struct node_and_or * build_and_or_merge (bool is_and, struct node_and_or *left, struct node_pipeline *right)

  build node_and_or_merge
- struct node_pipeline * build_pipeline (bool is_not)

  build node pipeline
- struct node_command * build_command (void)

  build command
- struct node_simple_command * build_simple_command (void)

  build simple command
- struct node_shell_command * build_shell_command (struct parser *parser)

  build shell command
- struct node_funcdec * build_funcdec ()

build node funcdec

- struct node_redirection * build_redirection (struct parser *parser)

  build node redirection
- struct node_prefix * build_prefix (struct parser *parser)

  build node prefix
- struct node_element * build_element (struct parser *parser)

  build node element
- struct node_compound_list * build_compound_list (void)

  build node compound list
- struct node_while * build_while (void)

  build node while
- struct node_until * build_until (void)

  build node until
- struct node_case * build_case (struct parser *parser)

  build node case
- struct node_if * build_if (void)

  build node if
- struct node_for * build_for (void)

  build node for
- struct node_else_clause * build_else_clause (struct parser *parser)

  build node else clause
- struct node_do_group * build_do_group (void)

  build do group
- struct node_case_clause * build_case_clause (void)

  build node case clause
- struct node_case_item * build_case_item (void)

  build node case item

4.2.1 Detailed Description

Define ast and parser structures.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

4.2.2 Function Documentation

4.2.2.1 build_and_or_final()

```
struct node_and_or* build_and_or_final (
            bool is_and,
            struct node_pipeline * left,
            struct node_pipeline * right )
```

build node and_or_final

Parameters

is_and	
left	
right	

Returns

struct node_and_or*

4.2.2.2 build_and_or_merge()

```
struct node_and_or* build_and_or_merge (
            bool is_and,
            struct node_and_or * left,
            struct node_pipeline * right )
```

build node_and_or_merge

Parameters

is_and	
left	
right	

Returns

struct node_and_or*

4.2.2.3 build_case()

```
struct node_case* build_case (
            struct parser * parser )
```

build node case

Parameters

parser	

Returns

struct node_case*

4.2.2.4 build_case_clause()

```
struct node_case_clause* build_case_clause (
            void  )
```

build node case clause

Returns

struct node_case_clause*

4.2.2.5 build_case_item()

```
struct node_case_item* build_case_item (
            void  )
```

build node case item

Returns

struct node_case_item*

4.2.2.6 build_command()

```
struct node_command* build_command (
            void  )
```

build command

Returns

struct node_command*

4.2.2.7 build_compound_list()

```
struct node_compound_list* build_compound_list (
            void  )
```

build node compound list

Returns

struct node_compound_list*

4.2.2.8 build_do_group()

```
struct node_do_group* build_do_group (
            void  )
```

build do group

Returns

struct node_do_group*

4.2.2.9 build_element()

```
struct node_element* build_element (
            struct parser * parser )
```

build node element

Parameters

parser	

Returns

struct node_element*

4.2.2.10 build_else_clause()

```
struct node_else_clause* build_else_clause (
            struct parser * parser )
```

build node else clause

Parameters

parser	

Returns

struct node_else_clause*

4.2.2.11 build_for()

```
struct node_for* build_for (
            void  )
```

build node for

Returns

struct node_for*

4.2.2.12 build_funcdec()

```
struct node_funcdec* build_funcdec ( )
```

build node funcdec

Returns

struct node_funcdec*

4.2.2.13 build_if()

```
struct node_if* build_if (
            void  )
```

build node if

Returns

struct node_if*

4.2.2.14 build_input()

```
struct node_input* build_input (
            void  )
```

build node input

Returns

struct node_input*

4.2.2.15 build_list()

```
struct node_list* build_list (
            void  )
```

build node list

Returns

struct node_list*

4.2.2.16 build_pipeline()

```
struct node_pipeline* build_pipeline (
            bool is_not )
```

build node pipeline

Parameters

is_not	

Returns

struct node_pipeline*

4.2.2.17 build_prefix()

```
struct node_prefix* build_prefix (
            struct parser * parser )
```

build node prefix

Parameters

parser	

Returns

struct node_prefix*

4.2.2.18 build_redirection()

```
struct node_redirection* build_redirection (
            struct parser * parser )
```

build node redirection

Parameters

parser	

Returns

struct node_redirection*

4.2.2.19 build_shell_command()

```
struct node_shell_command* build_shell_command (
            struct parser * parser )
```

build shell command

Parameters

parser	

Returns

struct node_shell_command*

4.2.2.20 build_simple_command()

```
struct node_simple_command* build_simple_command (
            void  )
```

build simple command

Returns

struct node_simple_command*

4.2.2.21 build_until()

```
struct node_until* build_until (
            void  )
```

build node until

Returns

struct node_until*

4.2.2.22 build_while()

```
struct node_while* build_while (
            void  )
```

build node while

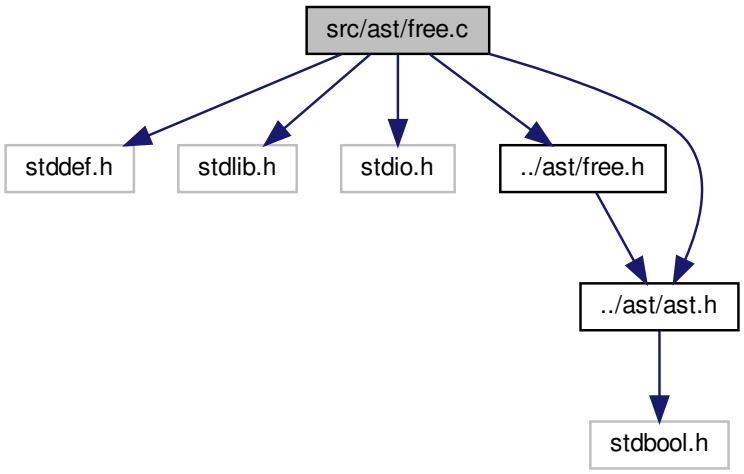
Returns

struct node_while*

4.3 src/ast/free.c File Reference

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include "../ast/free.h"
#include "../ast/ast.h"
```

Include dependency graph for free.c:



Macros

- #define AST_EXISTS(ast)
- #define FREE_AST(ast)
- #define DEBUG_FLAG false
- #define DEBUG(msg)

Functions

- void free_input (struct node_input *ast)
- void free_and_or (struct node_and_or *ast)

  free and/or node
- void free_redirection (struct node_redirection *ast)

  free redirection node
- void free_prefix (struct node_prefix *ast)

  free prefix node
- void free_element (struct node_element *ast)

  free element node
- void free_until (struct node_until *ast)

  free until node
- void free_if (struct node_if *ast)

  free if node
- void free_else_clause (struct node_else_clause *ast)

  free else clause node
- void free_do_group (struct node_do_group *ast)

  free do group node
- void free_case_clause (struct node_case_clause *ast)

  free case clause
- void free_case_item (struct node_case_item *ast)

  free case item node
- void free_command (struct node_command *ast)
- void free_simple_command (struct node_simple_command *ast)

  free simple command node
- void free_pipeline (struct node_pipeline *ast)

  free pipeline node
- void free_list (struct node_list *ast)

  free list node
- void free_shell_command (struct node_shell_command *ast)

  free shell command node
- void free_compound_list (struct node_compound_list *ast)

  free compound list node
- void free_range (struct range *range)
- void free_for (struct node_for *ast)

  free for node
- void free_while (struct node_while *ast)

  free while node
- void free_case (struct node_case *ast)

  free case node
- void free_funcdec (struct node_funcdec *ast)

  free funcdec node

4.3.1 Macro Definition Documentation

4.3.1.1 AST_EXISTS

```
#define AST_EXISTS(
              ast )
```

Value:

```
if (!ast)\
        return;
```

4.3.1.2 DEBUG

```
#define DEBUG(
              msg )
```

Value:

```
if (DEBUG_FLAG) \
                      printf("%s", msg);
```

4.3.1.3 DEBUG_FLAG

```
#define DEBUG_FLAG false
```

4.3.1.4 FREE_AST

```
#define FREE_AST(
              ast )
```

Value:

```
free(ast); \
    ast = NULL;
```

4.3.2 Function Documentation

4.3.2.1 free_and_or()

```
void free_and_or (
             struct node_and_or * ast )
```

free and/or node

Parameters

ast	

4.3.2.2 free_case()

```
void free_case (
            struct node_case * ast )
```

free case node

Parameters

ast	

4.3.2.3 free_case_clause()

```
void free_case_clause (
            struct node_case_clause * ast )
```

free case clause

Parameters

ast	

4.3.2.4 free_case_item()

```
void free_case_item (
            struct node_case_item * ast )
```

free case item node

Parameters

ast	

4.3.2.5 free_command()

```
void free_command (
```

```
            struct node_command * ast )
```

Parameters

ast	

4.3.2.6 free_compound_list()

```
void free_compound_list (
            struct node_compound_list * ast )
```

free compound list node

Parameters

ast	

4.3.2.7 free_do_group()

```
void free_do_group (
            struct node_do_group * ast )
```

free do group node

Parameters

ast	

4.3.2.8 free_element()

```
void free_element (
            struct node_element * ast )
```

free element node

Parameters

ast	

4.3.2.9 free_else_clause()

```
void free_else_clause (
            struct node_else_clause * ast )
```

free else clause node

Parameters

ast	

4.3.2.10 free_for()

```
void free_for (
            struct node_for * ast )
```

free for node

Parameters

ast	

4.3.2.11 free_funcdec()

```
void free_funcdec (
            struct node_funcdec * ast )
```

free funcdec node

Parameters

ast	

4.3.2.12 free_if()

```
void free_if (
            struct node_if * ast )
```

free if node

Parameters

ast	

4.3.2.13 free_input()

```
void free_input (
            struct node_input * ast )
```

Parameters

ast	

4.3.2.14 free_list()

```
void free_list (
            struct node_list * ast )
```

free list node

Parameters

ast	

4.3.2.15 free_pipeline()

```
void free_pipeline (
            struct node_pipeline * ast )
```

free pipeline node

Parameters

ast	

4.3.2.16 free_prefix()

```
void free_prefix (
            struct node_prefix * ast )
```

free prefix node

Parameters

ast	

4.3.2.17 free_range()

```
void free_range (
            struct range * range )
```

4.3.2.18 free_redirection()

```
void free_redirection (
            struct node_redirection * ast )
```

free redirection node

Parameters

ast	

4.3.2.19 free_shell_command()

```
void free_shell_command (
            struct node_shell_command * ast )
```

free shell command node

Parameters

ast	

4.3.2.20 free_simple_command()

```
void free_simple_command (
            struct node_simple_command * ast )
```

free simple command node

Parameters

ast	

4.3.2.21 free_until()

```
void free_until (
            struct node_until * ast )
```

free until node

Parameters

ast	

4.3.2.22 free_while()

```
void free_while (
            struct node_while * ast )
```

free while node

Parameters

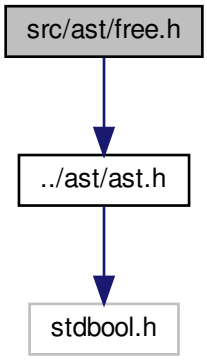
ast	

4.4 src/ast/free.h File Reference

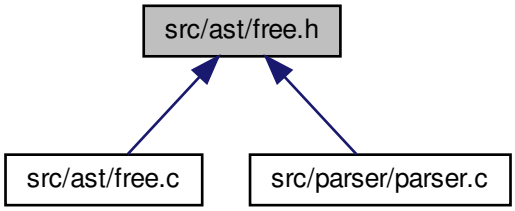
Free functions.

```
#include "../ast/ast.h"
```

Include dependency graph for free.h:



This graph shows which files directly or indirectly include this file:



Functions

- void free_input (struct node_input *ast)
- void free_list (struct node_list *ast)

  free list node
- void free_and_or (struct node_and_or *ast)

  free and/or node
- void free_pipeline (struct node_pipeline *ast)

  free pipeline node
- void free_command (struct node_command *ast)
- void free_simple_command (struct node_simple_command *ast)

  free simple command node
- void free_shell_command (struct node_shell_command *ast)

  free shell command node
- void free_funcdec (struct node_funcdec *ast)

free funcdec node

- void free_redirection (struct node_redirection *ast)

  free redirection node
- void free_prefix (struct node_prefix *ast)

  free prefix node
- void free_element (struct node_element *ast)

  free element node
- void free_compound_list (struct node_compound_list *ast)

  free compound list node
- void free_while (struct node_while *ast)

  free while node
- void free_until (struct node_until *ast)

  free until node
- void free_case (struct node_case *ast)

  free case node
- void free_if (struct node_if *ast)

  free if node
- void free_for (struct node_for *ast)

  free for node
- void free_else_clause (struct node_else_clause *ast)

  free else clause node
- void free_do_group (struct node_do_group *ast)

  free do group node
- void free_case_clause (struct node_case_clause *ast)

  free case clause
- void free_case_item (struct node_case_item *ast)

  free case item node

4.4.1 Detailed Description

Free functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.4.2 Function Documentation

4.4.2.1 free_and_or()

```
void free_and_or (
            struct node_and_or * ast )
```

free and/or node

Parameters

ast	

4.4.2.2 free_case()

```
void free_case (
            struct node_case * ast )
```

free case node

Parameters

ast	

4.4.2.3 free_case_clause()

```
void free_case_clause (
            struct node_case_clause * ast )
```

free case clause

Parameters

ast	

4.4.2.4 free_case_item()

```
void free_case_item (
            struct node_case_item * ast )
```

free case item node

Parameters

ast	

4.4.2.5 free_command()

```
void free_command (
```

```
            struct node_command * ast )
```

Parameters

ast	

4.4.2.6 free_compound_list()

```
void free_compound_list (
            struct node_compound_list * ast )
```

free compound list node

Parameters

ast	

4.4.2.7 free_do_group()

```
void free_do_group (
            struct node_do_group * ast )
```

free do group node

Parameters

ast	

4.4.2.8 free_element()

```
void free_element (
            struct node_element * ast )
```

free element node

Parameters

ast	

4.4.2.9 free_else_clause()

```
void free_else_clause (
            struct node_else_clause * ast )
```

free else clause node

Parameters

ast	

4.4.2.10 free_for()

```
void free_for (
            struct node_for * ast )
```

free for node

Parameters

ast	

4.4.2.11 free_funcdec()

```
void free_funcdec (
            struct node_funcdec * ast )
```

free funcdec node

Parameters

ast	

4.4.2.12 free_if()

```
void free_if (
            struct node_if * ast )
```

free if node

Parameters

ast	

4.4.2.13 free_input()

```
void free_input (
            struct node_input * ast )
```

Parameters

ast	

4.4.2.14 free_list()

```
void free_list (
            struct node_list * ast )
```

free list node

Parameters

ast	

4.4.2.15 free_pipeline()

```
void free_pipeline (
            struct node_pipeline * ast )
```

free pipeline node

Parameters

ast	

4.4.2.16 free_prefix()

```
void free_prefix (
            struct node_prefix * ast )
```

free prefix node

Parameters

ast	

4.4.2.17 free_redirection()

```
void free_redirection (
            struct node_redirection * ast )
```

free redirection node

Parameters

ast	

4.4.2.18 free_shell_command()

```
void free_shell_command (
            struct node_shell_command * ast )
```

free shell command node

Parameters

ast	

4.4.2.19 free_simple_command()

```
void free_simple_command (
            struct node_simple_command * ast )
```

free simple command node

Parameters

ast	

4.4.2.20 free_until()

```
void free_until (
            struct node_until * ast )
```

free until node

Parameters

ast	

4.4.2.21 free_while()

```
void free_while (
            struct node_while * ast )
```

free while node

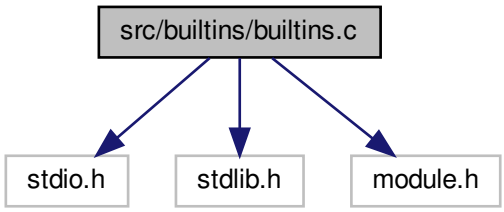
Parameters

ast	

4.5 src/builtins/builtins.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "module.h"
```

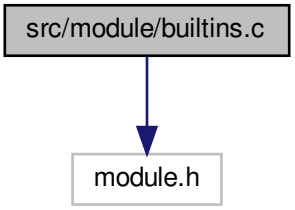
Include dependency graph for builtins.c:



4.6 src/module/builtins.c File Reference

```
#include "module.h"
```

Include dependency graph for builtins.c:



Functions

- bool parse_export ()

4.6.1 Function Documentation

4.6.1.1 parse_export()

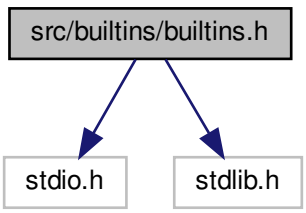
```
bool parse_export ( )
```

4.7 src/builtins/builtins.h File Reference

Builtin functions.

```
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for builtins.h:



Functions

- bool parse_export ()

4.7.1 Detailed Description

Builtin functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.7.2 Function Documentation

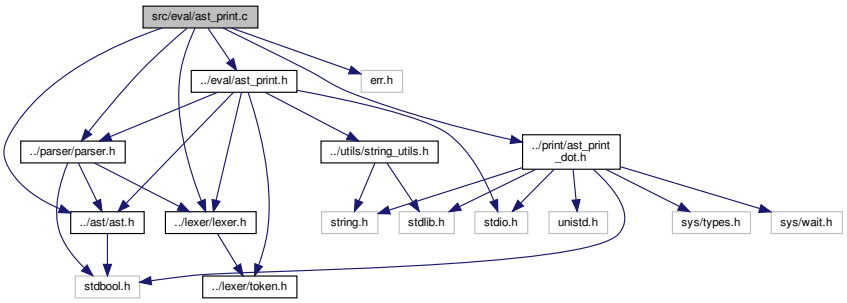
4.7.2.1 parse_export()

```
bool parse_export ( )
```

4.8 src/eval/ast_print.c File Reference

```
#include "../ast/ast.h"
#include "../lexer/lexer.h"
#include "../parser/parser.h"
#include "../eval/ast_print.h"
#include <err.h>
#include "../print/ast_print_dot.h"
```

Include dependency graph for ast_print.c:



Macros

- #define PRINT_FLAG false
- #define PRINT_NODE(msg)

Functions

- void print_node_input (struct node_input *ast, FILE *f)
  print node_input
- void print_node_list (struct node_list *ast, FILE *f)
  print node list
- void print_node_and_or (struct node_and_or *ast, FILE *f, void *node)
  print node_and_or
- void print_node_pipeline (struct node_pipeline *ast, FILE *f, void *node)
  print node pipeline
- void print_node_command (struct node_command *ast, FILE *f, void *node)
  print node command
- void print_node_simple_command (struct node_simple_command *ast, FILE *f, void *node)
  print note simple command
- void print_node_shell_command (struct node_shell_command *ast, FILE *f, void *node)
  print note shell command
- void print_node_funcdec (struct node_funcdec *ast, FILE *f, void *node)
  print node funcdec
- void print_node_redirection (struct node_redirection *ast, FILE *f, void *node)
  print node redirection
- void print_node_prefix (struct node_prefix *ast, FILE *f, void *node)
  print node prefix
- void print_node_element (struct node_element *ast, FILE *f, void *node)
  print node element
- void print_node_compound_list (struct node_compound_list *ast, FILE *f, void *node)
  print node compound list
- void print_node_while (struct node_while *ast, FILE *f, void *node)
  print node while
- void print_node_until (struct node_until *ast, FILE *f, void *node)
  print node until
- void print_node_case (struct node_case *ast, FILE *f, void *node)
  print node case
- void print_node_if (struct node_if *ast, FILE *f, void *node)
  print node if
- void print_node_elif (struct node_if *ast, FILE *f, void *node)
  print node elif
- void print_node_for (struct node_for *ast, FILE *f, void *node)
  print node for
- void print_node_else_clause (struct node_else_clause *ast, FILE *f, void *node)
  print node else clause
- void print_node_do_group (struct node_do_group *ast, FILE *f, void *node)
  print node do group
- void print_node_case_clause (struct node_case_clause *ast, FILE *f, void *node)
  print node do group
- void print_node_case_item (struct node_case_item *ast, FILE *f, void *node)
  print node case_item
- void print_ast (struct node_input *ast)
  print ast

4.8.1 Macro Definition Documentation

4.8.1.1 PRINT_FLAG

```
#define PRINT_FLAG false
```

4.8.1.2 PRINT_NODE

```
#define PRINT_NODE(
              msg )
```

Value:

```
if (PRINT_FLAG) \
           fprintf(f, "%s\n", msg)
```

4.8.2 Function Documentation

4.8.2.1 print_ast()

```
void print_ast (
             struct node_input * ast )
```

print ast

Parameters

ast	

Returns

* void

4.8.2.2 print_node_and_or()

```
void print_node_and_or (
             struct node_and_or * ast,
             FILE * f,
             void * node )
```

print node_and_or

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.3 print_node_case()

```
void print_node_case (
            struct node_case * ast,
            FILE * f,
            void * node )
```

print node case

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.4 print_node_case_clause()

```
void print_node_case_clause (
            struct node_case_clause * ast,
            FILE * f,
            void * node )
```

print node do group

Parameters

ast	
f	
node	

Returns

* void

4.8.2.5 print_node_case_item()

```
void print_node_case_item (
            struct node_case_item * ast,
            FILE * f,
            void * node )
```

print node case_item

Parameters

ast	
f	
node	

Returns

* void

4.8.2.6 print_node_command()

```
void print_node_command (
            struct node_command * ast,
            FILE * f,
            void * node )
```

print node command

Parameters

ast	
f	
node	

Returns

* void

4.8.2.7 print_node_compound_list()

```
void print_node_compound_list (
            struct node_compound_list * ast,
            FILE * f,
            void * node )
```

print node compound list

Parameters

ast	
f	
node	

Returns

* void

4.8.2.8 print_node_do_group()

```
void print_node_do_group (
            struct node_do_group * ast,
            FILE * f,
            void * node )
```

print node do group

Parameters

ast	
f	
node	

Returns

* void

4.8.2.9 print_node_element()

```
void print_node_element (
            struct node_element * ast,
            FILE * f,
            void * node )
```

print node element

Parameters

ast	
f	
node	

Returns

* void

4.8.2.10 print_node_elif()

```
void print_node_elif (
            struct node_if * ast,
            FILE * f,
            void * node )
```

print node elif

Parameters

ast	
f	
node	

Returns

* void

4.8.2.11 print_node_else_clause()

```
void print_node_else_clause (
            struct node_else_clause * ast,
            FILE * f,
            void * node )
```

print node else clause

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.12 print_node_for()

```
void print_node_for (
            struct node_for * ast,
            FILE * f,
            void * node )
```

print node for

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.13 print_node_funcdec()

```
void print_node_funcdec (
            struct node_funcdec * ast,
            FILE * f,
            void * node )
```

print node funcdec

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.14 print_node_if()

```
void print_node_if (
            struct node_if * ast,
            FILE * f,
            void * node )
```

print node if

Parameters

ast	
f	
node	

Returns

* void

4.8.2.15 print_node_input()

```
void print_node_input (
            struct node_input * ast,
            FILE * f )
```

print node_input

Parameters

ast	
f	

4.8.2.16 print_node_list()

```
void print_node_list (
            struct node_list * ast,
            FILE * f )
```

print node list

Parameters

ast	
f	

4.8.2.17 print_node_pipeline()

```
void print_node_pipeline (
            struct node_pipeline * ast,
            FILE * f,
            void * node )
```

print node pipeline

Parameters

ast	
f	
node	

Returns

* void

4.8.2.18 print_node_prefix()

```
void print_node_prefix (
            struct node_prefix * ast,
            FILE * f,
            void * node )
```

print node prefix

Parameters

ast	
f	
node	

Returns

* void

4.8.2.19 print_node_redirection()

```
void print_node_redirection (
            struct node_redirection * ast,
```

```
            FILE * f,
            void * node )
```

print node redirection

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.20 print_node_shell_command()

```
void print_node_shell_command (
            struct node_shell_command * ast,
            FILE * f,
            void * node )
```

print note shell command

Parameters

ast	
f	
node	

Returns

∗ void

4.8.2.21 print_node_simple_command()

```
void print_node_simple_command (
            struct node_simple_command * ast,
            FILE * f,
            void * node )
```

print note simple command

Parameters

ast	
f	
node	

Returns

* void

4.8.2.22 print_node_until()

```
void print_node_until (
            struct node_until * ast,
            FILE * f,
            void * node )
```

print node until

Parameters

ast	
f	
node	

Returns

* void

4.8.2.23 print_node_while()

```
void print_node_while (
            struct node_while * ast,
            FILE * f,
            void * node )
```

print node while

Parameters

ast	
f	
node	

Returns

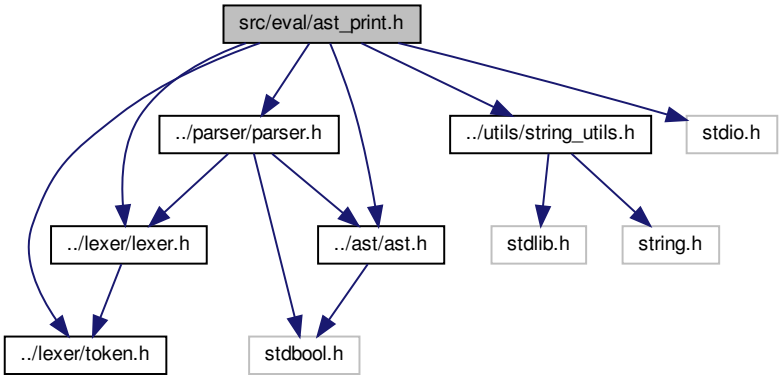
* void

4.9 src/eval/ast_print.h File Reference

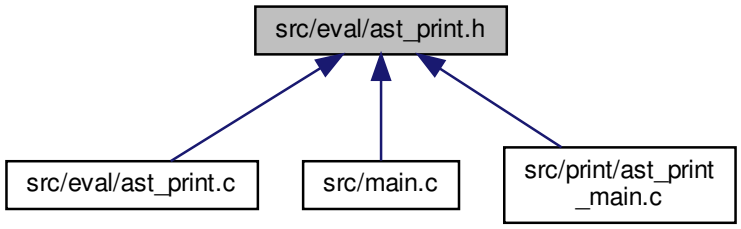
Print functions.

```
#include "../parser/parser.h"
#include "../lexer/lexer.h"
#include "../lexer/token.h"
#include "../utils/string_utils.h"
#include "../ast/ast.h"
#include <stdio.h>
```

Include dependency graph for ast_print.h:



This graph shows which files directly or indirectly include this file:



Functions

- void print_node_input (struct node_input *ast, FILE *f)

  print node_input
- void print_node_list (struct node_list *ast, FILE *f)

  print node list
- void print_node_and_or (struct node_and_or *ast, FILE *f, void *node)

  print node_and_or
- void print_node_pipeline (struct node_pipeline *ast, FILE *f, void *node)

  print node pipeline
- void print_node_command (struct node_command *ast, FILE *f, void *node)

print node command

- void print_node_simple_command (struct node_simple_command *ast, FILE *f, void *node)

  print note simple command
- void print_node_shell_command (struct node_shell_command *ast, FILE *f, void *node)

  print note shell command
- void print_node_funcdec (struct node_funcdec *ast, FILE *f, void *node)

  print node funcdec
- void print_node_redirection (struct node_redirection *ast, FILE *f, void *node)

  print node redirection
- void print_node_prefix (struct node_prefix *ast, FILE *f, void *node)

  print node prefix
- void print_node_element (struct node_element *ast, FILE *f, void *node)

  print node element
- void print_node_compound_list (struct node_compound_list *ast, FILE *f, void *node)

  print node compound list
- void print_node_while (struct node_while *ast, FILE *f, void *node)

  print node while
- void print_node_until (struct node_until *ast, FILE *f, void *node)

  print node until
- void print_node_case (struct node_case *ast, FILE *f, void *node)

  print node case
- void print_node_if (struct node_if *ast, FILE *f, void *node)

  print node if
- void print_node_elif (struct node_if *ast, FILE *f, void *node)

  print node elif
- void print_node_for (struct node_for *ast, FILE *f, void *node)

  print node for
- void print_node_else_clause (struct node_else_clause *ast, FILE *f, void *node)

  print node else clause
- void print_node_do_group (struct node_do_group *ast, FILE *f, void *node)

  print node do group
- void print_node_case_clause (struct node_case_clause *ast, FILE *f, void *node)

  print node do group
- void print_node_case_item (struct node_case_item *ast, FILE *f, void *node)

  print node case_item
- void print_ast (struct node_input *ast)

  print ast

4.9.1 Detailed Description

Print functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.9.2 Function Documentation

4.9.2.1 print_ast()

```
void print_ast (
            struct node_input * ast )
```

print ast

Parameters

ast	

Returns

* void

4.9.2.2 print_node_and_or()

```
void print_node_and_or (
            struct node_and_or * ast,
            FILE * f,
            void * node )
```

print node_and_or

Parameters

ast	
f	
node	

Returns

* void

4.9.2.3 print_node_case()

```
void print_node_case (
            struct node_case * ast,
            FILE * f,
            void * node )
```

print node case

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.4 print_node_case_clause()

```
void print_node_case_clause (
            struct node_case_clause * ast,
            FILE * f,
            void * node )
```

print node do group

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.5 print_node_case_item()

```
void print_node_case_item (
            struct node_case_item * ast,
            FILE * f,
            void * node )
```

print node case_item

Parameters

ast	
f	
node	

Returns

* void

4.9.2.6 print_node_command()

```
void print_node_command (
            struct node_command * ast,
            FILE * f,
            void * node )
```

print node command

Parameters

ast	
f	
node	

Returns

* void

4.9.2.7 print_node_compound_list()

```
void print_node_compound_list (
            struct node_compound_list * ast,
            FILE * f,
            void * node )
```

print node compound list

Parameters

ast	
f	
node	

Returns

* void

4.9.2.8 print_node_do_group()

```
void print_node_do_group (
            struct node_do_group * ast,
            FILE * f,
            void * node )
```

print node do group

Parameters

ast	
f	
node	

Returns

* void

4.9.2.9 print_node_element()

```
void print_node_element (
            struct node_element * ast,
            FILE * f,
            void * node )
```

print node element

Parameters

ast	
f	
node	

Returns

* void

4.9.2.10 print_node_elif()

```
void print_node_elif (
            struct node_if * ast,
            FILE * f,
            void * node )
```

print node elif

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.11 print_node_else_clause()

```
void print_node_else_clause (
            struct node_else_clause * ast,
            FILE * f,
            void * node )
```

print node else clause

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.12 print_node_for()

```
void print_node_for (
            struct node_for * ast,
            FILE * f,
            void * node )
```

print node for

Parameters

ast	
f	
node	

Returns

* void

4.9.2.13 print_node_funcdec()

```
void print_node_funcdec (
            struct node_funcdec * ast,
            FILE * f,
            void * node )
```

print node funcdec

Parameters

ast	
f	
node	

Returns

* void

4.9.2.14 print_node_if()

```
void print_node_if (
            struct node_if * ast,
            FILE * f,
            void * node )
```

print node if

Parameters

ast	
f	
node	

Returns

* void

4.9.2.15 print_node_input()

```
void print_node_input (
            struct node_input * ast,
            FILE * f )
```

print node_input

Parameters

ast	
f	

4.9.2.16 print_node_list()

```
void print_node_list (
            struct node_list * ast,
            FILE * f )
```

print node list

Parameters

ast	
f	

4.9.2.17 print_node_pipeline()

```
void print_node_pipeline (
            struct node_pipeline * ast,
            FILE * f,
            void * node )
```

print node pipeline

Parameters

ast	
f	
node	

Returns

* void

4.9.2.18 print_node_prefix()

```
void print_node_prefix (
            struct node_prefix * ast,
            FILE * f,
            void * node )
```

print node prefix

Parameters

ast	
f	
node	

Returns

* void

4.9.2.19 print_node_redirection()

```
void print_node_redirection (
            struct node_redirection * ast,
            FILE * f,
            void * node )
```

print node redirection

Parameters

ast	
f	
node	

Returns

* void

4.9.2.20 print_node_shell_command()

```
void print_node_shell_command (
            struct node_shell_command * ast,
            FILE * f,
            void * node )
```

print note shell command

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.21 print_node_simple_command()

```
void print_node_simple_command (
            struct node_simple_command * ast,
            FILE * f,
            void * node )
```

print note simple command

Parameters

ast	
f	
node	

Returns

∗ void

4.9.2.22 print_node_until()

```
void print_node_until (
            struct node_until * ast,
            FILE * f,
            void * node )
```

print node until

Parameters

ast	
f	
node	

Returns

* void

4.9.2.23 print_node_while()

```
void print_node_while (
            struct node_while * ast,
            FILE * f,
            void * node )
```

print node while

Parameters

ast	
f	
node	

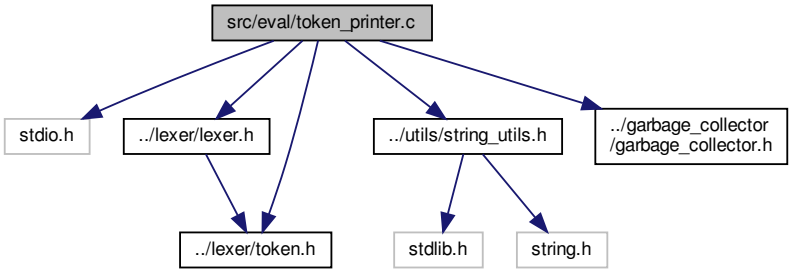
Returns

* void

4.10 src/eval/token_printer.c File Reference

```
#include <stdio.h>
#include "../lexer/lexer.h"
#include "../lexer/token.h"
#include "../utils/string_utils.h"
#include "../garbage_collector/garbage_collector.h"
```

Include dependency graph for token_printer.c:



Functions

- int main (int argc, char *argv[ ])

4.10.1 Function Documentation

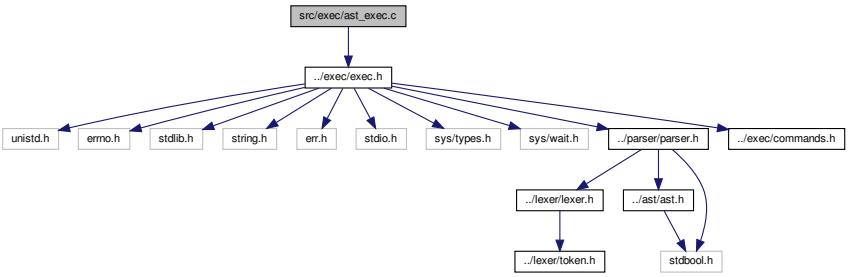
4.10.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

4.11 src/exec/ast_exec.c File Reference

```
#include "../exec/exec.h"
```

Include dependency graph for ast_exec.c:



Functions

- int main (int argc, char ∗argv[ ])

4.11.1 Function Documentation

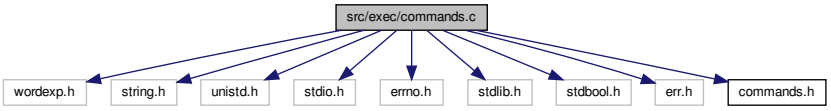
4.11.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

4.12 src/exec/commands.c File Reference

```
#include <wordexp.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <stdbool.h>
#include <err.h>
#include "commands.h"
```

Include dependency graph for commands.c:



Macros

- #define _XOPEN_SOURCE

Functions

- int print_without_sp (char *c)
- void print_echo (char **args, bool e, bool n)
- void echo (char **args)

  implementation of command echo
- void cd (char **args)

  implementation of command cd
- void export (char **args)

  implementation of command export
- void exit_shell (char **args)

  implementation of exit_shell

Variables

- char ** environ

4.12.1 Macro Definition Documentation

4.12.1.1 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE
```

4.12.2 Function Documentation

4.12.2.1 cd()

```
void cd (
            char ** args )
```

implementation of command cd

Parameters

args	

4.12.2.2 echo()

```
void echo (
            char ** args )
```

implementation of command echo

Parameters

args	

4.12.2.3 exit_shell()

```
void exit_shell (
            char ** args )
```

implementation of exit_shell

Parameters

args	

4.12.2.4 export()

```
void export (
            char ** args )
```

implementation of command export

Parameters

args	

4.12.2.5 print_echo()

```
void print_echo (
             char ** args,
             bool e,
             bool n )
```

4.12.2.6 print_without_sp()

```
int print_without_sp (
             char * c )
```

4.12.3 Variable Documentation

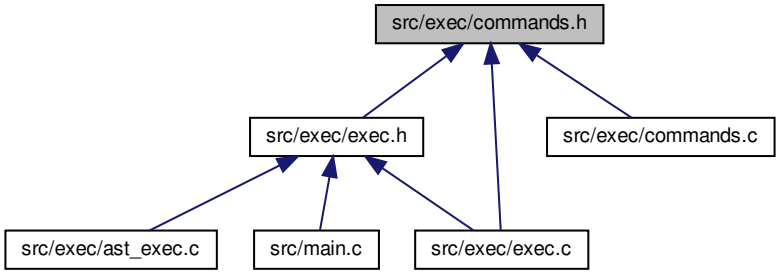
4.12.3.1 environ

```
char** environ
```

4.13 src/exec/commands.h File Reference

Extra commands functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct echo_tab

Functions

- void echo (char **args)
  implementation of command echo
- void cd (char **args)
  implementation of command cd
- void export (char **args)
  implementation of command export
- void exit_shell (char **args)
  implementation of exit_shell

4.13.1 Detailed Description

Extra commands functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.13.2 Function Documentation

4.13.2.1 cd()

```
void cd (
            char ** args )
```

implementation of command cd

Parameters

args	

4.13.2.2 echo()

```
void echo (
            char ** args )
```

implementation of command echo

Parameters

args	

4.13.2.3 exit_shell()

```
void exit_shell (
            char ** args )
```

implementation of exit_shell

Parameters

args	

4.13.2.4 export()

```
void export (
            char ** args )
```

implementation of command export

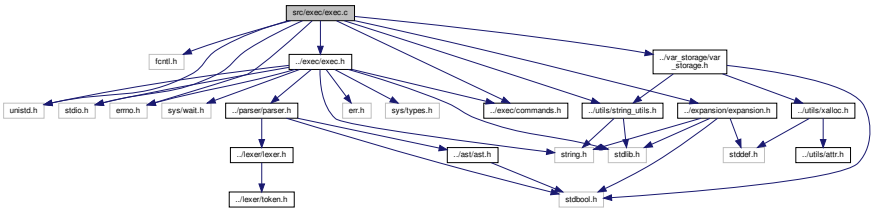
Parameters

args	

4.14 src/exec/exec.c File Reference

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include "../exec/exec.h"
#include "../utils/string_utils.h"
#include "../var_storage/var_storage.h"
#include "../expansion/expansion.h"
#include "../exec/commands.h"
#include <errno.h>
```

Include dependency graph for exec.c:



Macros

- #define _XOPEN_SOURCE 700
- #define READ_END 0
- #define WRITE_END 1
- #define STDOUT_FILENO 1
- #define STDIN_FILENO 0
- #define DEBUG_FLAG true
- #define DEBUG(msg)

Functions

- bool dup_file (char *file, char *flag, int io)
- bool manage_redirections (struct tab_redi tab)
- bool execute (char **args, struct tab_redi tab)
- bool exec_node_input (struct node_input *ast)

  execute input
- bool exec_node_list (struct node_list *ast)

  execute list
- bool exec_node_and_or (struct node_and_or *ast)

  execute and/or
- bool exec_node_pipeline (struct node_pipeline *ast)

  execute pipeline
- bool exec_node_command (struct node_command *ast, bool with_fork)

  execute command
- struct tab_redi init_tab_redi (struct tab_redi tab)
- struct tab_redi append_tab_redi (struct tab_redi tab, struct node_redirection *e)
- bool exec_node_simple_command (struct node_simple_command *ast, bool with_fork)

  execute simple command

- bool exec_node_shell_command (struct node_shell_command *ast)

  execute shell command
- bool exec_node_funcdec (struct node_funcdec *ast)

  execute funcdec
- bool exec_node_redirection (struct node_redirection *ast)

  execute redirection
- bool exec_node_prefix (struct node_prefix *ast)

  execute prefix
- bool exec_node_element (struct node_element *ast)

  execute element
- bool exec_node_compound_list (struct node_compound_list *ast)

  execute compound list
- bool exec_node_while (struct node_while *ast)

  execute while
- bool exec_node_until (struct node_until *ast)

  execute until
- bool exec_node_case (struct node_case *ast)

  execute case
- bool exec_node_if (struct node_if *ast)

  execute if
- bool exec_node_elif (struct node_if *ast)

  execute elif
- bool exec_node_for (struct node_for *ast)

  execute for
- bool exec_node_else_clause (struct node_else_clause *ast)

  execute else clause
- bool exec_node_do_group (struct node_do_group *ast)

  execute do group
- bool exec_node_case_clause (struct node_case_clause *ast)

  execute case clause
- bool exec_node_case_item (struct node_case_item *ast)

  execute case item

Variables

- const struct commands cmd [3]

4.14.1 Macro Definition Documentation

4.14.1.1 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 700
```

4.14.1.2 DEBUG

```
#define DEBUG(
              msg )
```

Value:

```
if (DEBUG_FLAG) \
                    printf("%s\n", msg);
```

4.14.1.3 DEBUG_FLAG

```
#define DEBUG_FLAG true
```

4.14.1.4 READ_END

```
#define READ_END 0
```

4.14.1.5 STDIN_FILENO

```
#define STDIN_FILENO 0
```

4.14.1.6 STDOUT_FILENO

```
#define STDOUT_FILENO 1
```

4.14.1.7 WRITE_END

```
#define WRITE_END 1
```

4.14.2 Function Documentation

4.14.2.1 append_tab_redi()

```
struct tab_redi append_tab_redi (
            struct tab_redi tab,
            struct node_redirection * e )
```

4.14.2.2 dup_file()

```
bool dup_file (
            char * file,
            char * flag,
            int io )
```

4.14.2.3 exec_node_and_or()

```
bool exec_node_and_or (
            struct node_and_or * ast )
```

execute and/or

Parameters

ast	

Returns

true
false

4.14.2.4 exec_node_case()

```
bool exec_node_case (
            struct node_case * ast )
```

execute case

Parameters

ast	

Returns

true
false

4.14.2.5 exec_node_case_clause()

```
bool exec_node_case_clause (
            struct node_case_clause * ast )
```

execute case clause

Parameters

ast	

Returns

true
false

4.14.2.6 exec_node_case_item()

```
bool exec_node_case_item (
            struct node_case_item * ast )
```

execute case item

Parameters

ast	

Returns

true
false

4.14.2.7 exec_node_command()

```
bool exec_node_command (
            struct node_command * ast,
            bool with_fork )
```

execute command

Parameters

ast	
with_fork	

Returns

true
false

4.14.2.8 exec_node_compound_list()

```
bool exec_node_compound_list (
            struct node_compound_list * ast )
```

execute compound list

Parameters

ast	

Returns

true
false

4.14.2.9 exec_node_do_group()

```
bool exec_node_do_group (
            struct node_do_group * ast )
```

execute do group

Parameters

ast	

Returns

true
false

4.14.2.10 exec_node_element()

```
bool exec_node_element (
            struct node_element * ast )
```

execute element

Parameters

ast	

Returns

true
false

4.14.2.11 exec_node_elif()

```
bool exec_node_elif (
            struct node_if * ast )
```

execute elif

Parameters

ast	

Returns

true
false

4.14.2.12 exec_node_else_clause()

```
bool exec_node_else_clause (
            struct node_else_clause * ast )
```

execute else clause

Parameters

ast	

Returns

true
false

4.14.2.13 exec_node_for()

```
bool exec_node_for (
            struct node_for * ast )
```

execute for

Parameters

ast	

Returns

true
false

4.14.2.14 exec_node_funcdec()

```
bool exec_node_funcdec (
            struct node_funcdec * ast )
```

execute funcdec

Parameters

ast	

Returns

true
false

4.14.2.15 exec_node_if()

```
bool exec_node_if (
            struct node_if * ast )
```

execute if

Parameters

ast	

Returns

true
false

4.14.2.16 exec_node_input()

```
bool exec_node_input (
            struct node_input * ast )
```

execute input

Parameters

ast	

Returns

true
false

4.14.2.17 exec_node_list()

```
bool exec_node_list (
            struct node_list * ast )
```

execute list

Parameters

ast	

Returns

true
false

4.14.2.18 exec_node_pipeline()

```
bool exec_node_pipeline (
            struct node_pipeline * ast )
```

execute pipeline

Parameters

ast	

Returns

true
false

4.14.2.19 exec_node_prefix()

```
bool exec_node_prefix (
            struct node_prefix * ast )
```

execute prefix

Parameters

ast	

Returns

true
false

4.14.2.20 exec_node_redirection()

```
bool exec_node_redirection (
            struct node_redirection * ast )
```

execute redirection

Parameters

ast	

Returns

true
false

4.14.2.21 exec_node_shell_command()

```
bool exec_node_shell_command (
            struct node_shell_command * ast )
```

execute shell command

Parameters

ast	

Returns

true
false

4.14.2.22 exec_node_simple_command()

```
bool exec_node_simple_command (
            struct node_simple_command * ast,
            bool with_fork )
```

execute simple command

Parameters

ast	
with_fork	

Returns

true
false

4.14.2.23 exec_node_until()

```
bool exec_node_until (
            struct node_until * ast )
```

execute until

Parameters

ast	

Returns

true
false

4.14.2.24 exec_node_while()

```
bool exec_node_while (
            struct node_while * ast )
```

execute while

Parameters

ast	

Returns

true
false

4.14.2.25 execute()

```
bool execute (
            char ** args,
            struct tab_redi tab )
```

4.14.2.26 init_tab_redi()

```
struct tab_redi init_tab_redi (
            struct tab_redi tab )
```

4.14.2.27 manage_redirections()

```
bool manage_redirections (
            struct tab_redi tab )
```

4.14.3 Variable Documentation

4.14.3.1 cmd

```
const struct commands cmd[3]
```

Initial value:

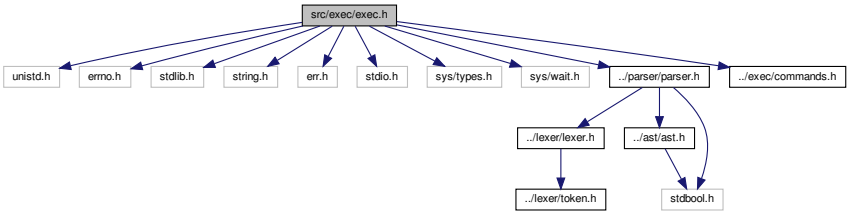
```
= {
    {"cd", &cd},

    {"export", &export},
    {NULL, NULL}}
```

4.15 src/exec/exec.h File Reference

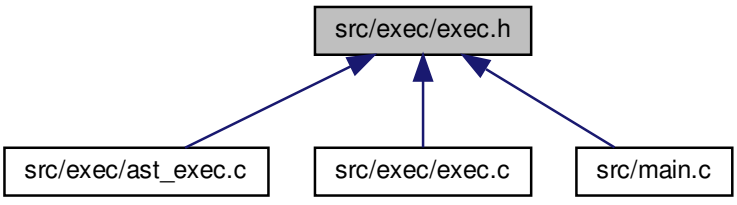
Execution functions.

```
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <err.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "../parser/parser.h"
#include "../exec/commands.h"
```

Include dependency graph for exec.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct commands
- struct std
- struct tab_redi

Macros

- #define NB_MAX_PIPE 10
- #define ERROR(msg)

Functions

- struct tab_redirection * init_tab_redirection (void)
  create and init the table of redirection
- bool exec_node_input (struct node_input *ast)
  execute input
- bool exec_node_list (struct node_list *ast)
  execute list
- bool exec_node_and_or (struct node_and_or *ast)
  execute and/or
- bool exec_node_pipeline (struct node_pipeline *ast)
  execute pipeline
- bool exec_node_command (struct node_command *ast, bool with_fork)
  execute command
- bool exec_node_simple_command (struct node_simple_command *ast, bool with_fork)
  execute simple command
- bool exec_node_shell_command (struct node_shell_command *ast)
  execute shell command
- bool exec_node_funcdec (struct node_funcdec *ast)
  execute funcdec
- bool exec_node_redirection (struct node_redirection *ast)
  execute redirection
- bool exec_node_prefix (struct node_prefix *ast)
  execute prefix
- bool exec_node_element (struct node_element *ast)
  execute element
- bool exec_node_compound_list (struct node_compound_list *ast)
  execute compound list
- bool exec_node_while (struct node_while *ast)
  execute while
- bool exec_node_until (struct node_until *ast)
  execute until
- bool exec_node_case (struct node_case *ast)
  execute case
- bool exec_node_if (struct node_if *ast)
  execute if
- bool exec_node_elif (struct node_if *ast)
  execute elif
- bool exec_node_for (struct node_for *ast)

execute for

- bool exec_node_else_clause (struct node_else_clause *ast)

  execute else clause

- bool exec_node_do_group (struct node_do_group *ast)

  execute do group

- bool exec_node_case_clause (struct node_case_clause *ast)

  execute case clause

- bool exec_node_case_item (struct node_case_item *ast)

  execute case item

4.15.1 Detailed Description

Execution functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.15.2 Macro Definition Documentation

4.15.2.1 ERROR

```
#define ERROR(
              msg )
```

Value:

```
fprintf(stderr, "%s\n", msg); \
            return true; \
```

4.15.2.2 NB_MAX_PIPE

```
#define NB_MAX_PIPE 10
```

4.15.3 Function Documentation

4.15.3.1 exec_node_and_or()

```
bool exec_node_and_or (
            struct node_and_or * ast )
```

execute and/or

Parameters

ast	

Returns

true
false

4.15.3.2 exec_node_case()

```
bool exec_node_case (
            struct node_case * ast )
```

execute case

Parameters

ast	

Returns

true
false

4.15.3.3 exec_node_case_clause()

```
bool exec_node_case_clause (
            struct node_case_clause * ast )
```

execute case clause

Parameters

ast	

Returns

true
false

4.15.3.4 exec_node_case_item()

```
bool exec_node_case_item (
            struct node_case_item * ast )
```

execute case item

Parameters

ast	

Returns

true
false

4.15.3.5 exec_node_command()

```
bool exec_node_command (
            struct node_command * ast,
            bool with_fork )
```

execute command

Parameters

ast	
with_fork	

Returns

true
false

4.15.3.6 exec_node_compound_list()

```
bool exec_node_compound_list (
            struct node_compound_list * ast )
```

execute compound list

Parameters

ast	

Returns

true
false

4.15.3.7 exec_node_do_group()

```
bool exec_node_do_group (
            struct node_do_group * ast )
```

execute do group

Parameters

ast	

Returns

true
false

4.15.3.8 exec_node_element()

```
bool exec_node_element (
            struct node_element * ast )
```

execute element

Parameters

ast	

Returns

true
false

4.15.3.9 exec_node_elif()

```
bool exec_node_elif (
            struct node_if * ast )
```

execute elif

Parameters

ast	

Returns

true
false

4.15.3.10 exec_node_else_clause()

```
bool exec_node_else_clause (
            struct node_else_clause * ast )
```

execute else clause

Parameters

ast	

Returns

true
false

4.15.3.11 exec_node_for()

```
bool exec_node_for (
            struct node_for * ast )
```

execute for

Parameters

ast	

Returns

true
false

4.15.3.12 exec_node_funcdec()

```
bool exec_node_funcdec (
            struct node_funcdec * ast )
```

execute funcdec

Parameters

ast	

Returns

true
false

4.15.3.13 exec_node_if()

```
bool exec_node_if (
            struct node_if * ast )
```

execute if

Parameters

ast	

Returns

true
false

4.15.3.14 exec_node_input()

```
bool exec_node_input (
            struct node_input * ast )
```

execute input

Parameters

ast	

Returns

true
false

4.15.3.15 exec_node_list()

```
bool exec_node_list (
            struct node_list * ast )
```

execute list

Parameters

ast	

Returns

true
false

4.15.3.16 exec_node_pipeline()

```
bool exec_node_pipeline (
            struct node_pipeline * ast )
```

execute pipeline

Parameters

ast	

Returns

true
false

4.15.3.17 exec_node_prefix()

```
bool exec_node_prefix (
            struct node_prefix * ast )
```

execute prefix

Parameters

ast	

Returns

true
false

4.15.3.18 exec_node_redirection()

```
bool exec_node_redirection (
            struct node_redirection * ast )
```

execute redirection

Parameters

ast	

Returns

true
false

4.15.3.19 exec_node_shell_command()

```
bool exec_node_shell_command (
            struct node_shell_command * ast )
```

execute shell command

Parameters

ast	

Returns

true
false

4.15.3.20 exec_node_simple_command()

```
bool exec_node_simple_command (
            struct node_simple_command * ast,
            bool with_fork )
```

execute simple command

Parameters

ast	
with_fork	

Returns

true
false

4.15.3.21 exec_node_until()

```
bool exec_node_until (
            struct node_until * ast )
```

execute until

Parameters

ast	

Returns

true
false

4.15.3.22 exec_node_while()

```
bool exec_node_while (
            struct node_while * ast )
```

execute while

Parameters

ast	

Returns

true
false

4.15.3.23 init_tab_redirection()

```
struct tab_redirection* init_tab_redirection (
            void  )
```

create and init the table of redirection

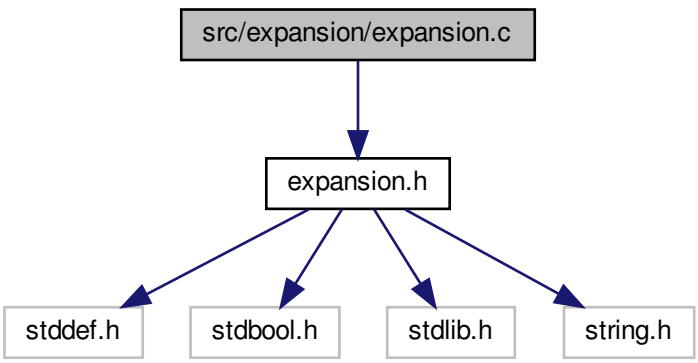
Returns

struct tab_redirection*

4.16 src/expansion/expansion.c File Reference

```
#include "expansion.h"
```

Include dependency graph for expansion.c:



Functions

- char ∗ substitute (char ∗word)

4.16.1 Function Documentation

4.16.1.1 substitute()

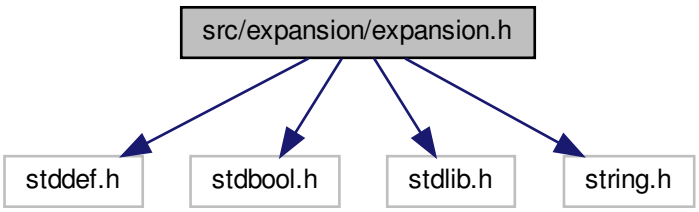
```
char* substitute (
            char * word )
```

4.17 src/expansion/expansion.h File Reference

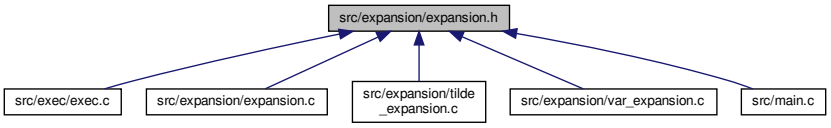
Var storage structures and functions.

```
#include <stddef.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for expansion.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct program_data_storage

Enumerations

- enum param_type {
  PAR_NUMBER, PAR_STAR, PAR_AT, PAR_HASH,
  PAR_QUES, PAR_UNKNOWN }

Functions

- char * substitute (char *word)
- void new_program_data_storage (int argc, char *argv[ ])
- void free_program_data_storage (void)
- void update_last_status (int status)
- char * perform_var_expansion (char *word)
- enum param_type is_special_char (char c)
- char * substitute_number (char c)
- struct buffer * substitute_star (void)
- struct buffer * substitute_at (void)
- char * substitute_hash (void)
- char * substitute_ques (void)
- char * substitute_random (char *word, size_t *i, bool *should_continue, int is_brack)
- char * substitute_uid (char *word, size_t *i, bool *should_continue, int is_brack)
- char * substitute_oldpwd (char *word, size_t *i, bool *should_continue, int is_brack)
- char * substitute_ifs (char *word, size_t *i, bool *should_continue, int is_brack)
- int get_random_int (void)
- size_t get_next_brack_index (const char *c, size_t j)
- size_t get_next_dollar_index (const char *c, size_t j)
- char * perform_tilde_expansion (char *word)
- char * substitute_minus_tilde (char *word, size_t *i)
- char * substitute_plus_tilde (char *word, size_t *i)
- char * substitute_tilde (char *word, size_t *i)

Variables

- struct program_data_storage * program_data

4.17.1 Detailed Description

Var storage structures and functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

4.17.2 Enumeration Type Documentation

4.17.2.1 param_type

```
enum param_type
```

Enumerator

PAR_NUMBER	
PAR_STAR	
PAR_AT	
PAR_HASH	
PAR_QUES	
PAR_UNKNOWN	

4.17.3 Function Documentation

4.17.3.1 free_program_data_storage()

```
void free_program_data_storage (
            void  )
```

4.17.3.2 get_next_brack_index()

```
size_t get_next_brack_index (
            const char * c,
            size_t j )
```

4.17.3.3 get_next_dollar_index()

```
size_t get_next_dollar_index (
            const char * c,
            size_t j )
```

4.17.3.4 get_random_int()

```
int get_random_int (
            void  )
```

4.17.3.5 is_special_char()

```
enum param_type is_special_char (
            char c )
```

4.17.3.6 new_program_data_storage()

```
void new_program_data_storage (
            int argc,
            char * argv[] )
```

4.17.3.7 perform_tilde_expansion()

```
char* perform_tilde_expansion (
            char * word )
```

4.17.3.8 perform_var_expansion()

```
char* perform_var_expansion (
            char * word )
```

4.17.3.9 substitute()

```
char* substitute (
            char * word )
```

4.17.3.10 substitute_at()

```
struct buffer* substitute_at (
            void  )
```

4.17.3.11 substitute_hash()

```
char* substitute_hash (
            void  )
```

4.17.3.12 substitute_ifs()

```
char* substitute_ifs (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.17.3.13 substitute_minus_tilde()

```
char* substitute_minus_tilde (
            char * word,
            size_t * i )
```

4.17.3.14 substitute_number()

```
char* substitute_number (
            char c )
```

4.17.3.15 substitute_oldpwd()

```
char* substitute_oldpwd (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.17.3.16 substitute_plus_tilde()

```
char* substitute_plus_tilde (
            char * word,
            size_t * i )
```

4.17.3.17 substitute_ques()

```
char* substitute_ques (
            void  )
```

4.17.3.18 substitute_random()

```
char* substitute_random (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.17.3.19 substitute_star()

```
struct buffer* substitute_star (
            void  )
```

4.17.3.20 substitute_tilde()

```
char* substitute_tilde (
            char * word,
            size_t * i )
```

4.17.3.21 substitute_uid()

```
char* substitute_uid (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.17.3.22 update_last_status()

```
void update_last_status (
            int status )
```

4.17.4 Variable Documentation

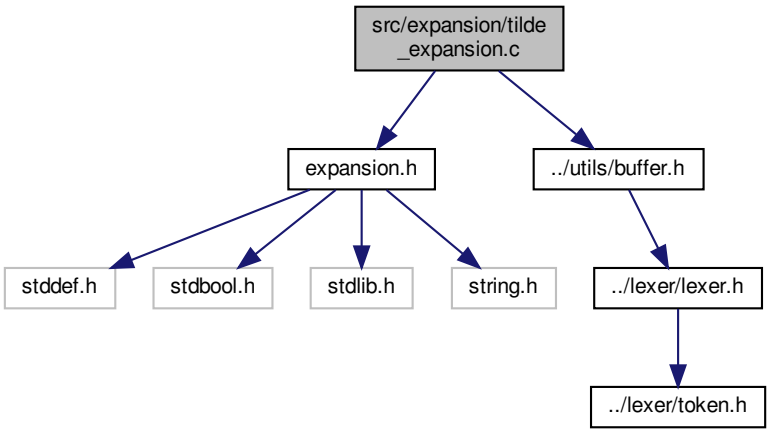
4.17.4.1 program_data

```
struct program_data_storage* program_data
```

4.18 src/expansion/tilde_expansion.c File Reference

```
#include "expansion.h"
#include "../utils/buffer.h"
```

Include dependency graph for tilde_expansion.c:



Functions

- char * perform_tilde_expansion (char *word)
- bool is_valid_tilde (char *word, size_t i)
- char * substitute_minus_tilde (char *word, size_t *i)
- char * substitute_plus_tilde (char *word, size_t *i)
- char * substitute_tilde (char *word, size_t *i)

4.18.1 Function Documentation

4.18.1.1 is_valid_tilde()

```
bool is_valid_tilde (
            char * word,
            size_t i )
```

4.18.1.2 perform_tilde_expansion()

```
char* perform_tilde_expansion (
            char * word )
```

4.18.1.3 substitute_minus_tilde()

```
char* substitute_minus_tilde (
            char * word,
            size_t * i )
```

4.18.1.4 substitute_plus_tilde()

```
char* substitute_plus_tilde (
            char * word,
            size_t * i )
```

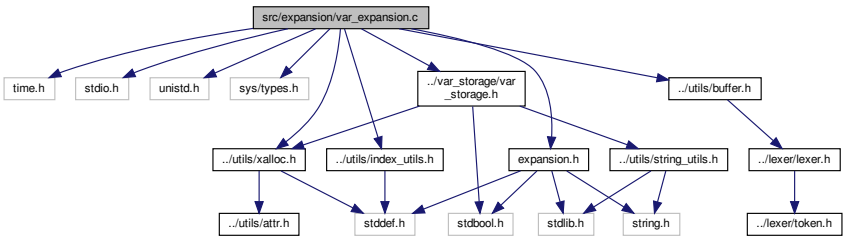
4.18.1.5 substitute_tilde()

```
char* substitute_tilde (
            char * word,
            size_t * i )
```

4.19 src/expansion/var_expansion.c File Reference

```
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include "expansion.h"
#include "../var_storage/var_storage.h"
#include "../utils/buffer.h"
#include "../utils/xalloc.h"
#include "../utils/index_utils.h"
```

Include dependency graph for var_expansion.c:



Functions

- void new_program_data_storage (int argc, char ∗argv[ ])
- void free_program_data_storage (void)
- void update_last_status (int status)
- char ∗ perform_var_expansion (char ∗word)
- char ∗ substitute_number (char c)
- struct buffer ∗ substitute_star (void)
- struct buffer ∗ substitute_at (void)
- char ∗ substitute_hash (void)
- char ∗ substitute_ques (void)
- bool next_param_is_printable (char ∗word, size_t i, size_t param_len, bool is_brack)
- char ∗ substitute_random (char ∗word, size_t ∗i, bool ∗should_continue, int is_brack)
- char ∗ substitute_uid (char ∗word, size_t ∗i, bool ∗should_continue, int is_brack)
- char ∗ substitute_oldpwd (char ∗word, size_t ∗i, bool ∗should_continue, int is_brack)
- char ∗ substitute_ifs (char ∗word, size_t ∗i, bool ∗should_continue, int is_brack)
- enum param_type is_special_char (char c)
- int get_random_int (void)

4.19.1 Function Documentation

4.19.1.1 free_program_data_storage()

```
void free_program_data_storage (
            void  )
```

4.19.1.2 get_random_int()

```
int get_random_int (
            void  )
```

4.19.1.3 is_special_char()

```
enum param_type is_special_char (
            char c )
```

4.19.1.4 new_program_data_storage()

```
void new_program_data_storage (
            int argc,
            char * argv[] )
```

4.19.1.5 next_param_is_printable()

```
bool next_param_is_printable (
            char * word,
            size_t i,
            size_t param_len,
            bool is_brack )
```

4.19.1.6 perform_var_expansion()

```
char* perform_var_expansion (
            char * word )
```

4.19.1.7 substitute_at()

```
struct buffer* substitute_at (
            void  )
```

4.19.1.8 substitute_hash()

```
char* substitute_hash (
            void  )
```

4.19.1.9 substitute_ifs()

```
char* substitute_ifs (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.19.1.10 substitute_number()

```
char* substitute_number (
            char c )
```

4.19.1.11 substitute_oldpwd()

```
char* substitute_oldpwd (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.19.1.12 substitute_ques()

```
char* substitute_ques (
            void  )
```

4.19.1.13 substitute_random()

```
char* substitute_random (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

4.19.1.14 substitute_star()

```
struct buffer* substitute_star (
            void  )
```

4.19.1.15 substitute_uid()

```
char* substitute_uid (
            char * word,
            size_t * i,
            bool * should_continue,
            int is_brack )
```

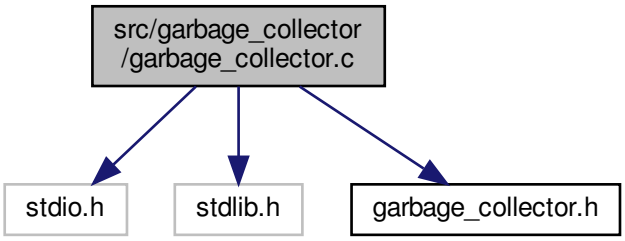
4.19.1.16 update_last_status()

```
void update_last_status (
            int status )
```

4.20 src/garbage_collector/garbage_collector.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "garbage_collector.h"
```

Include dependency graph for garbage_collector.c:



Functions

- void new_garbage_collector (void)

  create the garbage collector
- void append_to_garbage (void *addr)

  append addr to list of elements
- void free_garbage_collector (void)

  free list of elements
- void print_garbage_collector (void)

4.20.1 Function Documentation

4.20.1.1 append_to_garbage()

```
void append_to_garbage (
            void * addr )
```

append addr to list of elements

Parameters

addr	

4.20.1.2 free_garbage_collector()

```
void free_garbage_collector (
            void  )
```

free list of elements

4.20.1.3 new_garbage_collector()

```
void new_garbage_collector (
            void  )
```

create the garbage collector

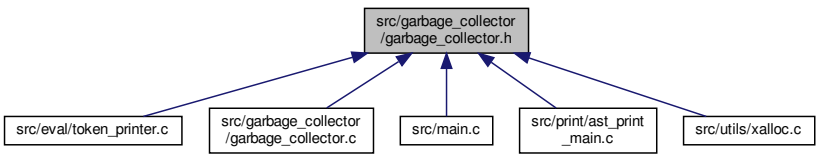
4.20.1.4 print_garbage_collector()

```
void print_garbage_collector (
            void  )
```

4.21 src/garbage_collector/garbage_collector.h File Reference

Execution functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct garbage_element
- struct garbage_collector

Functions

- void new_garbage_collector (void)
  create the garbage collector
- void append_to_garbage (void *addr)
  append addr to list of elements
- void free_garbage_collector ()
  free list of elements

Variables

- struct garbage_collector * garbage_collector

4.21.1 Detailed Description

Execution functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

4.21.2 Function Documentation

4.21.2.1 append_to_garbage()

```
void append_to_garbage (
            void * addr )
```

append addr to list of elements

Parameters

addr	

4.21.2.2 free_garbage_collector()

```
void free_garbage_collector ( )
```

free list of elements

4.21.2.3 new_garbage_collector()

```
void new_garbage_collector (
            void  )
```

create the garbage collector

4.21.3 Variable Documentation

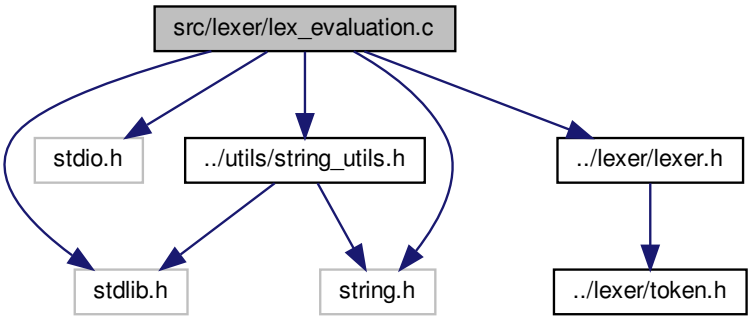
4.21.3.1 garbage_collector

```
struct garbage_collector* garbage_collector
```

4.22 src/lexer/lex_evaluation.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "../utils/string_utils.h"
#include "../lexer/lexer.h"
```

Include dependency graph for lex_evaluation.c:



Functions

- char * lex_backslash (char *c, size_t i)
- struct token * lex_great_less_and (const char *c, size_t i)

  process great less and into token
- struct token * lex_io_number (char *c, size_t *i)

  process io number into token
- struct token * lex_great_less (char *c, size_t i)

  process great less into token
- struct token * lex_comments (char *c, size_t i)

  process comments into token
- struct token * lex_uni_character (char *c, size_t i)

  process uni character into token
- struct token * lex_assignment_word (char *c, size_t *i)

  process assignment word into token
- struct token * lex_assignment_value (char *c, size_t *i)

  process assignment word into token
- enum token_type evaluate_keyword (char *c)

  Return the associated keyword of a string token.
- enum token_type evaluate_token (char *c)

  Return the associated type of a string token.

4.22.1 Function Documentation

4.22.1.1 evaluate_keyword()

```
enum token_type evaluate_keyword (
            char * c )
```

Return the associated keyword of a string token.

Parameters

c	the string to be compared to all the keywords.

4.22.1.2 evaluate_token()

```
enum token_type evaluate_token (
            char * c )
```

Return the associated type of a string token.

Parameters

c	the string to be compared to all the tokens.

4.22.1.3 lex_assignment_value()

```
struct token* lex_assignment_value (
            char * c,
            size_t * i )
```

process assignment word into token

Parameters

c	
i	

Returns

struct token*

4.22.1.4 lex_assignment_word()

```
struct token* lex_assignment_word (
            char * c,
            size_t * i )
```

process assignment word into token

Parameters

c	
i	

Returns

struct token*

4.22.1.5 lex_backslash()

```
char* lex_backslash (
            char * c,
            size_t i )
```

4.22.1.6 lex_comments()

```
struct token* lex_comments (
            char * c,
            size_t i )
```

process comments into token

Parameters

c	
i	

Returns

struct token*

c[i - 1]

4.22.1.7 lex_great_less()

```
struct token* lex_great_less (
            char * c,
            size_t i )
```

process great less into token

Parameters

c	
i	

Returns

struct token*

4.22.1.8 lex_great_less_and()

```
struct token* lex_great_less_and (
            const char * c,
            size_t i )
```

process great less and into token

Parameters

c	
i	

Returns

struct token*

4.22.1.9 lex_io_number()

```
struct token* lex_io_number (
            char * c,
            size_t * i )
```

process io number into token

Parameters

c	
i	

Returns

struct token*

4.22.1.10 lex_uni_character()

```
struct token* lex_uni_character (
            char * c,
            size_t i )
```

process uni character into token

Parameters

c	
i	

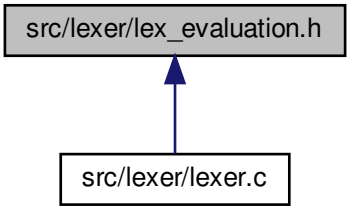
Returns

struct token*

4.23 src/lexer/lex_evaluation.h File Reference

Unit lexing functions.

This graph shows which files directly or indirectly include this file:



Functions

- struct token ∗ lex_great_less_and (const char ∗c, size_t i)
  process great less and into token
- struct token ∗ lex_io_number (char ∗c, size_t ∗i)
  process io number into token
- char ∗ lex_backslash (const char ∗c, size_t i)
  process backslash in the lexer
- struct token ∗ lex_great_less (char ∗c, size_t i)
  process great less into token
- struct token ∗ lex_comments (char ∗c, size_t i)
  process comments into token
- struct token ∗ lex_uni_character (char ∗c, size_t i)
  process uni character into token
- struct token ∗ lex_assignment_word (char ∗c, size_t ∗i)
  process assignment word into token
- struct token ∗ lex_assignment_value (char ∗c, size_t ∗i)
  process assignment word into token
- enum token_type evaluate_keyword (char ∗c)
  Return the associated keyword of a string token.
- enum token_type evaluate_token (char ∗c)
  Return the associated type of a string token.

4.23.1 Detailed Description

Unit lexing functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.23.2 Function Documentation

4.23.2.1 evaluate_keyword()

```
enum token_type evaluate_keyword (
            char * c )
```

Return the associated keyword of a string token.

Parameters

c	the string to be compared to all the keywords.

4.23.2.2 evaluate_token()

```
enum token_type evaluate_token (
            char * c )
```

Return the associated type of a string token.

Parameters

c	the string to be compared to all the tokens.

4.23.2.3 lex_assignment_value()

```
struct token* lex_assignment_value (
            char * c,
            size_t * i )
```

process assignment word into token

Parameters

c	
i	

Returns

struct token*

4.23.2.4 lex_assignment_word()

```
struct token* lex_assignment_word (
            char * c,
            size_t * i )
```

process assignment word into token

Parameters

c	
i	

Returns

struct token*

4.23.2.5 lex_backslash()

```
char* lex_backslash (
            const char * c,
            size_t i )
```

process backslash in the lexer

Parameters

c	
i	

Returns

char*

4.23.2.6 lex_comments()

```
struct token* lex_comments (
            char * c,
            size_t i )
```

process comments into token

Parameters

c	
i	

Returns

struct token*

c[i - 1]

4.23.2.7 lex_great_less()

```
struct token* lex_great_less (
            char * c,
            size_t i )
```

process great less into token

Parameters

c	
i	

Returns

struct token*

4.23.2.8 lex_great_less_and()

```
struct token* lex_great_less_and (
            const char * c,
            size_t i )
```

process great less and into token

Parameters

c	
i	

Returns

struct token*

4.23.2.9 lex_io_number()

```
struct token* lex_io_number (
            char * c,
            size_t * i )
```

process io number into token

Parameters

c	
i	

Returns

struct token*

4.23.2.10 lex_uni_character()

```
struct token* lex_uni_character (
            char * c,
            size_t i )
```

process uni character into token

Parameters

c	
i	

Returns

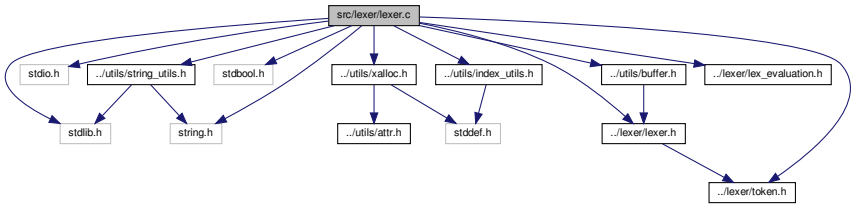
struct token*

4.24 src/lexer/lexer.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#include "../utils/xalloc.h"
#include "../utils/string_utils.h"
#include "../utils/buffer.h"
#include "../lexer/token.h"
#include "../lexer/lexer.h"
#include "../lexer/lex_evaluation.h"
#include "../utils/index_utils.h"
```

Include dependency graph for lexer.c:



Macros

- #define _POSIX_C_SOURCE 200112L

Functions

- char ** split (char *str)
- int lex_full (struct lexer *lexer, char *c, size_t j)
- int lex_part (struct lexer *lexer, struct buffer *buffer, char *c, size_t *j)
- int lex_parenthesis (struct lexer *lexer, struct buffer *buffer, char *c, size_t *j)
- int lex_separator (struct lexer *lexer, struct buffer *buffer, char *c, size_t *j)
- int lex_parameter (struct lexer *lexer, struct buffer *buffer, char *c, size_t *j)
- void init_lexer (struct lexer *lexer)

  Fill the token list by creating all the tokens from the given string.
- struct lexer * new_lexer (char *str)

  Allocate and init a new lexer.
- void free_lexer (struct lexer *lexer)

  Free all ressources allocated in the lexer.
- struct token * peek (struct lexer *lexer)

  Return the next token without consume it.
- struct token * pop (struct lexer *lexer)

  Return and consume the next token from the input stream.
- void append (struct lexer *lexer, struct token *token)

  Append a new token to the token_list of the lexer.

Variables

- bool is_word = false

4.24.1 Macro Definition Documentation

4.24.1.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 200112L
```

4.24.2 Function Documentation

4.24.2.1 append()

```
void append (
            struct lexer * lexer,
            struct token * token )
```

Append a new token to the token_list of the lexer.

Parameters

lexer	the lexer.
token	the token to append.

4.24.2.2 free_lexer()

```
void free_lexer (
            struct lexer * lexer )
```

Free all ressources allocated in the lexer.

Parameters

lexer	the lexer to free.

4.24.2.3 init_lexer()

```
void init_lexer (
            struct lexer * lexer )
```

Fill the token list by creating all the tokens from the given string.

Parameters

lexer	the lexer.

4.24.2.4 lex_full()

```
int lex_full (
              struct lexer * lexer,
              char * c,
              size_t j )
```

4.24.2.5 lex_parameter()

```
int lex_parameter (
              struct lexer * lexer,
              struct buffer * buffer,
              char * c,
              size_t * j )
```

4.24.2.6 lex_parenthesis()

```
int lex_parenthesis (
              struct lexer * lexer,
              struct buffer * buffer,
              char * c,
              size_t * j )
```

4.24.2.7 lex_part()

```
int lex_part (
              struct lexer * lexer,
              struct buffer * buffer,
              char * c,
              size_t * j )
```

4.24.2.8 lex_separator()

```
int lex_separator (
              struct lexer * lexer,
              struct buffer * buffer,
              char * c,
              size_t * j )
```

4.24.2.9 new_lexer()

```
struct lexer* new_lexer (
              char * str )
```

Allocate and init a new lexer.

Parameters

str	the string to use as input stream.

4.24.2.10 peek()

```
struct token* peek (
            struct lexer * lexer )
```

Return the next token without consume it.

Returns

the next token from the input stream

Parameters

lexer	the lexer to lex from

4.24.2.11 pop()

```
struct token* pop (
            struct lexer * lexer )
```

Return and consume the next token from the input stream.

Returns

the next token from the input stream

Parameters

lexer	the lexer to lex from

4.24.2.12 split()

```
char** split (
            char * str )
```

4.24.3 Variable Documentation

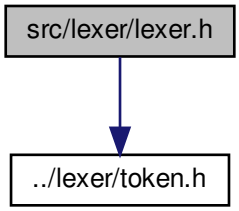
4.24.3.1 is_word

```
bool is_word = false
```

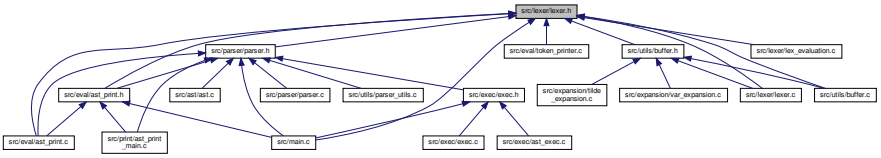
4.25 src/lexer/lexer.h File Reference

Main lexing functions.

```
#include "../lexer/token.h"
```
Include dependency graph for lexer.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct lexer

  Lexer architecture and methods.

Functions

- struct lexer * new_lexer (char *str)

  Allocate and init a new lexer.
- void free_lexer (struct lexer *lexer)

  Free all ressources allocated in the lexer.
- struct token * peek (struct lexer *lexer)

  Return the next token without consume it.
- struct token * pop (struct lexer *lexer)

  Return and consume the next token from the input stream.
- void append (struct lexer *lexer, struct token *token)

  Append a new token to the token_list of the lexer.
- void init_lexer (struct lexer *lexer)

  Fill the token list by creating all the tokens from the given string.
- int is_separator (char c)

4.25.1 Detailed Description

Main lexing functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.25.2 Function Documentation

4.25.2.1 append()

```
void append (
            struct lexer * lexer,
            struct token * token )
```

Append a new token to the token_list of the lexer.

Parameters

lexer	the lexer.
token	the token to append.

4.25.2.2 free_lexer()

```
void free_lexer (
            struct lexer * lexer )
```

Free all ressources allocated in the lexer.

Parameters

lexer	the lexer to free.

4.25.2.3 init_lexer()

```
void init_lexer (
            struct lexer * lexer )
```

Fill the token list by creating all the tokens from the given string.

Parameters

lexer	the lexer.

4.25.2.4 is_separator()

```
int is_separator (
            char c )
```

4.25.2.5 new_lexer()

```
struct lexer* new_lexer (
            char * str )
```

Allocate and init a new lexer.

Parameters

str	the string to use as input stream.

4.25.2.6 peek()

```
struct token* peek (
            struct lexer * lexer )
```

Return the next token without consume it.

Returns

the next token from the input stream

Parameters

lexer	the lexer to lex from

4.25.2.7 pop()

```
struct token* pop (
            struct lexer * lexer )
```

Return and consume the next token from the input stream.

Returns

the next token from the input stream

Parameters

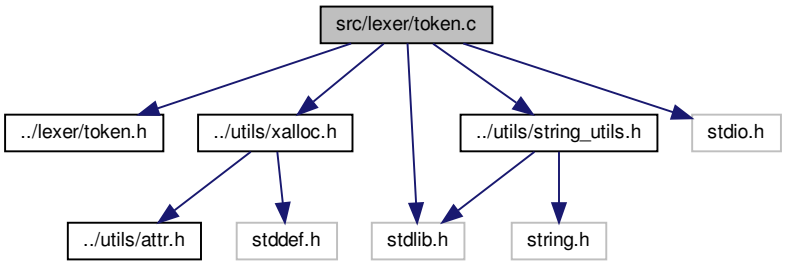
lexer	the lexer to lex from

4.26 src/lexer/token.c File Reference

```
#include "../lexer/token.h"
#include "../utils/xalloc.h"
#include "../utils/string_utils.h"
#include <stdio.h>
```

```
#include <stdlib.h>
```

Include dependency graph for token.c:



Functions

- struct token * new_token (void)

  Token allocator and initializer.
- struct token * new_token_type (int type)
- struct token * new_token_io_number (char number)
- struct token * new_token_word (char *value)
- struct token * new_token_error (char *err)
- void free_token (struct token *token)

  Wrapper to release memory of a token.
- int is_type (struct token *token, unsigned int type)

4.26.1 Function Documentation

4.26.1.1 free_token()

```
void free_token (
            struct token * token )
```

Wrapper to release memory of a token.

Parameters

token	the token to free

4.26.1.2 is_type()

```
int is_type (
```

```
            struct token * token,
            unsigned int type )
```

4.26.1.3 new_token()

```
struct token* new_token (
            void  )
```

Token allocator and initializer.

Returns

a pointer to the allocated token.

4.26.1.4 new_token_error()

```
struct token* new_token_error (
            char * err )
```

4.26.1.5 new_token_io_number()

```
struct token* new_token_io_number (
            char number )
```

4.26.1.6 new_token_type()

```
struct token* new_token_type (
            int type )
```

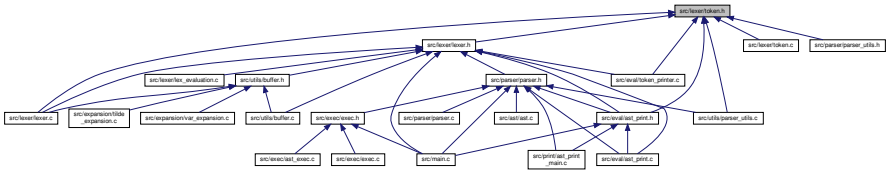
4.26.1.7 new_token_word()

```
struct token* new_token_word (
            char * value )
```

4.27 src/lexer/token.h File Reference

Token structures and functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct token

  Token struct declaration.
- struct token_list

  Basically a lined-list of tokens.

Macros

- #define MAX_TOKEN 256

Enumerations

- enum token_type {
  TOK_ERROR, TOK_NEWLINE, TOK_EOF, TOK_AND,
  TOK_SEPAND, TOK_OR, TOK_PIPE, TOK_SEMI,
  TOK_LPAREN, TOK_RPAREN, TOK_LCURL, TOK_RCURL,
  TOK_DLESSDASH, TOK_DLESS, TOK_LESSGREAT, TOK_LESSAND,
  TOK_LESS, TOK_DGREAT, TOK_GREATAND, TOK_CLOBBER,
  TOK_ASS_WORD, TOK_GREAT, TOK_IONUMBER, TOK_NOT,
  TOK_COMM, TOK_WORD, KW_IF, KW_THEN,
  KW_ELSE, KW_ELIF, KW_FI, KW_DO,
  KW_DONE, KW_FOR, KW_WHILE, KW_UNTIL,
  KW_CASE, KW_ESAC, KW_IN, KW_DSEMI,
  KW_UNKNOWN }

  Type of a token (operators, value, ...)

Functions

- struct token ∗ new_token (void)

  Token allocator and initializer.
- struct token ∗ new_token_type (int type)
- struct token ∗ new_token_io_number (char number)
- struct token ∗ new_token_word (char ∗value)
- struct token ∗ new_token_error (char ∗err)
- int is_type (struct token ∗token, unsigned int type)
- void free_token (struct token ∗token)

  Wrapper to release memory of a token.

4.27.1 Detailed Description

Token structures and functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.27.2 Macro Definition Documentation

4.27.2.1 MAX_TOKEN

```
#define MAX_TOKEN 256
```

4.27.3 Enumeration Type Documentation

4.27.3.1 token_type

```
enum token_type
```

Type of a token (operators, value, ...)

Enumerator

TOK_ERROR	
TOK_NEWLINE	
TOK_EOF	
TOK_AND	
TOK_SEPAND	
TOK_OR	
TOK_PIPE	
TOK_SEMI	

Enumerator

TOK_LPAREN	
TOK_RPAREN	
TOK_LCURL	
TOK_RCURL	
TOK_DLESSDASH	
TOK_DLESS	
TOK_LESSGREAT	
TOK_LESSAND	
TOK_LESS	
TOK_DGREAT	
TOK_GREATAND	
TOK_CLOBBER	
TOK_ASS_WORD	
TOK_GREAT	
TOK_IONUMBER	
TOK_NOT	
TOK_COMM	
TOK_WORD	
KW_IF	
KW_THEN	
KW_ELSE	
KW_ELIF	
KW_FI	
KW_DO	
KW_DONE	
KW_FOR	
KW_WHILE	
KW_UNTIL	
KW_CASE	
KW_ESAC	
KW_IN	
KW_DSEMI	
KW_UNKNOWN	

4.27.4 Function Documentation

4.27.4.1 free_token()

```
void free_token (
            struct token * token )
```

Wrapper to release memory of a token.

Parameters

token	the token to free

4.27.4.2 is_type()

```
int is_type (
            struct token * token,
            unsigned int type )
```

4.27.4.3 new_token()

```
struct token* new_token (
            void  )
```

Token allocator and initializer.

Returns

a pointer to the allocated token.

4.27.4.4 new_token_error()

```
struct token* new_token_error (
            char * err )
```

4.27.4.5 new_token_io_number()

```
struct token* new_token_io_number (
            char number )
```

4.27.4.6 new_token_type()

```
struct token* new_token_type (
            int type )
```

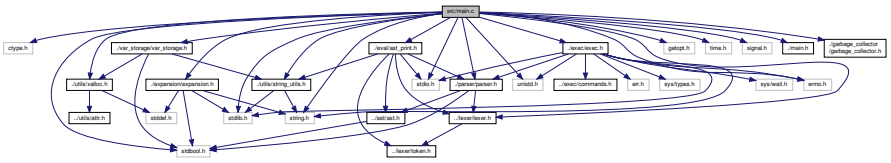
4.27.4.7 new_token_word()

```
struct token* new_token_word (
            char * value )
```

4.28 src/main.c File Reference

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <errno.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include <signal.h>
#include "./main.h"
#include "./parser/parser.h"
#include "./lexer/lexer.h"
#include "./utils/xalloc.h"
#include "./exec/exec.h"
#include "./utils/string_utils.h"
#include "./eval/ast_print.h"
#include "./var_storage/var_storage.h"
#include "./expansion/expansion.h"
#include "./garbage_collector/garbage_collector.h"
```

Include dependency graph for main.c:



Macros

- #define _POSIX_C_SOURCE 200809L

Functions

- void print_usage ()
- void print_prompt ()
- void init_42sh_process (struct option_sh *option)
- int main (int ac, char **av)

4.28.1 Macro Definition Documentation

4.28.1.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 200809L
```

4.28.2 Function Documentation

4.28.2.1 init_42sh_process()

```
void init_42sh_process (
            struct option_sh * option )
```

4.28.2.2 main()

```
int main (
            int ac,
            char ** av )
```

4.28.2.3 print_prompt()

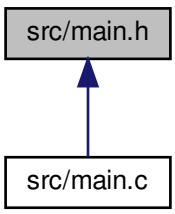
```
void print_prompt ( )
```

4.28.2.4 print_usage()

```
void print_usage ( )
```

4.29 src/main.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct option_sh

Macros

- #define USAGE "Usage : ./42sh [GNU long option] [option] [file]\n"
- #define START_COLOR "\033"
- #define CYAN "36m"
- #define BLINK "\033[5m"
- #define END_COLOR "\033[0m"

4.29.1 Macro Definition Documentation

4.29.1.1 BLINK

```
#define BLINK "\033[5m"
```

4.29.1.2 CYAN

```
#define CYAN "36m"
```

4.29.1.3 END_COLOR

```
#define END_COLOR "\033[0m"
```

4.29.1.4 START_COLOR

```
#define START_COLOR "\033"
```

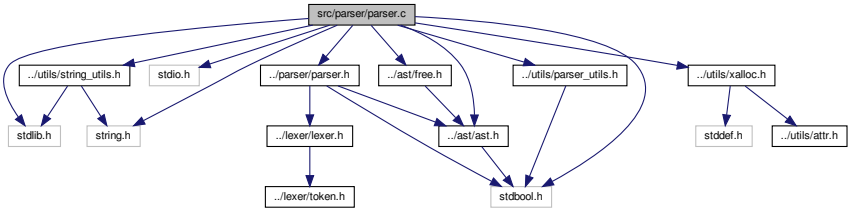
4.29.1.5 USAGE

```
#define USAGE "Usage :  ./42sh [GNU long option] [option] [file]\n"
```

4.30 src/parser/parser.c File Reference

```
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "../parser/parser.h"
#include "../ast/free.h"
#include "../ast/ast.h"
#include "../utils/parser_utils.h"
#include "../utils/xalloc.h"
#include "../utils/string_utils.h"
```

Include dependency graph for parser.c:



Macros

- #define DEBUG_FLAG true
- #define DEBUG(msg)

Functions

- struct parser * init_parser (struct lexer *lexer)

  initialize a parser
- void free_parser (struct parser *p)

  free the parser
- struct token * get_next_token (struct parser *p)
- void parser_comment (struct parser *p)
- void parser_eat (struct parser *p)
- void next_token (struct parser *parser)
- void * parse (struct lexer *lexer)

  parse all of the token given by lexer
- bool parse_input (struct parser *parser, struct node_input **ast)

  parse rule input
- bool parse_list (struct parser *parser, struct node_list **ast)

  parse rule list
- bool parse_and_or (struct parser *parser, struct node_and_or **ast)

  parse rule and or
- bool parse_pipeline (struct parser *parser, struct node_pipeline **ast)

  parse rule pipeline
- bool parse_command (struct parser *p, struct node_command **ast)

  parse rule command

- void parse_multiple_element (struct parser *parser, struct node_simple_command *ast)
- void parse_multiple_prefix (struct parser *parser, struct node_simple_command *ast)
- bool parse_simple_command (struct parser *parser, struct node_simple_command **ast)

  parse rule simple command
- bool parse_shell_command (struct parser *parser, struct node_shell_command **ast)

  parse rule shell command
- bool parse_funcdec (struct parser *parser, struct node_funcdec **ast)

  parse rule funcdec
- bool parse_redirection (struct parser *parser, struct node_redirection **ast)

  parse rule redirection
- bool parse_prefix (struct parser *parser, struct node_prefix **ast)

  parse rule prefix
- bool parse_element (struct parser *parser, struct node_element **ast)

  parse rule element
- bool parse_compound_list (struct parser *parser, struct node_compound_list **ast)

  parse rule compound list
- bool parse_rule_for (struct parser *parser, struct node_for **ast)

  parse rule for
- bool parse_rule_while (struct parser *parser, struct node_while **ast)

  parse rule while
- bool parse_rule_until (struct parser *parser, struct node_until **ast)

  parse rule until
- bool parse_rule_case (struct parser *parser, struct node_case **ast)

  parse rule case
- bool parse_rule_if (struct parser *parser, struct node_if **ast)

  parse rule if
- bool parse_rule_elif (struct parser *parser, struct node_if **ast)
- bool parse_else_clause (struct parser *parser, struct node_else_clause **ast)

  parse else clause
- bool parse_do_group (struct parser *parser, struct node_do_group **ast)

  parse rule do group
- bool parse_case_clause (struct parser *parser, struct node_case_clause **ast)

  parse rule case clause
- bool parse_case_item (struct parser *parser, struct node_case_item **ast)

  parse rule case item

4.30.1 Macro Definition Documentation

4.30.1.1 DEBUG

```
#define DEBUG(
              msg )
```

Value:

```
if (DEBUG_FLAG) \
                        printf("%s", msg);
```

4.30.1.2 DEBUG_FLAG

```
#define DEBUG_FLAG true
```

4.30.2 Function Documentation

4.30.2.1 free_parser()

```
void free_parser (
            struct parser * p )
```

free the parser

Parameters

p	

4.30.2.2 get_next_token()

```
struct token* get_next_token (
            struct parser * p )
```

4.30.2.3 init_parser()

```
struct parser* init_parser (
            struct lexer * lexer )
```

initialize a parser

Parameters

lexer	

Returns

struct parser*

4.30.2.4 next_token()

```
void next_token (
            struct parser * parser )
```

4.30.2.5 parse()

```
void* parse (
            struct lexer * lexer )
```

parse all of the token given by lexer

Parameters

lexer	

Returns

void*

4.30.2.6 parse_and_or()

```
bool parse_and_or (
            struct parser * parser,
            struct node_and_or ** ast )
```

parse rule and or

Parameters

parser	
ast	

Returns

true
false

4.30.2.7 parse_case_clause()

```
bool parse_case_clause (
            struct parser * parser,
            struct node_case_clause ** ast )
```

parse rule case clause

Parameters

parser	
ast	

Returns

true
false

4.30.2.8 parse_case_item()

```
bool parse_case_item (
            struct parser * parser,
            struct node_case_item ** ast )
```

parse rule case item

Parameters

parser	
ast	

Returns

true
false

4.30.2.9 parse_command()

```
bool parse_command (
            struct parser * parser,
            struct node_command ** ast )
```

parse rule command

Parameters

parser	
ast	

Returns

true
false

4.30.2.10 parse_compound_list()

```
bool parse_compound_list (
            struct parser * parser,
            struct node_compound_list ** ast )
```

parse rule compound list

Parameters

parser	
ast	

Returns

true
false

4.30.2.11 parse_do_group()

```
bool parse_do_group (
            struct parser * parser,
            struct node_do_group ** ast )
```

parse rule do group

Parameters

parser	
ast	

Returns

true
false

4.30.2.12 parse_element()

```
bool parse_element (
            struct parser * parser,
            struct node_element ** ast )
```

parse rule element

Parameters

parser	
ast	

Returns

true
false

4.30.2.13 parse_else_clause()

```
bool parse_else_clause (
            struct parser * parser,
            struct node_else_clause ** ast )
```

parse else clause

Parameters

parser	
ast	

Returns

true
false

4.30.2.14 parse_funcdec()

```
bool parse_funcdec (
            struct parser * parser,
            struct node_funcdec ** ast )
```

parse rule funcdec

Parameters

parser	
ast	

Returns

true
false

4.30.2.15 parse_input()

```
bool parse_input (
            struct parser * parser,
            struct node_input ** ast )
```

parse rule input

Parameters

parser	
ast	

Returns

true
false

4.30.2.16 parse_list()

```
bool parse_list (
            struct parser * parser,
            struct node_list ** ast )
```

parse rule list

Parameters

parser	
ast	

Returns

true
false

4.30.2.17 parse_multiple_element()

```
void parse_multiple_element (
            struct parser * parser,
            struct node_simple_command * ast )
```

4.30.2.18 parse_multiple_prefix()

```
void parse_multiple_prefix (
            struct parser * parser,
            struct node_simple_command * ast )
```

4.30.2.19 parse_pipeline()

```
bool parse_pipeline (
            struct parser * parser,
            struct node_pipeline ** ast )
```

parse rule pipeline

Parameters

parser	
ast	

Returns

true
false

4.30.2.20 parse_prefix()

```
bool parse_prefix (
            struct parser * parser,
            struct node_prefix ** ast )
```

parse rule prefix

Parameters

parser	
ast	

Returns

true
false

4.30.2.21 parse_redirection()

```
bool parse_redirection (
            struct parser * parser,
            struct node_redirection ** ast )
```

parse rule redirection

Parameters

parser	
ast	

Returns

true
false

4.30.2.22 parse_rule_case()

```
bool parse_rule_case (
            struct parser * parser,
            struct node_case ** ast )
```

parse rule case

Parameters

parser	
ast	

Returns

true
false

4.30.2.23 parse_rule_elif()

```
bool parse_rule_elif (
            struct parser * parser,
            struct node_if ** ast )
```

4.30.2.24 parse_rule_for()

```
bool parse_rule_for (
            struct parser * parser,
            struct node_for ** ast )
```

parse rule for

Parameters

parser	
ast	

Returns

true
false

4.30.2.25 parse_rule_if()

```
bool parse_rule_if (
            struct parser * parser,
            struct node_if ** ast )
```

parse rule if

Parameters

parser	
ast	

Returns

true
false

4.30.2.26 parse_rule_until()

```
bool parse_rule_until (
            struct parser * parser,
            struct node_until ** ast )
```

parse rule until

Parameters

parser	
ast	

Returns

true
false

4.30.2.27 parse_rule_while()

```
bool parse_rule_while (
            struct parser * parser,
            struct node_while ** ast )
```

parse rule while

Parameters

parser	
ast	

Returns

true
false

4.30.2.28 parse_shell_command()

```
bool parse_shell_command (
            struct parser * parser,
            struct node_shell_command ** ast )
```

parse rule shell command

Parameters

parser	
ast	

Returns

true
false

4.30.2.29 parse_simple_command()

```
bool parse_simple_command (
            struct parser * parser,
            struct node_simple_command ** ast )
```

parse rule simple command

Parameters

parser	
ast	

Returns

true
false

4.30.2.30 parser_comment()

```
void parser_comment (
            struct parser * p )
```

4.30.2.31 parser_eat()

```
void parser_eat (
            struct parser * p )
```

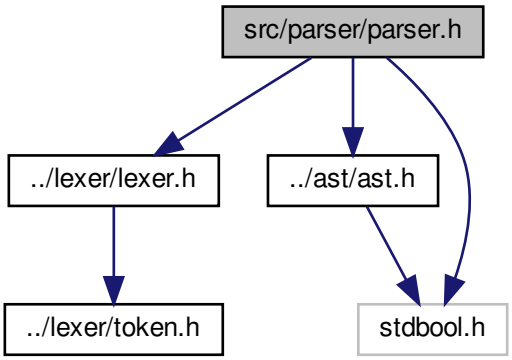
4.31 src/parser/parser.h File Reference

Parsing functions.

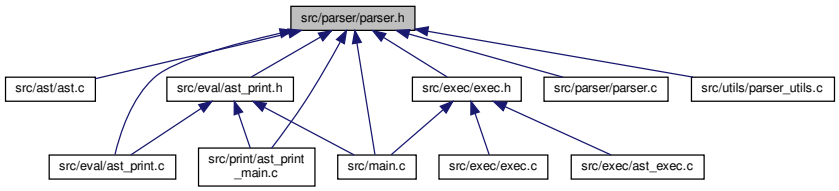
```
#include "../lexer/lexer.h"
#include "../ast/ast.h"
```

```
#include <stdbool.h>
```

Include dependency graph for parser.h:



This graph shows which files directly or indirectly include this file:



Functions

- struct parser ∗ init_parser (struct lexer ∗lexer)

  initialize a parser
- bool parse_look_ahead (struct parser ∗parser, struct token ∗expected_token)

  look the next token without moving the list of tokens
- void ∗ parse (struct lexer ∗lexer)

  parse all of the token given by lexer
- bool parse_input (struct parser ∗parser, struct node_input ∗∗ast)

  parse rule input
- bool parse_list (struct parser ∗parser, struct node_list ∗∗ast)

  parse rule list
- bool parse_and_or (struct parser ∗parser, struct node_and_or ∗∗ast)

  parse rule and or
- bool parse_pipeline (struct parser ∗parser, struct node_pipeline ∗∗ast)

  parse rule pipeline
- bool parse_command (struct parser ∗parser, struct node_command ∗∗ast)

  parse rule command

- bool parse_simple_command (struct parser *parser, struct node_simple_command **ast)
  parse rule simple command
- bool parse_shell_command (struct parser *parser, struct node_shell_command **ast)
  parse rule shell command
- bool parse_funcdec (struct parser *parser, struct node_funcdec **ast)
  parse rule funcdec
- bool parse_redirection (struct parser *parser, struct node_redirection **ast)
  parse rule redirection
- bool parse_element (struct parser *parser, struct node_element **ast)
  parse rule element
- bool parse_prefix (struct parser *parser, struct node_prefix **ast)
  parse rule prefix
- bool parse_compound_list (struct parser *parser, struct node_compound_list **ast)
  parse rule compound list
- bool parse_rule_for (struct parser *parser, struct node_for **ast)
  parse rule for
- bool parse_rule_while (struct parser *parser, struct node_while **ast)
  parse rule while
- bool parse_rule_until (struct parser *parser, struct node_until **ast)
  parse rule until
- bool parse_rule_case (struct parser *parser, struct node_case **ast)
  parse rule case
- bool parse_rule_if (struct parser *parser, struct node_if **ast)
  parse rule if
- bool parse_else_clause (struct parser *parser, struct node_else_clause **ast)
  parse else clause
- bool parse_do_group (struct parser *parser, struct node_do_group **ast)
  parse rule do group
- bool parse_case_clause (struct parser *parser, struct node_case_clause **ast)
  parse rule case clause
- bool parse_case_item (struct parser *parser, struct node_case_item **ast)
  parse rule case item
- void free_parser (struct parser *p)
  free the parser

4.31.1 Detailed Description

Parsing functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.31.2 Function Documentation

4.31.2.1 free_parser()

```
void free_parser (
            struct parser * p )
```

free the parser

Parameters

p	

4.31.2.2 init_parser()

```
struct parser* init_parser (
            struct lexer * lexer )
```

initialize a parser

Parameters

lexer	

Returns

struct parser*

4.31.2.3 parse()

```
void* parse (
            struct lexer * lexer )
```

parse all of the token given by lexer

Parameters

lexer	

Returns

void*

4.31.2.4 parse_and_or()

```
bool parse_and_or (
            struct parser * parser,
            struct node_and_or ** ast )
```

parse rule and or

Parameters

parser	
ast	

Returns

true
false

4.31.2.5 parse_case_clause()

```
bool parse_case_clause (
            struct parser * parser,
            struct node_case_clause ** ast )
```

parse rule case clause

Parameters

parser	
ast	

Returns

true
false

4.31.2.6 parse_case_item()

```
bool parse_case_item (
            struct parser * parser,
            struct node_case_item ** ast )
```

parse rule case item

Parameters

parser	
ast	

Returns

true
false

4.31.2.7 parse_command()

```
bool parse_command (
            struct parser * parser,
            struct node_command ** ast )
```

parse rule command

Parameters

parser	
ast	

Returns

true
false

4.31.2.8 parse_compound_list()

```
bool parse_compound_list (
            struct parser * parser,
            struct node_compound_list ** ast )
```

parse rule compound list

Parameters

parser	
ast	

Returns

true
false

4.31.2.9 parse_do_group()

```
bool parse_do_group (
            struct parser * parser,
            struct node_do_group ** ast )
```

parse rule do group

Parameters

parser	
ast	

Returns

true
false

4.31.2.10 parse_element()

```
bool parse_element (
            struct parser * parser,
            struct node_element ** ast )
```

parse rule element

Parameters

parser	
ast	

Returns

true
false

4.31.2.11 parse_else_clause()

```
bool parse_else_clause (
            struct parser * parser,
            struct node_else_clause ** ast )
```

parse else clause

Parameters

parser	
ast	

Returns

true
false

4.31.2.12 parse_funcdec()

```
bool parse_funcdec (
            struct parser * parser,
            struct node_funcdec ** ast )
```

parse rule funcdec

Parameters

parser	
ast	

Returns

true
false

4.31.2.13 parse_input()

```
bool parse_input (
            struct parser * parser,
            struct node_input ** ast )
```

parse rule input

Parameters

parser	
ast	

Returns

true
false

4.31.2.14 parse_list()

```
bool parse_list (
            struct parser * parser,
            struct node_list ** ast )
```

parse rule list

Parameters

parser	
ast	

Returns

true
false

4.31.2.15 parse_look_ahead()

```
bool parse_look_ahead (
            struct parser * parser,
            struct token * expected_token )
```

look the next token without moving the list of tokens

Parameters

parser	
expected_token	

Returns

true
false

4.31.2.16 parse_pipeline()

```
bool parse_pipeline (
            struct parser * parser,
            struct node_pipeline ** ast )
```

parse rule pipeline

Parameters

parser	
ast	

Returns

true
false

4.31.2.17 parse_prefix()

```
bool parse_prefix (
            struct parser * parser,
            struct node_prefix ** ast )
```

parse rule prefix

Parameters

parser	
ast	

Returns

true
false

4.31.2.18 parse_redirection()

```
bool parse_redirection (
            struct parser * parser,
            struct node_redirection ** ast )
```

parse rule redirection

Parameters

parser	
ast	

Returns

true
false

4.31.2.19 parse_rule_case()

```
bool parse_rule_case (
            struct parser * parser,
            struct node_case ** ast )
```

parse rule case

Parameters

parser	
ast	

Returns

true
false

4.31.2.20 parse_rule_for()

```
bool parse_rule_for (
            struct parser * parser,
            struct node_for ** ast )
```

parse rule for

Parameters

parser	
ast	

Returns

true
false

4.31.2.21 parse_rule_if()

```
bool parse_rule_if (
            struct parser * parser,
            struct node_if ** ast )
```

parse rule if

Parameters

parser	
ast	

Returns

true
false

4.31.2.22 parse_rule_until()

```
bool parse_rule_until (
            struct parser * parser,
            struct node_until ** ast )
```

parse rule until

Parameters

parser	
ast	

Returns

true
false

4.31.2.23 parse_rule_while()

```
bool parse_rule_while (
            struct parser * parser,
            struct node_while ** ast )
```

parse rule while

Parameters

parser	
ast	

Returns

true
false

4.31.2.24 parse_shell_command()

```
bool parse_shell_command (
            struct parser * parser,
            struct node_shell_command ** ast )
```

parse rule shell command

Parameters

parser	
ast	

Returns

true
false

4.31.2.25 parse_simple_command()

```
bool parse_simple_command (
            struct parser * parser,
            struct node_simple_command ** ast )
```

parse rule simple command

Parameters

parser	
ast	

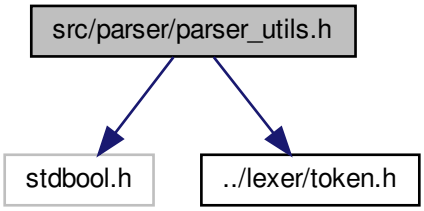
Returns

true
false

4.32 src/parser/parser_utils.h File Reference

```
#include <stdbool.h>
#include "../lexer/token.h"
```

Include dependency graph for parser_utils.h:



Functions

- bool is_redirection (struct token *token)

  check if there is a redirection

4.32.1 Function Documentation

4.32.1.1 is_redirection()

```
bool is_redirection (
            struct token * token )
```

check if there is a redirection

Parameters

token	

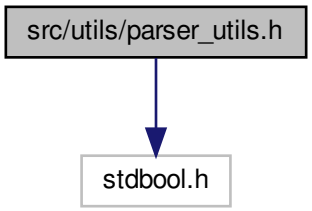
Returns

true
false

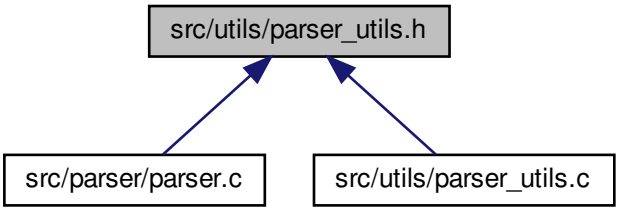
4.33 src/utils/parser_utils.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for parser_utils.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool is_redirection (struct token *token)

  Return true if the token is a redirection.
- struct node_prefix * append_prefix (struct node_simple_command *ast, struct node_prefix *prefix)

  Add prefix node to the prefix list of simple command node.
- struct node_element * append_element (struct node_simple_command *ast, struct node_element *element)

  Add element node to the element list of the simple command node.
- struct node_redirection * append_redirection (struct node_command *ast, struct node_redirection *redirection)

  Add redirection node to the redirection list of the command node.
- struct range * append_value_to_for (struct node_for *ast, char *value)

  Add new value to the range list of the for node.
- struct word_list * append_word_list (struct node_case_item *ast, char *value)

  Add new value to the pipeline list of the case item node.
- enum shell_type get_shell_command_type (int type)

  Get the shell command type object.

4.33.1 Function Documentation

4.33.1.1 append_element()

```
struct node_element* append_element (
            struct node_simple_command * ast,
            struct node_element * element )
```

Add element node to the element list of the simple command node.

Parameters

ast	
element	

Returns

struct node_element*

4.33.1.2 append_prefix()

```
struct node_prefix* append_prefix (
            struct node_simple_command * ast,
            struct node_prefix * prefix )
```

Add prefix node to the prefix list of simple command node.

Parameters

ast	
prefix	

Returns

struct node_prefix*

4.33.1.3 append_redirection()

```
struct node_redirection* append_redirection (
            struct node_command * ast,
            struct node_redirection * redirection )
```

Add redirection node to the redirection list of the command node.

Parameters

ast	
redirection	

Returns

struct node_redirection*

4.33.1.4 append_value_to_for()

```
struct range* append_value_to_for (
            struct node_for * ast,
            char * value )
```

Add new value to the range list of the for node.

Parameters

ast	
value	

Returns

struct range*

4.33.1.5 append_word_list()

```
struct word_list* append_word_list (
            struct node_case_item * ast,
            char * value )
```

Add new value to the pipeline list of the case item node.

Parameters

ast	
value	

Returns

struct word_list*

4.33.1.6 get_shell_command_type()

```
enum shell_type get_shell_command_type (
            int type )
```

Get the shell command type object.

Parameters

type	

Returns

enum shell_type

4.33.1.7 is_redirection()

```
bool is_redirection (
            struct token * token )
```

Return true if the token is a redirection.

Parameters

token	

Returns

true
false

Return true if the token is a redirection.

Parameters

token	

Returns

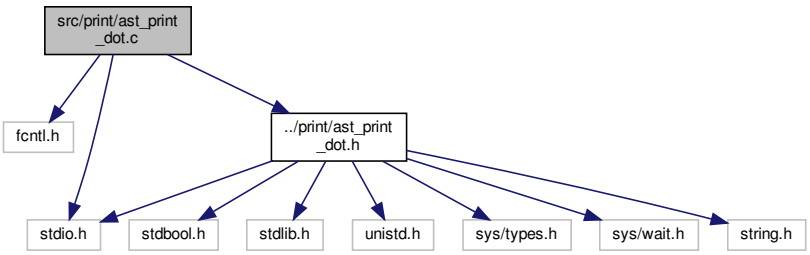
true
false

4.34 src/print/ast_print_dot.c File Reference

```
#include <fcntl.h>
#include <stdio.h>
```

```
#include "../print/ast_print_dot.h"
```

Include dependency graph for ast_print_dot.c:



Functions

- FILE * new_dot (void)

  create new dote file
- bool append_to_dot (FILE *dot_file, const char *str, bool is_new_line)

  append line to the dot file
- bool close_dot (FILE *dot_file)

  close dot file
- void convert_dot_to_png (void)

  convert file dot to png

4.34.1 Function Documentation

4.34.1.1 append_to_dot()

```
bool append_to_dot (
            FILE * dot_file,
            const char * str,
            bool is_new_line )
```

append line to the dot file

Parameters

dot_file	
str	
is_new_line	

Returns

true
false

4.34.1.2 close_dot()

```
bool close_dot (
            FILE * dot_file )
```

close dot file

Parameters

dot_file	

Returns

true
false

4.34.1.3 convert_dot_to_png()

```
void convert_dot_to_png (
            void  )
```

convert file dot to png

4.34.1.4 new_dot()

```
FILE* new_dot (
            void  )
```

create new dote file

Returns

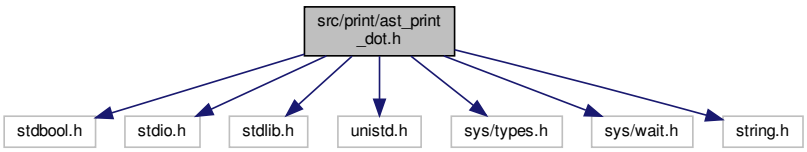
FILE*

4.35 src/print/ast_print_dot.h File Reference

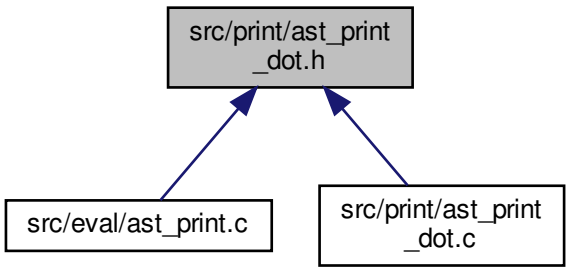
Dot file usage functions.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
```

Include dependency graph for ast_print_dot.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define DEFAULT_DOT_FILE_NAME "ast.dot"
- #define DEFAULT_PNG_FILE_NAME "ast.png"
- #define AST_STYLE_LOGIC "style=filled color=\"1.0 .3 .7\" fontname=\"Helvetica\" fontsize=12 "
- #define AST_STYLE_FUNCTION

Functions

- FILE * new_dot (void)
  create new dote file
- bool append_to_dot (FILE *dot_file, const char *str, bool is_new_line)
  append line to the dot file
- bool close_dot (FILE *dot_file)
  close dot file
- void convert_dot_to_png (void)
  convert file dot to png
- char * str (void *ptr)
  create string
- char * concat (char *arr[])
  concatenate string

4.35.1 Detailed Description

Dot file usage functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.35.2 Macro Definition Documentation

4.35.2.1 AST_STYLE_FUNCTION

```
#define AST_STYLE_FUNCTION
```

Value:

```
"style=filled,dotted "    \
  "fontname=\"Helvetica\" fontsize=9"
```

4.35.2.2 AST_STYLE_LOGIC

```
#define AST_STYLE_LOGIC "style=filled color=\"1.0 .3 .7\" fontname=\"Helvetica\" fontsize=12 "
```

4.35.2.3 DEFAULT_DOT_FILE_NAME

```
#define DEFAULT_DOT_FILE_NAME "ast.dot"
```

4.35.2.4 DEFAULT_PNG_FILE_NAME

```
#define DEFAULT_PNG_FILE_NAME "ast.png"
```

4.35.3 Function Documentation

4.35.3.1 append_to_dot()

```
bool append_to_dot (
            FILE * dot_file,
            const char * str,
            bool is_new_line )
```

append line to the dot file

Parameters

dot_file	
str	
is_new_line	

Returns

true
false

4.35.3.2 close_dot()

```
bool close_dot (
            FILE * dot_file )
```

close dot file

Parameters

dot_file	

Returns

true
false

4.35.3.3 concat()

```
char* concat (
            char * arr[] )
```

concatenate string

Parameters

arr	

Returns

char*

4.35.3.4 convert_dot_to_png()

```
void convert_dot_to_png (
            void  )
```

convert file dot to png

4.35.3.5 new_dot()

```
FILE* new_dot (
            void  )
```

create new dote file

Returns

FILE*

4.35.3.6 str()

```
char* str (
            void * ptr )
```

create string

Parameters

ptr	

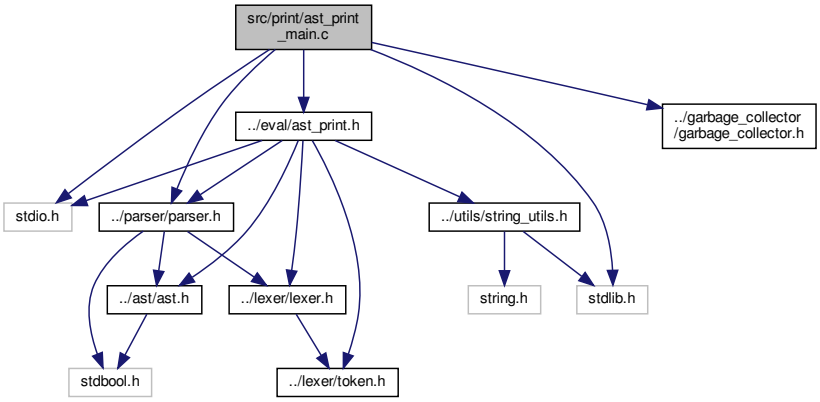
Returns

char*

4.36 src/print/ast_print_main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../parser/parser.h"
#include "../garbage_collector/garbage_collector.h"
#include "../eval/ast_print.h"
```

Include dependency graph for ast_print_main.c:



Functions

- int main (int argc, char *argv[])

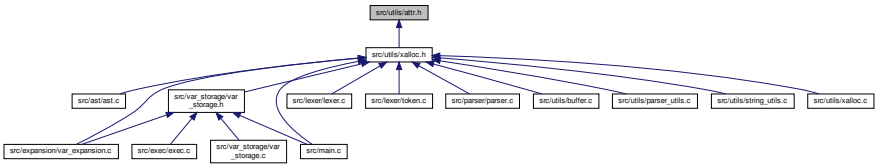
4.36.1 Function Documentation

4.36.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

4.37 src/utils/attr.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define ATTR(Att) __attribute__((Att))
- #define __malloc ATTR(malloc)

4.37.1 Macro Definition Documentation

4.37.1.1 __malloc

```
#define __malloc ATTR(malloc)
```

4.37.1.2 ATTR

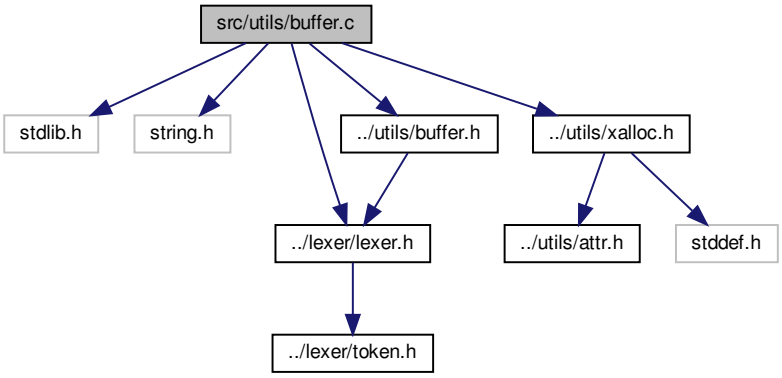
```
#define ATTR(
             Att ) __attribute__((Att))
```

4.38 src/utils/buffer.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "../lexer/lexer.h"
#include "../utils/buffer.h"
```

```
#include "../utils/xalloc.h"
```

Include dependency graph for buffer.c:



Functions

- struct buffer * new_buffer ()

  Create buffer.
- void append_buffer (struct buffer *buffer, char c)

  Append characters to the buffer.
- void append_string_to_buffer (struct buffer *buffer, char *str)

  Append string to the buffer.
- size_t buffer_len (struct buffer *buffer)

  Give the len of the buffer.
- void append_word_if_needed (struct lexer *lexer, struct buffer *buffer)

  Append word to buffer.
- void free_buffer (struct buffer *buffer)

  Free the buffer.
- void flush (struct buffer *buffer)

  Empty a string buffer.

4.38.1 Function Documentation

4.38.1.1 append_buffer()

```
void append_buffer (
            struct buffer * buffer,
            char c )
```

Append characters to the buffer.

Parameters

buffer	
c	

4.38.1.2 append_string_to_buffer()

```
void append_string_to_buffer (
             struct buffer * buffer,
             char * str )
```

Append string to the buffer.

Parameters

buffer	
str	

4.38.1.3 append_word_if_needed()

```
void append_word_if_needed (
             struct lexer * lexer,
             struct buffer * buffer )
```

Append word to buffer.

Parameters

lexer	
buffer	

4.38.1.4 buffer_len()

```
size_t buffer_len (
             struct buffer * buffer )
```

Give the len of the buffer.

Parameters

buffer	

Returns

size_t

4.38.1.5 flush()

```
void flush (
            struct buffer * buffer )
```

Empty a string buffer.

Parameters

buffer	the string to be clear.
size	the length of the buffer.

4.38.1.6 free_buffer()

```
void free_buffer (
            struct buffer * buffer )
```

Free the buffer.

Parameters

buffer	

4.38.1.7 new_buffer()

```
struct buffer* new_buffer ( )
```

Create buffer.

Returns

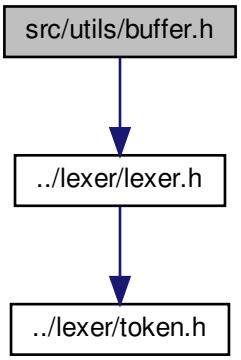
struct buffer*

4.39 src/utils/buffer.h File Reference

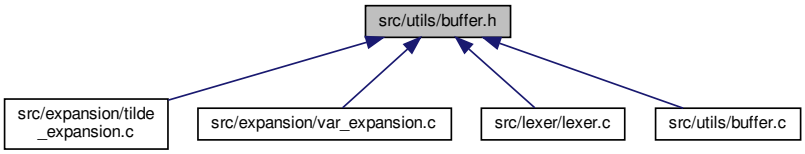
Buffer structure and functions.

```
#include "../lexer/lexer.h"
```

Include dependency graph for buffer.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct buffer

Macros

- #define BUFFER_SIZE 256

Functions

- struct buffer * new_buffer ()

  Create buffer.
- void append_buffer (struct buffer *buffer, char c)

  Append characters to the buffer.
- void append_string_to_buffer (struct buffer *buffer, char *str)

  Append string to the buffer.
- void free_buffer (struct buffer *buffer)

Free the buffer.

- size_t buffer_len (struct buffer *buffer)

  Give the len of the buffer.

- void append_word_if_needed (struct lexer *lexer, struct buffer *buffer)

  Append word to buffer.

- void flush (struct buffer *buffer)

  Empty a string buffer.

4.39.1 Detailed Description

Buffer structure and functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.39.2 Macro Definition Documentation

4.39.2.1 BUFFER_SIZE

```
#define BUFFER_SIZE 256
```

4.39.3 Function Documentation

4.39.3.1 append_buffer()

```
void append_buffer (
            struct buffer * buffer,
            char c )
```

Append characters to the buffer.

Parameters

buffer	
c	

4.39.3.2 append_string_to_buffer()

```
void append_string_to_buffer (
            struct buffer * buffer,
            char * str )
```

Append string to the buffer.

Parameters

buffer	
str	

4.39.3.3 append_word_if_needed()

```
void append_word_if_needed (
            struct lexer * lexer,
            struct buffer * buffer )
```

Append word to buffer.

Parameters

lexer	
buffer	

4.39.3.4 buffer_len()

```
size_t buffer_len (
            struct buffer * buffer )
```

Give the len of the buffer.

Parameters

buffer	

Returns

size_t

4.39.3.5 flush()

```
void flush (
            struct buffer * buffer )
```

Empty a string buffer.

Parameters

buffer	the string to be clear.
size	the length of the buffer.

4.39.3.6 free_buffer()

```
void free_buffer (
            struct buffer * buffer )
```

Free the buffer.

Parameters

buffer	

4.39.3.7 new_buffer()

```
struct buffer* new_buffer ( )
```

Create buffer.

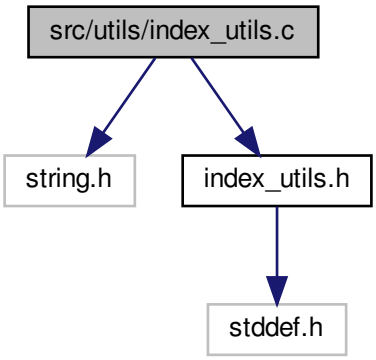
Returns

struct buffer*

4.40 src/utils/index_utils.c File Reference

```
#include <string.h>
#include "index_utils.h"
```

Include dependency graph for index_utils.c:



Functions

- int is_separator (char c)
- size_t get_next_index (const char *str, char c, size_t i)
- size_t get_previous_index (const char *str, char c, size_t i)
- size_t get_previous_separator_index (const char *str, size_t i)
- size_t get_next_separator_index (const char *str, size_t i)
- size_t get_next_close_curl_index (const char *str, size_t i)

4.40.1 Function Documentation

4.40.1.1 get_next_close_curl_index()

```
size_t get_next_close_curl_index (
            const char * str,
            size_t i )
```

4.40.1.2 get_next_index()

```
size_t get_next_index (
            const char * str,
            char c,
            size_t i )
```

4.40.1.3 get_next_separator_index()

```
size_t get_next_separator_index (
            const char * str,
            size_t i )
```

4.40.1.4 get_previous_index()

```
size_t get_previous_index (
            const char * str,
            char c,
            size_t i )
```

4.40.1.5 get_previous_separator_index()

```
size_t get_previous_separator_index (
            const char * str,
            size_t i )
```

4.40.1.6 is_separator()

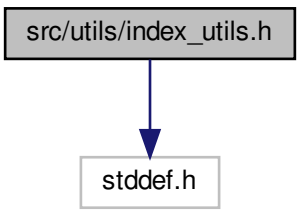
```
int is_separator (
            char c )
```

4.41 src/utils/index_utils.h File Reference

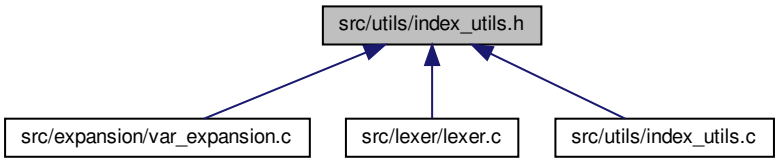
Index functions.

```
#include <stddef.h>
```

Include dependency graph for index_utils.h:



This graph shows which files directly or indirectly include this file:



Functions

- int is_separator (char c)
- size_t get_next_index (const char *str, char c, size_t i)
- size_t get_previous_index (const char *str, char c, size_t i)
- size_t get_previous_separator_index (const char *str, size_t j)
- size_t get_next_separator_index (const char *c, size_t j)
- size_t get_next_close_curl_index (const char *str, size_t j)

4.41.1 Detailed Description

Index functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.41.2 Function Documentation

4.41.2.1 get_next_close_curl_index()

```
size_t get_next_close_curl_index (
            const char * str,
            size_t j )
```

4.41.2.2 get_next_index()

```
size_t get_next_index (
            const char * str,
            char c,
            size_t i )
```

4.41.2.3 get_next_separator_index()

```
size_t get_next_separator_index (
            const char * c,
            size_t j )
```

4.41.2.4 get_previous_index()

```
size_t get_previous_index (
            const char * str,
            char c,
            size_t i )
```

4.41.2.5 get_previous_separator_index()

```
size_t get_previous_separator_index (
            const char * str,
            size_t j )
```

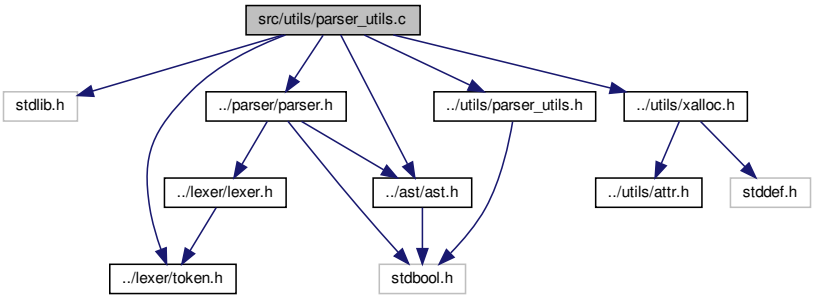
4.41.2.6 is_separator()

```
int is_separator (
            char c )
```

4.42 src/utils/parser_utils.c File Reference

```
#include <stdlib.h>
#include "../lexer/token.h"
#include "../parser/parser.h"
#include "../ast/ast.h"
#include "../utils/parser_utils.h"
#include "../utils/xalloc.h"
```

Include dependency graph for parser_utils.c:



Functions

- bool is_redirection (struct token *token)

  check if there is a redirection
- struct node_prefix * append_prefix (struct node_simple_command *ast, struct node_prefix *prefix)

  Add prefix node to the prefix list of simple command node.
- struct node_element * append_element (struct node_simple_command *ast, struct node_element *element)

  Add element node to the element list of the simple command node.
- struct node_redirection * append_redirection (struct node_command *ast, struct node_redirection *redirection)

  Add redirection node to the redirection list of the command node.
- struct range * append_value_to_for (struct node_for *ast, char *value)

  Add new value to the range list of the for node.
- struct word_list * append_word_list (struct node_case_item *ast, char *value)

  Add new value to the pipeline list of the case item node.
- enum shell_type get_shell_command_type (int type)

  Get the shell command type object.

4.42.1 Function Documentation

4.42.1.1 append_element()

```
struct node_element* append_element (
            struct node_simple_command * ast,
            struct node_element * element )
```

Add element node to the element list of the simple command node.

Parameters

ast	
element	

Returns

struct node_element*

4.42.1.2 append_prefix()

```
struct node_prefix* append_prefix (
            struct node_simple_command * ast,
            struct node_prefix * prefix )
```

Add prefix node to the prefix list of simple command node.

Parameters

ast	
prefix	

Returns

struct node_prefix*

4.42.1.3 append_redirection()

```
struct node_redirection* append_redirection (
            struct node_command * ast,
            struct node_redirection * redirection )
```

Add redirection node to the redirection list of the command node.

Parameters

ast	
redirection	

Returns

struct node_redirection*

4.42.1.4 append_value_to_for()

```
struct range* append_value_to_for (
            struct node_for * ast,
            char * value )
```

Add new value to the range list of the for node.

Parameters

ast	
value	

Returns

struct range*

4.42.1.5 append_word_list()

```
struct word_list* append_word_list (
            struct node_case_item * ast,
            char * value )
```

Add new value to the pipeline list of the case item node.

Parameters

ast	
value	

Returns

struct word_list*

4.42.1.6 get_shell_command_type()

```
enum shell_type get_shell_command_type (
            int type )
```

Get the shell command type object.

Parameters

type	

Returns

enum shell_type

4.42.1.7 is_redirection()

```
bool is_redirection (
            struct token * token )
```

check if there is a redirection

Return true if the token is a redirection.

Parameters

token	

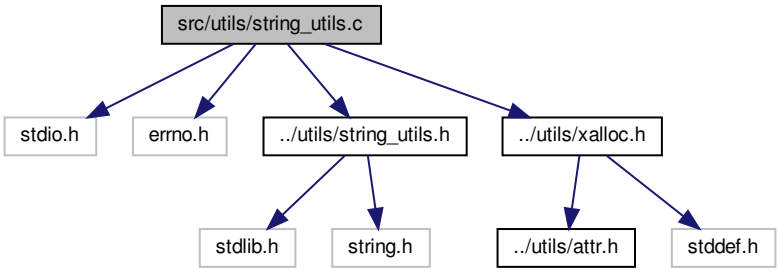
Returns

true
false

4.43 src/utils/string_utils.c File Reference

```
#include <stdio.h>
#include <errno.h>
#include "../utils/string_utils.h"
#include "../utils/xalloc.h"
```

Include dependency graph for string_utils.c:



Functions

- char * type_to_str (int type)

  Return the associated string of a token type.

- int is (const char *a, const char *b)

  Return true is a == b.
- int is_number (char c)

  Return true is c is a number.
- char * substr (char *src, int pos, int len)

  Return the substring between pos and len - 1.
- char * my_strdup (const char *c)
- void error (char *msg)

  Print an error in stderr when an invalid token appeared.

4.43.1 Function Documentation

4.43.1.1 error()

```
void error (
            char * msg )
```

Print an error in stderr when an invalid token appeared.

Parameters

msg	the message to display.

4.43.1.2 is()

```
int is (
            const char * a,
            const char * b )
```

Return true is a == b.

Parameters

a	the first string to be compared.
b	the decond string to be compared.

4.43.1.3 is_number()

```
int is_number (
            char c )
```

Return true is c is a number.

Parameters

c	the caracter.

4.43.1.4 my_strdup()

```
char* my_strdup (
            const char * c )
```

4.43.1.5 substr()

```
char* substr (
            char * src,
            int pos,
            int len )
```

Return the substring between pos and len - 1.

Parameters

src	the string.
pos	the starting index.
len	the ending index.

4.43.1.6 type_to_str()

```
char* type_to_str (
            int type )
```

Return the associated string of a token type.

Parameters

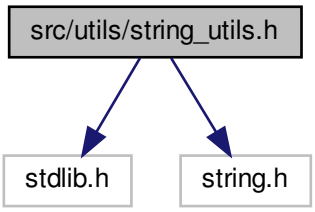
type	the enum value of the token.

4.44 src/utils/string_utils.h File Reference

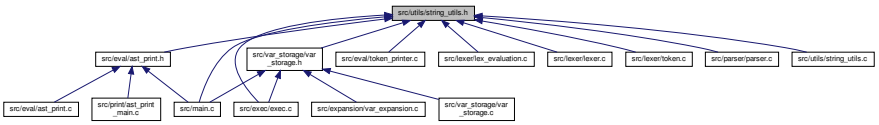
String usage functions.

```
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for string_utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define MAX_STR_LEN 256

Functions

- char * type_to_str (int type)

  Return the associated string of a token type.
- int is (const char *a, const char *b)

  Return true is a == b.
- int is_number (char c)

  Return true is c is a number.
- char * substr (char *src, int pos, int len)

  Return the substring between pos and len - 1.
- void error (char *msg)

  Print an error in stderr when an invalid token appeared.
- char * my_strdup (const char *c)

4.44.1 Detailed Description

String usage functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

4.44.2 Macro Definition Documentation

4.44.2.1 MAX_STR_LEN

```
#define MAX_STR_LEN 256
```

4.44.3 Function Documentation

4.44.3.1 error()

```
void error (
            char * msg )
```

Print an error in stderr when an invalid token appeared.

Parameters

msg	the message to display.

4.44.3.2 is()

```
int is (
            const char * a,
            const char * b )
```

Return true is a == b.

Parameters

a	the first string to be compared.
b	the decond string to be compared.

4.44.3.3 is_number()

```
int is_number (
            char c )
```

Return true is c is a number.

Parameters

c	the caracter.

4.44.3.4 my_strdup()

```
char* my_strdup (
            const char * c )
```

4.44.3.5 substr()

```
char* substr (
            char * src,
            int pos,
            int len )
```

Return the substring between pos and len - 1.

Parameters

src	the string.
pos	the starting index.
len	the ending index.

4.44.3.6 type_to_str()

```
char* type_to_str (
            int type )
```

Return the associated string of a token type.

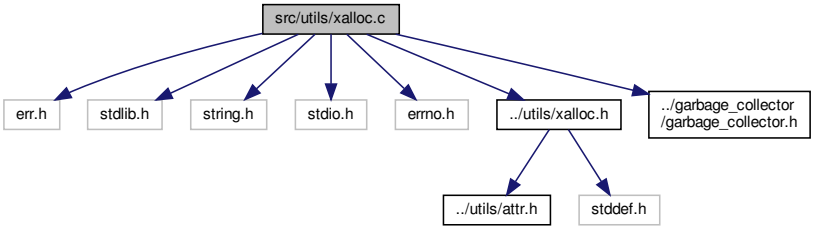
Parameters

type	the enum value of the token.

4.45 src/utils/xalloc.c File Reference

```
#include <err.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include "../utils/xalloc.h"
#include "../garbage_collector/garbage_collector.h"
```

Include dependency graph for xalloc.c:



Functions

- void * xmalloc (size_t size)

  Safe malloc wrapper.
- void * xrealloc (void *ptr, size_t size)

  Safe realloc wrapper.
- void * xcalloc (size_t nmb, size_t size)

4.45.1 Function Documentation

4.45.1.1 xcalloc()

```
void* xcalloc (
            size_t nmb,
            size_t size )
```

4.45.1.2 xmalloc()

```
void* xmalloc (
            size_t size )
```

Safe malloc wrapper.

Parameters

size	the size to allocate

Returns

a pointer to the allocated memory

4.45.1.3 xrealloc()

```
void* xrealloc (
            void * ptr,
            size_t size )
```

Safe realloc wrapper.

Parameters

ptr	the pointer to reallocate
size	the new size to allocate

Returns

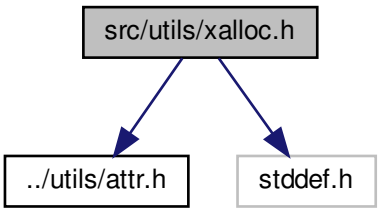
a pointer to the allocated memory

4.46 src/utils/xalloc.h File Reference

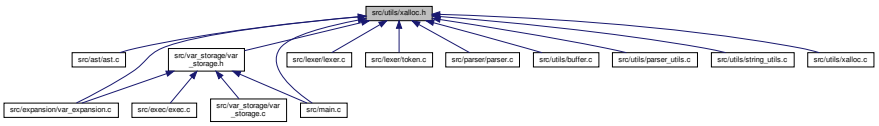
Special allocation functions.

```
#include "../utils/attr.h"
#include <stddef.h>
```

Include dependency graph for xalloc.h:



This graph shows which files directly or indirectly include this file:



Functions

- void * xmalloc (size_t size) __malloc

  Safe malloc wrapper.
- void * xrealloc (void *ptr, size_t size)

  Safe realloc wrapper.
- void * xcalloc (size_t nmb, size_t size)

4.46.1 Detailed Description

Special allocation functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.46.2 Function Documentation

4.46.2.1 xcalloc()

```
void* xcalloc (
            size_t nmb,
            size_t size )
```

4.46.2.2 xmalloc()

```
void* xmalloc (
            size_t size )
```

Safe malloc wrapper.

Parameters

size	the size to allocate

Returns

a pointer to the allocated memory

4.46.2.3 xrealloc()

```
void* xrealloc (
            void * ptr,
            size_t size )
```

Safe realloc wrapper.

Parameters

ptr	the pointer to reallocate
size	the new size to allocate

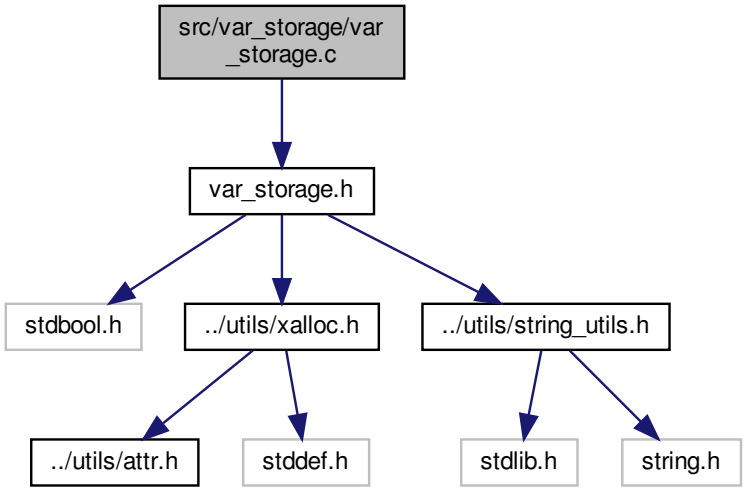
Returns

a pointer to the allocated memory

4.47 src/var_storage/var_storage.c File Reference

```
#include "var_storage.h"
```

Include dependency graph for var_storage.c:



Functions

- struct var_storage * new_var_storage (void)
- void free_var_storage (void)
- int hash (char *key)
- bool var_exists (char *key)
- bool put_var (char *key, char *val)
- struct variable * get_var (char *key)
- char * get_value (char *key)
- enum var_type get_var_type (char *value)

4.47.1 Function Documentation

4.47.1.1 free_var_storage()

```
void free_var_storage (
            void  )
```

4.47.1.2 get_value()

```
char* get_value (
            char * key )
```

4.47.1.3 get_var()

```
struct variable* get_var (
            char * key )
```

4.47.1.4 get_var_type()

```
enum var_type get_var_type (
            char * value )
```

4.47.1.5 hash()

```
int hash (
            char * key )
```

4.47.1.6 new_var_storage()

```
struct var_storage* new_var_storage (
            void  )
```

4.47.1.7 put_var()

```
bool put_var (
            char * key,
            char * val )
```

4.47.1.8 var_exists()

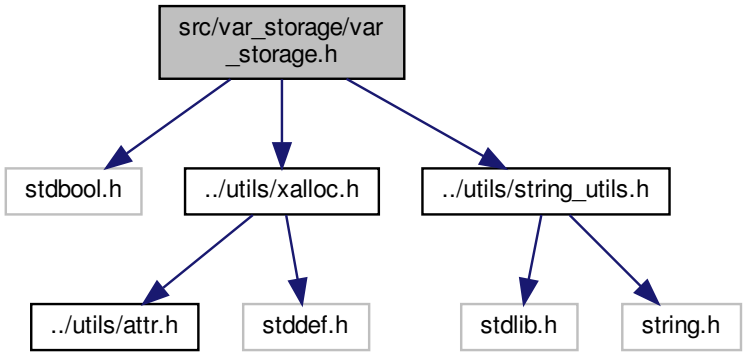
```
bool var_exists (
            char * key )
```

4.48 src/var_storage/var_storage.h File Reference

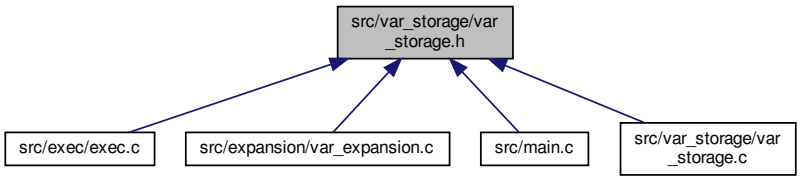
Var storage structures and functions.

```
#include <stdbool.h>
#include "../utils/xalloc.h"
#include "../utils/string_utils.h"
```

Include dependency graph for var_storage.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct variable
- struct var_storage

Macros

- #define STORAGE_SIZE 2048

Enumerations

- enum var_type { VAR_INT, VAR_FLOAT, VAR_STRING, VAR_ERROR }

Functions

- struct var_storage * new_var_storage (void)
- void free_var_storage (void)
- bool var_exists (char *key)
- enum var_type get_var_type (char *value)
- bool put_var (char *key, char *val)
- struct variable * get_var (char *key)
- char * get_value (char *key)

Variables

- struct var_storage * var_storage

4.48.1 Detailed Description

Var storage structures and functions.

Author

Team

Version

0.1

Date

2020-05-03

Copyright

Copyright (c) 2020

4.48.2 Macro Definition Documentation

4.48.2.1 STORAGE_SIZE

```
#define STORAGE_SIZE 2048
```

4.48.3 Enumeration Type Documentation

4.48.3.1 var_type

```
enum var_type
```

Enumerator

VAR_INT	
VAR_FLOAT	
VAR_STRING	
VAR_ERROR	

4.48.4 Function Documentation

4.48.4.1 free_var_storage()

```
void free_var_storage (
            void  )
```

4.48.4.2 get_value()

```
char* get_value (
            char * key )
```

4.48.4.3 get_var()

```
struct variable* get_var (
            char * key )
```

4.48.4.4 get_var_type()

```
enum var_type get_var_type (
            char * value )
```

4.48.4.5 new_var_storage()

```
struct var_storage* new_var_storage (
            void  )
```

4.48.4.6 put_var()

```
bool put_var (
             char * key,
             char * val )
```

4.48.4.7 var_exists()

```
bool var_exists (
             char * key )
```

4.48.5 Variable Documentation

4.48.5.1 var_storage

```
struct var_storage* var_storage
```