

haigo
0.1

Generated by Doxygen 1.7.1

Tue Jan 17 2012 16:51:52

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	Go Text Protocol Commands	7
4.1.1	Detailed Description	7
4.2	Go Text Protocol Administrative Commands	8
4.2.1	Function Documentation	8
4.2.1.1	gtp_known_command	8
4.2.1.2	gtp_list_commands	9
4.2.1.3	gtp_name	10
4.2.1.4	gtp_protocol_version	10
4.2.1.5	gtp_quit	11
4.2.1.6	gtp_version	11
4.3	Go Text Protocol Setup Commands	12
4.3.1	Function Documentation	12
4.3.1.1	gtp_boardsize	12
4.3.1.2	gtp_clear_board	13
4.3.1.3	gtp_komi	14
4.4	Go Text Protocol Core Play Commands	14
4.4.1	Function Documentation	14
4.4.1.1	gtp_play	14
4.5	Go Text Protocol Debug Commands	16

4.5.1	Function Documentation	16
4.5.1.1	gtp_showboard	16
5	Data Structure Documentation	17
5.1	command Struct Reference	17
5.1.1	Detailed Description	17
5.1.2	Field Documentation	17
5.1.2.1	argc	17
5.1.2.2	argv	17
5.1.2.3	id	17
5.1.2.4	name	17
5.2	command_func Struct Reference	18
5.2.1	Detailed Description	18
5.2.2	Field Documentation	18
5.2.2.1	command	18
5.2.2.2	function	18
6	File Documentation	19
6.1	src/board.c File Reference	19
6.1.1	Function Documentation	20
6.1.1.1	free_board	20
6.1.1.2	get_board_as_string	20
6.1.1.3	get_board_size	21
6.1.1.4	get_label_x	21
6.1.1.5	get_label_y_left	22
6.1.1.6	get_label_y_right	22
6.1.1.7	init_board	22
6.1.1.8	is_hoshi	23
6.1.1.9	set_vertex	23
6.1.2	Variable Documentation	24
6.1.2.1	board	24
6.1.2.2	board_size	24
6.1.2.3	hoshi	24
6.2	src/board.h File Reference	24
6.2.1	Function Documentation	24
6.2.1.1	free_board	24
6.2.1.2	get_board_as_string	25

6.2.1.3	get_board_size	26
6.2.1.4	init_board	26
6.2.1.5	set_vertex	27
6.3	src/global_const.h File Reference	28
6.3.1	Define Documentation	29
6.3.1.1	BLACK	29
6.3.1.2	BLACK_STONE	29
6.3.1.3	BOARD_SIZE_DEF	29
6.3.1.4	BOARD_SIZE_MAX	29
6.3.1.5	BOARD_SIZE_MIN	29
6.3.1.6	EMPTY	29
6.3.1.7	FIELD_EMPTY	30
6.3.1.8	FIELD_HOSHI	30
6.3.1.9	MAX_OUTPUT_LENGTH	30
6.3.1.10	MAX_TOKEN_COUNT	30
6.3.1.11	MAX_TOKEN_LENGTH	30
6.3.1.12	WHITE	30
6.3.1.13	WHITE_STONE	30
6.4	src/io.c File Reference	30
6.4.1	Function Documentation	31
6.4.1.1	add_output	31
6.4.1.2	drop_comment	32
6.4.1.3	get_output_error	32
6.4.1.4	identify_tokens	32
6.4.1.5	init_tokens	33
6.4.1.6	is_input_empty	34
6.4.1.7	parse_gtp_input	34
6.4.1.8	print_output	35
6.4.1.9	read_gtp_input	35
6.4.1.10	set_output_error	36
6.4.1.11	trim	36
6.4.2	Variable Documentation	37
6.4.2.1	command_input_buffer	37
6.4.2.2	input_empty	37
6.4.2.3	output	37
6.4.2.4	output_error	37

6.5	src/io.h File Reference	37
6.5.1	Define Documentation	39
6.5.1.1	SIZE_INPUT_BUFFER	39
6.5.2	Function Documentation	39
6.5.2.1	add_output	39
6.5.2.2	drop_comment	39
6.5.2.3	get_output_error	40
6.5.2.4	identify_tokens	40
6.5.2.5	init_tokens	41
6.5.2.6	is_input_empty	41
6.5.2.7	parse_gtp_input	41
6.5.2.8	print_output	42
6.5.2.9	read_gtp_input	43
6.5.2.10	set_output_error	43
6.5.2.11	trim	44
6.6	src/main.c File Reference	44
6.6.1	Function Documentation	45
6.6.1.1	main	45
6.7	src/run_program.c File Reference	46
6.7.1	Function Documentation	47
6.7.1.1	init_known_commands	47
6.7.1.2	print_help_message	48
6.7.1.3	print_version	48
6.7.1.4	read_opts	48
6.7.1.5	run	49
6.7.1.6	select_command	50
6.7.1.7	set_quit_program	50
6.7.1.8	str_toupper	51
6.7.2	Variable Documentation	51
6.7.2.1	GTP_VERSION	51
6.7.2.2	help_message	51
6.7.2.3	known_commands	51
6.7.2.4	komi	51
6.7.2.5	PROGRAM_NAME	52
6.7.2.6	PROGRAM_VERSION	52
6.7.2.7	quit_program	52

6.8	src/run_program.h File Reference	52
6.8.1	Define Documentation	53
6.8.1.1	COUNT_KNOWN_COMMANDS	53
6.8.2	Function Documentation	53
6.8.2.1	run	53

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Go Text Protocol Commands	7
Go Text Protocol Administrative Commands	8
Go Text Protocol Setup Commands	12
Go Text Protocol Core Play Commands	14
Go Text Protocol Debug Commands	16

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

<code>command</code>	17
<code>command_func</code>	18

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

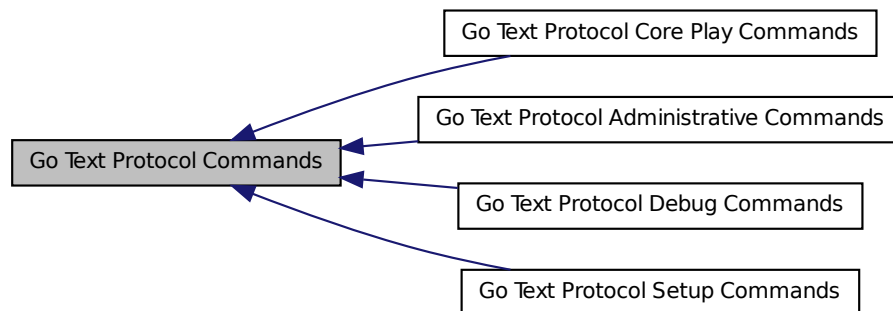
src/ board.c	19
src/ board.h	24
src/ global_const.h	28
src/ io.c	30
src/ io.h	37
src/ main.c	44
src/ run_program.c	46
src/ run_program.h	52

Chapter 4

Module Documentation

4.1 Go Text Protocol Commands

Collaboration diagram for Go Text Protocol Commands:



Modules

- [Go Text Protocol Administrative Commands](#)
- [Go Text Protocol Setup Commands](#)
- [Go Text Protocol Core Play Commands](#)
- [Go Text Protocol Debug Commands](#)

4.1.1 Detailed Description

The following functions are implemented as defined in the Go Text Protocol version 2

4.2 Go Text Protocol Administrative Commands

Collaboration diagram for Go Text Protocol Administrative Commands:



Functions

- void [gtp_quit](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Quits the program.
- void [gtp_version](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows the program's version number.
- void [gtp_protocol_version](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows the used GTP version number.
- void [gtp_name](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows the program's name.
- void [gtp_known_command](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows whether a given command is known or not.
- void [gtp_list_commands](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows a list of all know GTP commands.

4.2.1 Function Documentation

4.2.1.1 void [gtp_known_command](#) (int *argc*, char *argv*[][MAX_TOKEN_LENGTH])

Shows whether a given command is known or not.

[gtp_known_command\(\)](#) <command_name> returns either the string "true" or "false" therefore showing whether a given GTP command is known or not.

Parameters

- argc* Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 388 of file run_program.c.

```

int i;
bool is_command_known = false;

if ( argc <= 0 ) {
    set_output_error();
    add_output( "missing argument: command_name" );
    return;
}
if ( argc > 1 ) {
    set_output_error();
    add_output( "only one argument required: command_name" );
    return;
}

for ( i = 0; i < COUNT_KNOWN_COMMANDS; i++ ) {
    if ( strcmp( known_commands[i].command, argv[0] ) == 0 ) {
        is_command_known = true;
        break;
    }
}

if ( is_command_known == true ) {
    add_output("true");
}
else {
    add_output("false");
}

return;
}

```

4.2.1.2 void gtp_list_commands (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows a list of all know GTP commands.

[gtp_list_commands\(\)](#) shows the name of the program as defined by PROGRAM_NAME.

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 433 of file run_program.c.

```

{

```

```
int i;

for ( i = 0; i < COUNT_KNOWN_COMMANDS; i++ ) {
    add_output ( known_commands[i].command );
}

return;
}
```

4.2.1.3 void gtp_name (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows the program's name.

[gtp_name\(\)](#) shows the name of the program as defined by PROGRAM_NAME.

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 367 of file run_program.c.

```

{

    add_output (PROGRAM_NAME);

    return;
}
```

4.2.1.4 void gtp_protocol_version (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows the used GTP version number.

[gtp_protocol_version\(\)](#) shows the currently used Go Text Protocol version number. Currently this is version number 2 as defined in GTP_VERSION.

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 348 of file run_program.c.

```

{

    add_output (GTP_VERSION) ;

    return;
}
```

4.2.1.5 void gtp_quit (int argc, char argv[][MAX_TOKEN_LENGTH])

Quits the program.

[gtp_quit\(\)](#) quits the whole program by calling [set_quit_program\(\)](#).

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 308 of file run_program.c.

```

{

    set_quit_program();

    return;
}
```

4.2.1.6 void gtp_version (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows the program's version number.

[gtp_version\(\)](#) shows the version number as defined by PROGRAM_VERSION.

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.1 Administrative Commands

Definition at line 327 of file run_program.c.

```

{

    add_output (PROGRAM_VERSION);

    return;
}

```

4.3 Go Text Protocol Setup Commands

Collaboration diagram for Go Text Protocol Setup Commands:



Functions

- void [gtp_boardsize](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Changes the current board size.
- void [gtp_clear_board](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Clears the board.
- void [gtp_komi](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Sets komi.

4.3.1 Function Documentation

4.3.1.1 void [gtp_boardsize](#) (int *argc*, char *argv*[][MAX_TOKEN_LENGTH])

Changes the current board size.

[gtp_boardsize\(\)](#) changes the current size of the board. by PROGRAM_NAME.

Parameters

- argc* Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.2 Setup Commands

Definition at line 459 of file run_program.c.

```
int board_size = atoi( argv[0] );

if ( board_size < BOARD_SIZE_MIN || board_size > BOARD_SIZE_MAX ) {
    set_output_error();
    add_output("unacceptable size");
    return;
}

free_board();
init_board(board_size);

return;
}
```

4.3.1.2 void gtp_clear_board (int argc, char argv[][MAX_TOKEN_LENGTH])

Clears the board.

[gtp_clear_board\(\)](#) clears the board. The number of captured stones is set to zero for both colors. The move history is reset to empty.

Parameters

argc Number of arguments of GTP command

argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.2 Setup Commands

Definition at line 488 of file run_program.c.

```
int board_size = get_board_size();

free_board();
init_board(board_size);

// number of captured stones must be set to zero
// move history must be emptied

return;
}
```

4.3.1.3 void gtp_komi (int argc, char argv[][MAX_TOKEN_LENGTH])

Sets komi.

[gtp_komi\(\)](#) sets the komi to the given value.

Parameters

argc Number of arguments of GTP command
argv Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.2 Setup Commands

Definition at line 512 of file run_program.c.

```

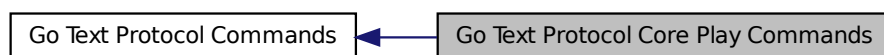
{
    // Check arg here!

    komi = atof( argv[0] );

    return;
}
```

4.4 Go Text Protocol Core Play Commands

Collaboration diagram for Go Text Protocol Core Play Commands:



Functions

- void [gtp_play](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
Description missing!

4.4.1 Function Documentation

4.4.1.1 void gtp_play (int argc, char argv[][MAX_TOKEN_LENGTH])

Description missing!

[gtp_play\(\)](#) Description missing!

Parameters

- argc* Number of arguments of GTP command
- argv* Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protocol version 2, 6.3.3 Core Play Commands

Definition at line 536 of file run_program.c.

```

{
    int color;
    int i, j;

    // Check if first argument is black or white:
    str_toupper( argv[0] );
    if ( strcmp( argv[0], "B" ) == 0 || strcmp( argv[0], "BLACK" ) == 0 ) {
        color = BLACK;
    }
    else if ( strcmp( argv[0], "W" ) == 0 || strcmp( argv[0], "WHITE" ) == 0 ) {
        color = WHITE;
    }
    else {
        set_output_error();
        add_output("invalid color");
        return;
    }

    // Check vertex if first coordinate is valid:
    i = (int) toupper( argv[1][0] ) - 65;
    if ( i > 8 ) {
        i--;
    }
    if ( i < 0 || i >= get_board_size() ) {
        set_output_error();
        add_output("invalid coordinate");
        return;
    }

    // Check if second coordinate is valid:
    argv[1][0] = ' ';
    j = atoi( argv[1] );
    j--;
    if ( j < 0 || j >= get_board_size() ) {
        set_output_error();
        add_output("invalid coordinate");
        return;
    }

    set_vertex( color, i, j );

    // Remove captured stones here ...
    // Update move history ...

    return;
}
```

4.5 Go Text Protocol Debug Commands

Collaboration diagram for Go Text Protocol Debug Commands:



Functions

- void [gtp_showboard](#) (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows a simple ASCII board.

4.5.1 Function Documentation

4.5.1.1 void gtp_showboard (int argc, char argv[][MAX_TOKEN_LENGTH])

Shows a simple ASCII board.

[gtp_showboard\(\)](#) gets a string representation of the board and sends it to the board_output variable, so it can then be printed.

Parameters

- argc** Number of arguments of GTP command
- argv** Array of all arguments for GTP command

Returns

nothing

See also

Go Text Protokol version 2, 6.3.6 Debug Commands

Definition at line 599 of file run_program.c.

```

char board_output[MAX_OUTPUT_LENGTH];

get_board_as_string(board_output);
add_output(board_output);

return;
}

```


Chapter 5

Data Structure Documentation

5.1 command Struct Reference

```
#include <io.h>
```

Data Fields

- int [id](#)
- char [name](#) [MAX_TOKEN_LENGTH]
- char [argv](#) [MAX_TOKEN_COUNT][MAX_TOKEN_LENGTH]
- int [argc](#)

5.1.1 Detailed Description

Definition at line 9 of file io.h.

5.1.2 Field Documentation

5.1.2.1 int argc

Definition at line 13 of file io.h.

5.1.2.2 char argv[MAX_TOKEN_COUNT][MAX_TOKEN_LENGTH]

Definition at line 12 of file io.h.

5.1.2.3 int id

Definition at line 10 of file io.h.

5.1.2.4 char name[MAX_TOKEN_LENGTH]

Definition at line 11 of file io.h.

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

- [src/io.h](#)

5.2 `command_func` Struct Reference

Data Fields

- `char * command`
- `void(* function)(int argc, char argv[][MAX_TOKEN_LENGTH])`

5.2.1 Detailed Description

Definition at line 26 of file `run_program.c`.

5.2.2 Field Documentation

5.2.2.1 `char* command`

Definition at line 27 of file `run_program.c`.

5.2.2.2 `void(* function)(int argc, char argv[][MAX_TOKEN_LENGTH])`

Definition at line 28 of file `run_program.c`.

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

- [src/run_program.c](#)

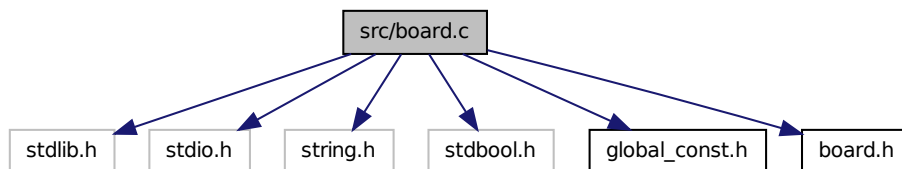
Chapter 6

File Documentation

6.1 src/board.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "global_const.h"
#include "board.h"
```

Include dependency graph for board.c:



Functions

- void [get_label_x](#) (int i, char x[])
- void [get_label_y_left](#) (int i, char x[])
- void [get_label_y_right](#) (int j, char y[])
- bool [is_hoshi](#) (int i, int j)
- void [set_vertex](#) (int color, int i, int j)
- void [init_board](#) (unsigned short wanted_board_size)
- void [free_board](#) (void)
- void [get_board_as_string](#) (char board_output[])
- int [get_board_size](#) (void)

Variables

- int ** [board](#)
- bool ** [hoshi](#)
- int [board_size](#) = 0

6.1.1 Function Documentation

6.1.1.1 void free_board (void)

Definition at line 98 of file board.c.

```

{
    int i;

    for ( i = 0; i < board_size; i++ ) {
        free(board[i]);
        free(hoshi[i]);
    }
    free(board);
    free(hoshi);

    return;
}

```

6.1.1.2 void get_board_as_string (char *board_output*[])

Definition at line 120 of file board.c.

```

{
    int i;          // Index for x-axis
    int j;          // Index for y-axis
    char x[3];      // Label for x-axis
    char y[2];      // Label for y-axis

    board_output[0] = '\0';
    strcat( board_output, "\n" );

    /* Print uppercase letters above the board */
    strcat( board_output, " " );
    for ( i = 0; i < board_size; i++ ) {
        get_label_x( i, x );
        strcat( board_output, " " );
        strcat( board_output, x );
    }
    strcat( board_output, "\n" );

    for ( j = board_size - 1; j >= 0; j-- ) {

        /* Print numbers left of board */
        get_label_y_left( j, y );
        strcat( board_output, " " );
        strcat( board_output, y );

        /* Print board fields */
        for ( i = 0; i < board_size; i++ ) {
            strcat( board_output, " " );
            switch ( board[i][j] ) {
                case WHITE:
                    strcat( board_output, WHITE_STONE );

```

```

        break;
    case BLACK:
        strcat( board_output, BLACK_STONE );
        break;
    case EMPTY:
        switch ( is_hoshi( i, j ) ) {
            case true:
                strcat( board_output, FIELD_HOSHI );
                break;
            case false:
                strcat( board_output, FIELD_EMPTY );
                break;
        }
        break;
    }
}

/* Print numbers right of board */
get_label_y_right( j, y );
strcat( board_output, " " );
strcat( board_output, y );
strcat( board_output, "\n" );
}

/* Print uppercase letters below board */
strcat( board_output, " " );
for ( i = 0; i < board_size; i++ ) {
    get_label_x( i, x );
    strcat( board_output, " " );
    strcat( board_output, x );
}

return;
}

```

6.1.1.3 int get_board_size (void)

Definition at line 232 of file board.c.

```

{

    return board_size;
}

```

6.1.1.4 void get_label_x (int i, char x[])

Definition at line 186 of file board.c.

```

{

    if ( i >= 8 ) {
        i++;
    }
    i += 65;
    x[0] = (char) i;
    x[1] = '\0';

    return;
}

```

6.1.1.5 void get_label_y_left (int i, char x[])

Definition at line 198 of file board.c.

```

{

    j++;

    y[0] = (char)(int)( j / 10 + 48 );
    y[1] = (char)( j % 10 + 48 );
    y[2] = '\0';
    if ( y[0] == '0' ) {
        y[0] = ' ';
    }

    return;
}

```

6.1.1.6 void get_label_y_right (int j, char y[])

Definition at line 212 of file board.c.

```

{

    j++;

    y[0] = (char)(int)( j / 10 + 48 );
    y[1] = (char)( j % 10 + 48 );
    y[2] = '\0';
    if ( y[0] == '0' ) {
        y[0] = y[1];
        y[1] = '\0';
    }

    return;
}

```

6.1.1.7 void init_board (unsigned short wanted_board_size)

Definition at line 32 of file board.c.

```

{

    int i, j;

    board_size = wanted_board_size;

    board = malloc( board_size * sizeof(int * ) );
    hoshi = malloc( board_size * sizeof(bool * ) );
    if ( board == NULL || hoshi == NULL ) {
        fprintf( stderr, "Failed to malloc memory");
        exit(EXIT_FAILURE);
    }

    for ( i = 0; i < board_size; i++ ) {
        board[i] = malloc( board_size * sizeof(int) );
        hoshi[i] = malloc( board_size * sizeof(bool) );
        if ( board[i] == NULL || hoshi[i] == NULL ) {
            fprintf( stderr, "Failed to malloc memory");
            exit(EXIT_FAILURE);
        }
    }
}

```

```

    }

    for ( j = 0; j < board_size; j++ ) {
        board[i][j] = EMPTY;
        hoshi[i][j] = false;
    }
}

switch (board_size) {
    case 19:
        hoshi[3][3] = true;
        hoshi[3][9] = true;
        hoshi[3][15] = true;
        hoshi[9][3] = true;
        hoshi[9][9] = true;
        hoshi[9][15] = true;
        hoshi[15][3] = true;
        hoshi[15][9] = true;
        hoshi[15][15] = true;
        break;
    case 13:
        hoshi[3][3] = true;
        hoshi[3][9] = true;
        hoshi[9][3] = true;
        hoshi[9][9] = true;
        hoshi[6][6] = true;
        break;
    case 9:
        hoshi[2][2] = true;
        hoshi[2][6] = true;
        hoshi[6][2] = true;
        hoshi[6][6] = true;
        hoshi[4][4] = true;
        break;
}

return;
}

```

6.1.1.8 bool is_hoshi (int *i*, int *j*)

Definition at line 227 of file board.c.

```

{

    return hoshi[i][j];
}

```

6.1.1.9 void set_vertex (int *color*, int *i*, int *j*)

Definition at line 237 of file board.c.

```

{

    board[i][j] = color;

    return;
}

```

6.1.2 Variable Documentation

6.1.2.1 `int** board`

Definition at line 9 of file board.c.

6.1.2.2 `int board_size = 0`

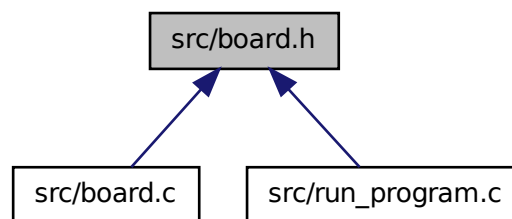
Definition at line 12 of file board.c.

6.1.2.3 `bool** hoshi`

Definition at line 10 of file board.c.

6.2 `src/board.h` File Reference

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



Functions

- void [init_board](#) (unsigned short wanted_board_size)
- void [free_board](#) (void)
- void [get_board_as_string](#) (char board_output[])
- int [get_board_size](#) (void)
- void [set_vertex](#) (int color, int i, int j)

6.2.1 Function Documentation

6.2.1.1 `void free_board (void)`

Definition at line 98 of file board.c.


```

    {
        int i;

        for ( i = 0; i < board_size; i++ ) {
            free(board[i]);
            free(hoshi[i]);
        }
        free(board);
        free(hoshi);

        return;
    }

```

6.2.1.2 void get_board_as_string (char board_output[])

Definition at line 120 of file board.c.

```

    {
        int i;          // Index for x-axis
        int j;          // Index for y-axis
        char x[3];      // Label for x-axis
        char y[2];      // Label for y-axis

        board_output[0] = '\0';
        strcat( board_output, "\n" );

        /* Print uppercase letters above the board */
        strcat( board_output, " " );
        for ( i = 0; i < board_size; i++ ) {
            get_label_x( i, x );
            strcat( board_output, " " );
            strcat( board_output, x );
        }
        strcat( board_output, "\n" );

        for ( j = board_size - 1; j >= 0; j-- ) {

            /* Print numbers left of board */
            get_label_y_left( j, y );
            strcat( board_output, " " );
            strcat( board_output, y );

            /* Print board fields */
            for ( i = 0; i < board_size; i++ ) {
                strcat( board_output, " " );
                switch ( board[i][j] ) {
                    case WHITE:
                        strcat( board_output, WHITE_STONE );
                        break;
                    case BLACK:
                        strcat( board_output, BLACK_STONE );
                        break;
                    case EMPTY:
                        switch ( is_hoshi( i, j ) ) {
                            case true:
                                strcat( board_output, FIELD_HOSHI );
                                break;
                            case false:
                                strcat( board_output, FIELD_EMPTY );
                                break;
                        }
                        break;
                }
            }
        }
    }

```

```

        /* Print numbers right of board */
        get_label_y_right( j, y );
        strcat( board_output, " " );
        strcat( board_output, y );
        strcat( board_output, "\n" );
    }

    /* Print uppercase letters below board */
    strcat( board_output, " " );
    for ( i = 0; i < board_size; i++ ) {
        get_label_x( i, x );
        strcat( board_output, " " );
        strcat( board_output, x );
    }

    return;
}

```

6.2.1.3 int get_board_size (void)

Definition at line 232 of file board.c.

```

        {

    return board_size;
}

```

6.2.1.4 void init_board (unsigned short wanted_board_size)

Definition at line 32 of file board.c.

```

                                                                    {

    int i, j;

    board_size = wanted_board_size;

    board = malloc( board_size * sizeof(int *) );
    hoshi = malloc( board_size * sizeof(bool *) );
    if ( board == NULL || hoshi == NULL ) {
        fprintf( stderr, "Failed to malloc memory");
        exit(EXIT_FAILURE);
    }

    for ( i = 0; i < board_size; i++ ) {
        board[i] = malloc( board_size * sizeof(int) );
        hoshi[i] = malloc( board_size * sizeof(bool) );
        if ( board[i] == NULL || hoshi[i] == NULL ) {
            fprintf( stderr, "Failed to malloc memory");
            exit(EXIT_FAILURE);
        }

        for ( j = 0; j < board_size; j++ ) {
            board[i][j] = EMPTY;
            hoshi[i][j] = false;
        }
    }

    switch (board_size) {
        case 19:
            hoshi[3][3] = true;

```

```
        hoshi[3][9]    = true;
        hoshi[3][15]   = true;
        hoshi[9][3]    = true;
        hoshi[9][9]    = true;
        hoshi[9][15]   = true;
        hoshi[15][3]   = true;
        hoshi[15][9]   = true;
        hoshi[15][15]  = true;
        break;
    case 13:
        hoshi[3][3]    = true;
        hoshi[3][9]    = true;
        hoshi[9][3]    = true;
        hoshi[9][9]    = true;
        hoshi[6][6]    = true;
        break;
    case 9:
        hoshi[2][2]    = true;
        hoshi[2][6]    = true;
        hoshi[6][2]    = true;
        hoshi[6][6]    = true;
        hoshi[4][4]    = true;
        break;
}

return;
}
```

6.2.1.5 void set_vertex (int *color*, int *i*, int *j*)

Definition at line 237 of file board.c.

```

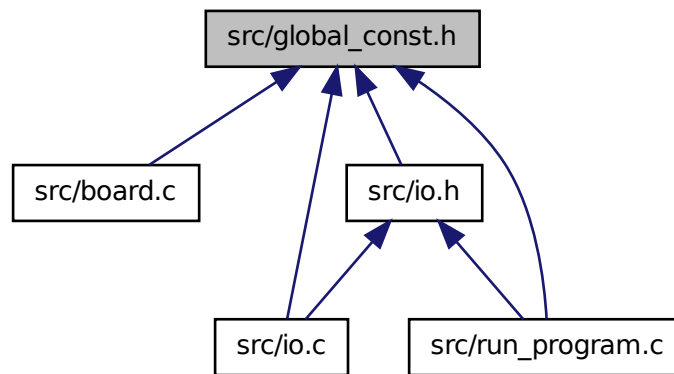
{

    board[i][j] = color;

    return;
}
```

6.3 src/global_const.h File Reference

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



Defines

- #define `MAX_TOKEN_LENGTH` 20
Defines the maximum length of any given GTP token, either command or argument.
- #define `MAX_TOKEN_COUNT` 10
Defines the maximum number of given GTP tokens for any command plus arguments.
- #define `MAX_OUTPUT_LENGTH` 2048
Sets the length of the GTP output buffer.
- #define `EMPTY` 0
Constant for an empty field.
- #define `BLACK` 1
Constant for black stone.
- #define `WHITE` -1
Constant for white stone.
- #define `FIELD_EMPTY` "."
Defines the character which is shown for an empty field.
- #define `FIELD_HOSHI` "+"
Defines the character which is shown for a star field.
- #define `WHITE_STONE` "0"

Defines the character which is shown for a white stone.

- `#define BLACK_STONE "X"`
Defines the character which is shown for a black stone.
- `#define BOARD_SIZE_MIN 2`
Defines the minimum board size which is accepted.
- `#define BOARD_SIZE_MAX 25`
Defines the maximum board size which is accepted.
- `#define BOARD_SIZE_DEF 19`
Defines the default board size.

6.3.1 Define Documentation

6.3.1.1 `#define BLACK 1`

Constant for black stone.

Definition at line 15 of file global_const.h.

6.3.1.2 `#define BLACK_STONE "X"`

Defines the character which is shown for a black stone.

Definition at line 26 of file global_const.h.

6.3.1.3 `#define BOARD_SIZE_DEF 19`

Defines the default board size.

Definition at line 33 of file global_const.h.

6.3.1.4 `#define BOARD_SIZE_MAX 25`

Defines the maximum board size which is accepted.

Definition at line 31 of file global_const.h.

6.3.1.5 `#define BOARD_SIZE_MIN 2`

Defines the minimum board size which is accepted.

Definition at line 29 of file global_const.h.

6.3.1.6 `#define EMPTY 0`

Constant for an empty field.

Definition at line 13 of file global_const.h.

6.3.1.7 #define FIELD_EMPTY "."

Defines the character which is shown for an empty field.

Definition at line 20 of file global_const.h.

6.3.1.8 #define FIELD_HOSHI "+"

Defines the character which is shown for a star field.

Definition at line 22 of file global_const.h.

6.3.1.9 #define MAX_OUTPUT_LENGTH 2048

Sets the length of the GTP output buffer.

Definition at line 10 of file global_const.h.

6.3.1.10 #define MAX_TOKEN_COUNT 10

Defines the maximum number of given GTP tokens for any command plus arguments.

Definition at line 7 of file global_const.h.

6.3.1.11 #define MAX_TOKEN_LENGTH 20

Defines the maximum length of any given GTP token, either command or argument.

Definition at line 5 of file global_const.h.

6.3.1.12 #define WHITE -1

Constant for white stone.

Definition at line 17 of file global_const.h.

6.3.1.13 #define WHITE_STONE "0"

Defines the character which is shown for a white stone.

Definition at line 24 of file global_const.h.

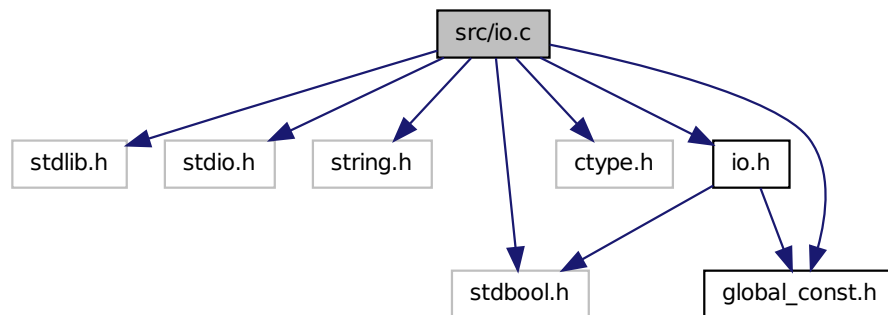
6.4 src/io.c File Reference

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

```
#include "global_const.h"
```

```
#include "io.h"
```

Include dependency graph for io.c:



Functions

- void [read_gtp_input](#) (struct [command](#) *command_data)
- void [set_output_error](#) (void)
- bool [get_output_error](#) (void)
- void [add_output](#) (const char to_output[])
- void [print_output](#) (int command_id)
- void [trim](#) (char *input)
- void [drop_comment](#) (char *input)
- bool [is_input_empty](#) (void)
- void [parse_gtp_input](#) (char *command_input_buffer, char tokens[][MAX_TOKEN_LENGTH])
- void [init_tokens](#) (char tokens[][MAX_TOKEN_LENGTH])
- void [identify_tokens](#) (char tokens[][MAX_TOKEN_LENGTH], struct [command](#) *command_data)

Variables

- char [command_input_buffer](#) [SIZE_INPUT_BUFFER]
- bool [input_empty](#) = false
- bool [output_error](#) = false
- char [output](#) [MAX_OUTPUT_LENGTH]

6.4.1 Function Documentation

6.4.1.1 void add_output (const char to_output[])

Definition at line 115 of file io.c.

```

{

    int new_output_length = strlen(output) + strlen(to_output) + 1;
    if ( new_output_length > MAX_OUTPUT_LENGTH ) {
        fprintf( stderr, "MAX_OUTPUT_LENGTH exceeded\n" );
        exit(EXIT_FAILURE);
    }

    strcat( output, to_output );
    strcat( output, "\n" );

    return;
}

```

6.4.1.2 void drop_comment (char * input)

Definition at line 232 of file io.c.

```

{

    int i = 0;
    char current_char = '\0';

    for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
        current_char = input[i];
        if ( current_char == '#' ) {
            input[i] = '\0';
            break;
        }
        if ( current_char == '\0' ) {
            break;
        }
    }

    return;
}

```

6.4.1.3 bool get_output_error (void)

Definition at line 100 of file io.c.

```

{

    return output_error;
}

```

6.4.1.4 void identify_tokens (char tokens[][MAX_TOKEN_LENGTH], struct command * command_data)

Definition at line 349 of file io.c.

```

{
    int id;
    int arg_start; // Index of first argument
    int i, j;

    // Check if first token is regular id,

```



```

// if not it must be the command name.
id = atoi( tokens[0] );
if ( id > 0 ) {
    command_data->id = id;
    strcpy( command_data->name, tokens[1] );
    arg_start = 2;
}
else if ( id < 0 ) {
    command_data->id = -1;
    strcpy( command_data->name, tokens[1] );
    arg_start = 2;
}
else {
    command_data->id = -1;
    strcpy( command_data->name, tokens[0] );
    arg_start = 1;
}

// Check for special case id 0:
if ( strcmp( tokens[0], "0" ) == 0 ) {
    command_data->id = 0;
    strcpy( command_data->name, tokens[1] );
    arg_start = 2;
}

// Copy arguments into command struct:
j = 0;
for ( i = arg_start; i < MAX_TOKEN_COUNT; i++ ) {
    if ( tokens[i][0] == '\0' ) {
        break;
    }
    strcpy( command_data->argv[j++], tokens[i] );
}
command_data->argc = j;

// DEBUG
/*
printf( "ID: %d\n", command_data->id );
printf( "NAME: %s\n", command_data->name );
printf( "ARGC: %d\n", command_data->argc );
for ( i = 0; i < j; i++ ) {
    printf( "ARG %d: %s\n", i, command_data->argv[i] );
}
*/

return;
}

```

6.4.1.5 void init_tokens (char tokens[][MAX_TOKEN_LENGTH])

Definition at line 328 of file io.c.

```

{
    int i;

    for ( i = 0; i < MAX_TOKEN_COUNT; i++ ) {
        tokens[i][0] = '\0';
    }

    return;
}

```

6.4.1.6 bool is_input_empty (void)

Definition at line 259 of file io.c.

```

        {

    return input_empty;
}

```

6.4.1.7 void parse_gtp_input (char * *command_input_buffer*, char *tokens*[[MAX_TOKEN_LENGTH]])

Definition at line 274 of file io.c.

```

    {
char current_char = '\0';
int i = 0; // Index of input buffer
int j = 0; // Counts number of tokens
int k = 0; // Index of each token

// Get tokens from input:
for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
    current_char = command_input_buffer[i];
    if ( ! isspace(current_char) && current_char != '\0' ) {
        if ( k < MAX_TOKEN_LENGTH ) {
            tokens[j][k] = current_char;
            k++;
        }
        else {
            set_output_error();
            add_output( "MAX_TOKEN_LENGTH exceeded" );
            init_tokens(tokens);
            return;
        }
    }
    else {
        tokens[j][k] = '\0';
        j++;
        k = 0;

        if ( j >= MAX_TOKEN_COUNT ) {
            set_output_error();
            add_output( "MAX_TOKEN_COUNT exceeded" );
            init_tokens(tokens);
            return;
        }

        // Set terminating argument:
        tokens[j][0] = '\0';
    }

    if ( current_char == '\0' ) {
        break;
    }
}

return;
}

```

6.4.1.8 void print_output (int *command_id*)

Definition at line 139 of file io.c.

```

{

    /*
    if ( input_empty == true ) {
        input_empty = false;
        return;
    }
    */

    if ( output_error == false ) {
        printf("=");
    }
    else {
        printf("?");
    }

    if ( command_id >= 0 ) {
        printf( "%d", command_id );
    }

    printf(" ");

    // If output is empty we fill it with an empty string to
    // get that additional newline:
    if ( strlen(output) == 0 ) {
        add_output("");
    }

    printf( "%s\n", output );

    strcpy( output, "" );
    //output_error = false;

    return;
}

```

6.4.1.9 void read_gtp_input (struct command * *command_data*)

Definition at line 28 of file io.c.

```

{

    int c = '\n';
    int i = 0;
    char tokens[MAX_TOKEN_COUNT][MAX_TOKEN_LENGTH];

    init_tokens(tokens);

    input_empty = false;
    output_error = false;

    do {
        c = getchar();
        command_input_buffer[i] = (char) c;
        i++;
    } while ( c != '\n' && i < SIZE_INPUT_BUFFER );

    // Overwrite last char with newline
    command_input_buffer[i-1] = '\0';
}

```

```

drop_comment(command_input_buffer);
trim(command_input_buffer);

if ( strlen(command_input_buffer) == 0 ) {
    input_empty = true;
    return;
}

parse_gtp_input( command_input_buffer, tokens );
identify_tokens( tokens, command_data );

/* Test output */
/*
i = 0;
while ( tokens[i][0] != '\0' ) {
    printf( "Argument: %s\n", tokens[i] );
    i++;
}
*/

//select_command();

//strcpy( command, command_input_buffer );

return;
}

```

6.4.1.10 void set_output_error (void)

Definition at line 84 of file io.c.

```

{

    output_error = true;

    return;
}

```

6.4.1.11 void trim (char * *input*)

Definition at line 186 of file io.c.

```

{
    char temp_input[SIZE_INPUT_BUFFER];
    char current_char = '\0';
    char last_char    = '\0';
    int i = 0;
    int j = 0;

    for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
        current_char = input[i];

        /* Skip leading whitespace */
        if ( isspace(current_char) && j == 0 ) {
            continue;
        }

        /* Write only one whitespace */
        if ( isspace(last_char) && !isspace(current_char) && current_char != '\0' ) {
            temp_input[j] = ' ';

```

```
        j++;
    }

    /* Write non-whitespace characters */
    if ( ! isspace(current_char) ) {
        temp_input[j] = current_char;
        j++;
    }

    last_char = current_char;
}

temp_input[j] = '\0';

strncpy( input, temp_input, SIZE_INPUT_BUFFER );

return;
}
```

6.4.2 Variable Documentation

6.4.2.1 char command_input_buffer[SIZE_INPUT_BUFFER]

Definition at line 10 of file io.c.

6.4.2.2 bool input_empty = false

Definition at line 12 of file io.c.

6.4.2.3 char output[MAX_OUTPUT_LENGTH]

Definition at line 14 of file io.c.

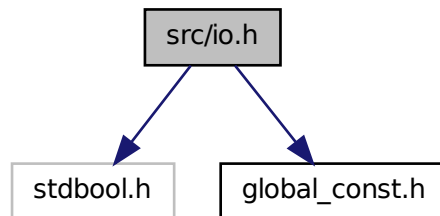
6.4.2.4 bool output_error = false

Definition at line 13 of file io.c.

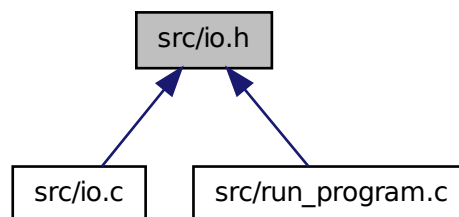
6.5 src/io.h File Reference

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

Include dependency graph for io.h:



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



Data Structures

- struct `command`

Defines

- #define `SIZE_INPUT_BUFFER` 256

Functions

- void `read_gtp_input` (struct `command` *command_data)
- void `add_output` (const char to_output[])
- void `set_output_error` (void)
- bool `get_output_error` (void)
- void `print_output` (int command_id)
- void `trim` (char *input)

- void [drop_comment](#) (char *input)
- void [parse_gtp_input](#) (char *command_input_buffer, char command[][MAX_TOKEN_LENGTH])
- void [init_tokens](#) (char tokens[][MAX_TOKEN_LENGTH])
- void [identify_tokens](#) (char tokens[][MAX_TOKEN_LENGTH], struct [command](#) *command_data)
- bool [is_input_empty](#) (void)

6.5.1 Define Documentation

6.5.1.1 #define SIZE_INPUT_BUFFER 256

Definition at line 7 of file io.h.

6.5.2 Function Documentation

6.5.2.1 void add_output (const char to_output[])

Definition at line 115 of file io.c.

```

{
    int new_output_length = strlen(output) + strlen(to_output) + 1;
    if ( new_output_length > MAX_OUTPUT_LENGTH ) {
        fprintf( stderr, "MAX_OUTPUT_LENGTH exceeded\n" );
        exit(EXIT_FAILURE);
    }

    strcat( output, to_output );
    strcat( output, "\n" );

    return;
}

```

6.5.2.2 void drop_comment (char * input)

Definition at line 232 of file io.c.

```

{
    int i = 0;
    char current_char = '\0';

    for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
        current_char = input[i];
        if ( current_char == '#' ) {
            input[i] = '\0';
            break;
        }
        if ( current_char == '\0' ) {
            break;
        }
    }

    return;
}

```

6.5.2.3 bool get_output_error (void)

Definition at line 100 of file io.c.

```

    {
        return output_error;
    }

```

6.5.2.4 void identify_tokens (char tokens[][MAX_TOKEN_LENGTH], struct command * command_data)

Definition at line 349 of file io.c.

```

    {
        int id;
        int arg_start; // Index of first argument
        int i, j;

        // Check if first token is regular id,
        // if not it must be the command name.
        id = atoi( tokens[0] );
        if ( id > 0 ) {
            command_data->id = id;
            strcpy( command_data->name, tokens[1] );
            arg_start = 2;
        }
        else if ( id < 0 ) {
            command_data->id = -1;
            strcpy( command_data->name, tokens[1] );
            arg_start = 2;
        }
        else {
            command_data->id = -1;
            strcpy( command_data->name, tokens[0] );
            arg_start = 1;
        }

        // Check for special case id 0:
        if ( strcmp( tokens[0], "0" ) == 0 ) {
            command_data->id = 0;
            strcpy( command_data->name, tokens[1] );
            arg_start = 2;
        }

        // Copy arguments into command struct:
        j = 0;
        for ( i = arg_start; i < MAX_TOKEN_COUNT; i++ ) {
            if ( tokens[i][0] == '\0' ) {
                break;
            }
            strcpy( command_data->argv[j++], tokens[i] );
        }
        command_data->argc = j;

        // DEBUG
        /*
        printf( "ID: %d\n", command_data->id );
        printf( "NAME: %s\n", command_data->name );
        printf( "ARGC: %d\n", command_data->argc );
        for ( i = 0; i < j; i++ ) {
            printf( "ARG %d: %s\n", i, command_data->argv[i] );

```



```

    }
    */

    return;
}

```

6.5.2.5 void init_tokens (char tokens[][MAX_TOKEN_LENGTH])

Definition at line 328 of file io.c.

```

{

    int i;

    for ( i = 0; i < MAX_TOKEN_COUNT; i++ ) {
        tokens[i][0] = '\0';
    }

    return;
}

```

6.5.2.6 bool is_input_empty (void)

Definition at line 259 of file io.c.

```

{

    return input_empty;
}

```

6.5.2.7 void parse_gtp_input (char * command_input_buffer, char command[][MAX_TOKEN_LENGTH])

Definition at line 274 of file io.c.

```

{
    char current_char = '\0';
    int i = 0; // Index of input buffer
    int j = 0; // Counts number of tokens
    int k = 0; // Index of each token

    // Get tokens from input:
    for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
        current_char = command_input_buffer[i];
        if ( ! isspace(current_char) && current_char != '\0' ) {
            if ( k < MAX_TOKEN_LENGTH ) {
                tokens[j][k] = current_char;
                k++;
            }
            else {
                set_output_error();
                add_output( "MAX_TOKEN_LENGTH exceeded" );
                init_tokens(tokens);
                return;
            }
        }
    }
    else {

```

```

        tokens[j][k] = '\0';
        j++;
        k = 0;

        if ( j >= MAX_TOKEN_COUNT ) {
            set_output_error();
            add_output( "MAX_TOKEN_COUNT exceeded" );
            init_tokens(tokens);
            return;
        }

        // Set terminating argument:
        tokens[j][0] = '\0';
    }

    if ( current_char == '\0' ) {
        break;
    }
}

return;
}

```

6.5.2.8 void print_output (int *command_id*)

Definition at line 139 of file io.c.

```

{

    /*
    if ( input_empty == true ) {
        input_empty = false;
        return;
    }
    */

    if ( output_error == false ) {
        printf("=");
    }
    else {
        printf("?");
    }

    if ( command_id >= 0 ) {
        printf( "%d", command_id );
    }

    printf(" ");

    // If output is empty we fill it with an empty string to
    // get that additional newline:
    if ( strlen(output) == 0 ) {
        add_output("");
    }

    printf( "%s\n", output );

    strcpy( output, "" );
    //output_error = false;

    return;
}

```

6.5.2.9 void read_gtp_input (struct command * *command_data*)

Definition at line 28 of file io.c.

```

{
    int c = '\n';
    int i = 0;
    char tokens[MAX_TOKEN_COUNT][MAX_TOKEN_LENGTH];

    init_tokens(tokens);

    input_empty = false;
    output_error = false;

    do {
        c = getchar();
        command_input_buffer[i] = (char) c;
        i++;
    } while ( c != '\n' && i < SIZE_INPUT_BUFFER );

    // Overwrite last char with newline
    command_input_buffer[i-1] = '\0';

    drop_comment(command_input_buffer);
    trim(command_input_buffer);

    if ( strlen(command_input_buffer) == 0 ) {
        input_empty = true;
        return;
    }

    parse_gtp_input( command_input_buffer, tokens );
    identify_tokens( tokens, command_data );

    /* Test output */
    /*
    i = 0;
    while ( tokens[i][0] != '\0' ) {
        printf( "Argument: %s\n", tokens[i] );
        i++;
    }
    */

    //select_command();

    //strcpy( command, command_input_buffer );

    return;
}

```

6.5.2.10 void set_output_error (void)

Definition at line 84 of file io.c.

```

{

    output_error = true;

    return;
}

```

6.5.2.11 void trim (char * *input*)

Definition at line 186 of file io.c.

```
    {
char temp_input[SIZE_INPUT_BUFFER];
char current_char = '\0';
char last_char    = '\0';
int i = 0;
int j = 0;

for ( i = 0; i < SIZE_INPUT_BUFFER; i++ ) {
    current_char = input[i];

    /* Skip leading whitespace */
    if ( isspace(current_char) && j == 0 ) {
        continue;
    }

    /* Write only one whitespace */
    if ( isspace(last_char) && !isspace(current_char) && current_char != '\0' ) {
        temp_input[j] = ' ';
        j++;
    }

    /* Write non-whitespace characters */
    if ( ! isspace(current_char) ) {
        temp_input[j] = current_char;
        j++;
    }

    last_char = current_char;
}

temp_input[j] = '\0';

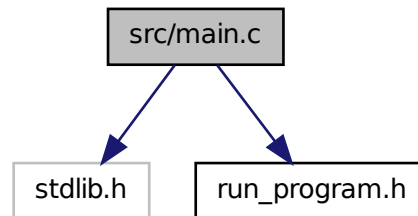
strncpy( input, temp_input, SIZE_INPUT_BUFFER );

return;
}
```

6.6 src/main.c File Reference

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

Include dependency graph for main.c:



Functions

- `int main (int argc, char **argv)`

The `main()` function is only a wrapper for the `run()` function.

6.6.1 Function Documentation

6.6.1.1 `int main (int argc, char ** argv)`

The `main()` function is only a wrapper for the `run()` function.

The `main()` function only calls `run()`. This is because `main()` itself cannot be unit-tested with check. Therefore the real work is done by `run()`.

Parameters

argc Number of command line arguments

argv Array of all command line arguments

Returns

EXIT_SUCCESS | EXIT_FAILURE

See also

info check

Definition at line 17 of file main.c.

```
{
    int exit_value = EXIT_FAILURE;

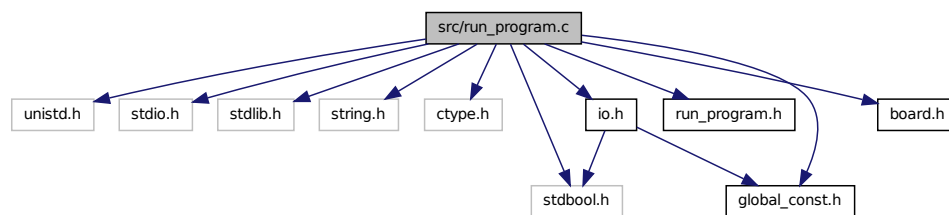
    exit_value = run( argc, argv );

    return exit_value;
}
```

6.7 src/run_program.c File Reference

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include "global_const.h"
#include "run_program.h"
#include "io.h"
#include "board.h"
```

Include dependency graph for run_program.c:



Data Structures

- struct [command_func](#)

Functions

- void [init_known_commands](#) (void)
- void [read_opts](#) (int argc, char **argv)
- void [select_command](#) (struct [command](#) *command_data)
- void [print_help_message](#) (void)
- void [print_version](#) (void)
- void [set_quit_program](#) (void)
 - Sets control variable for working loop.*
- void [str_toupper](#) (char string[])
- void [gtp_protocol_version](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
 - Shows the used GTP version number.*
- void [gtp_name](#) (int argc, char argv[][MAX_TOKEN_LENGTH])
 - Shows the program's name.*

- void `gtp_version` (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows the program's version number.
- void `gtp_known_command` (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows whether a given command is known or not.
- void `gtp_list_commands` (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows a list of all know GTP commands.
- void `gtp_quit` (int argc, char argv[][MAX_TOKEN_LENGTH])
Quits the program.
- void `gtp_boardsize` (int argc, char argv[][MAX_TOKEN_LENGTH])
Changes the current board size.
- void `gtp_clear_board` (int argc, char argv[][MAX_TOKEN_LENGTH])
Clears the board.
- void `gtp_komi` (int argc, char argv[][MAX_TOKEN_LENGTH])
Sets komi.
- void `gtp_play` (int argc, char argv[][MAX_TOKEN_LENGTH])
Description missing!
- void `gtp_showboard` (int argc, char argv[][MAX_TOKEN_LENGTH])
Shows a simple ASCII board.
- int `run` (int argc, char **argv)
Substitute for `main()` function, because `main()` itself cannot be unit-tested with check.

Variables

- const char `PROGRAM_NAME` [] = "haigo"
- const char `PROGRAM_VERSION` [] = "0.1"
- const char `GTP_VERSION` [] = "2"
- const char `help_message` []
- int `quit_program` = 0
If set to 1 the main control loop exits.
- struct `command_func known_commands` [COUNT_KNOWN_COMMANDS]
- float `komi` = 0.0

6.7.1 Function Documentation

6.7.1.1 void init_known_commands (void)

Definition at line 210 of file run_program.c.

```

        {

known_commands[0].command = "protocol_version";
known_commands[0].function = (*gtp_protocol_version);
known_commands[1].command = "name";
known_commands[1].function = (*gtp_name);
known_commands[2].command = "version";
known_commands[2].function = (*gtp_version);
known_commands[3].command = "known_command";
known_commands[3].function = (*gtp_known_command);
known_commands[4].command = "list_commands";
known_commands[4].function = (*gtp_list_commands);
known_commands[5].command = "quit";
known_commands[5].function = (*gtp_quit);
known_commands[6].command = "boardsize";
known_commands[6].function = (*gtp_boardsize);
known_commands[7].command = "clear_board";
known_commands[7].function = (*gtp_clear_board);
known_commands[8].command = "komi";
known_commands[8].function = (*gtp_komi);
known_commands[9].command = "play";
known_commands[9].function = (*gtp_play);
known_commands[10].command = "showboard";
known_commands[10].function = (*gtp_showboard);

return;
}

```

6.7.1.2 void print_help_message (void)

Definition at line 176 of file run_program.c.

```

        {

printf( "%s", help_message );

return;
}

```

6.7.1.3 void print_version (void)

Definition at line 193 of file run_program.c.

```

        {

printf( "%s %s\n", PROGRAM_NAME, PROGRAM_VERSION );

return;
}

```

6.7.1.4 void read_opts (int argc, char ** argv)

Definition at line 144 of file run_program.c.

```

int opt;

        {

```



```
while ( ( opt = getopt( argc, argv, VALID_OPTIONS ) ) != -1 ) {
    switch (opt) {
        case 'h':
            print_help_message();
            set_quit_program();
            break;
        case 'v':
            print_version();
            set_quit_program();
            break;
        default:
            exit(EXIT_FAILURE);
            break;
    }
}

return;
}
```

6.7.1.5 int run (int argc, char ** argv)

Substitute for [main\(\)](#) function, because [main\(\)](#) itself cannot be unit-tested with check.

The [run\(\)](#) function performs the following tasks:

1. Initialization
2. STDOUT buffer size set to NULL
3. Checking command line arguments (like -h, -v, etc.)
4. Starting the working loop

Parameters

argc Number of command line arguments (same as for [main\(\)](#)).

argv Array of all command line arguments (same as for [main\(\)](#)).

Returns

EXIT_SUCCESS | EXIT_FAILURE

See also

[info check](#)

Definition at line 79 of file run_program.c.

```
{
    struct command command_data;

    // Initialization
    init_board(BOARD_SIZE_DEF);
    init_known_commands();

    // STDOUT must be unbuffered:
    setbuf( stdout, NULL );

    // Read command line arguments:
    read_opts( argc, argv );
```

```

// Working loop:
while ( quit_program == 0 ) {
    read_gtp_input(&command_data);

    // Ignore empty lines:
    if ( is_input_empty() == true ) {
        continue;
    }

    if ( get_output_error() == false ) {
        select_command(&command_data);
    }

    print_output(command_data.id);
}

free_board();

return EXIT_SUCCESS;
}

```

6.7.1.6 void select_command (struct command * *command_data*)

Definition at line 248 of file run_program.c.

```

{
    int i;
    bool is_command = false;

    for ( i = 0; i < COUNT_KNOWN_COMMANDS; i++ ) {
        if ( strcmp( known_commands[i].command, command_data->name ) == 0 ) {
            is_command = true;
            known_commands[i].function( command_data->argc, command_data->argv );

            break;
        }
    }

    if ( is_command == false ) {
        set_output_error();
        add_output("unknown command");
    }

    return;
}

```

6.7.1.7 void set_quit_program (void)

Sets control variable for working loop.

The [set_quit_program\(\)](#) function sets the variable quit_program to 1. When this variable is 1, the control loop stops and the program exits.

Parameters

none

Returns

nothing

See also

[n/a]

Definition at line 126 of file run_program.c.

```
        {  
  
        quit_program = 1;  
  
        return;  
    }
```

6.7.1.8 void str_toupper (char *string*[])

Definition at line 277 of file run_program.c.

```
        {  
  
        int i;  
        int str_length = strlen(string);  
  
        for ( i = 0; i < str_length; i++ ) {  
            string[i] = toupper( string[i] );  
        }  
  
        return;  
    }
```

6.7.2 Variable Documentation**6.7.2.1 const char GTP_VERSION[] = "2"**

Definition at line 17 of file run_program.c.

6.7.2.2 const char help_message[]**Initial value:**

```
"This is a placeholder for the help message.\n\  
This message is shown when the program is called\n\  
with the command line argument -h.\n"
```

Definition at line 19 of file run_program.c.

6.7.2.3 struct command_func known_commands[COUNT_KNOWN_COMMANDS]

Definition at line 30 of file run_program.c.

6.7.2.4 float komi = 0.0

Definition at line 32 of file run_program.c.

6.7.2.5 `const char PROGRAM_NAME[] = "haigo"`

Definition at line 13 of file `run_program.c`.

6.7.2.6 `const char PROGRAM_VERSION[] = "0.1"`

Definition at line 14 of file `run_program.c`.

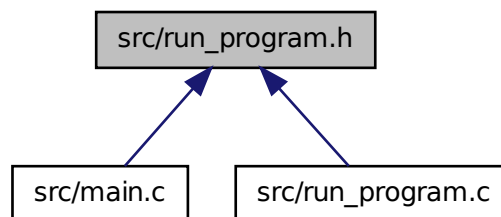
6.7.2.7 `int quit_program = 0`

If set to 1 the main control loop exits.

Definition at line 24 of file `run_program.c`.

6.8 `src/run_program.h` File Reference

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



Defines

- `#define COUNT_KNOWN_COMMANDS 11`

Defines the number of known GTP commands.

Functions

- `int run(int argc, char **argv)`

Substitute for `main()` function, because `main()` itself cannot be unit-tested with check.

6.8.1 Define Documentation

6.8.1.1 `#define COUNT_KNOWN_COMMANDS 11`

Defines the number of known GTP commands.

Definition at line 5 of file run_program.h.

6.8.2 Function Documentation

6.8.2.1 `int run (int argc, char ** argv)`

Substitute for `main()` function, because `main()` itself cannot be unit-tested with check.

The `run()` function performs the following tasks:

1. Initialization
2. STDOUT buffer size set to NULL
3. Checking command line arguments (like -h, -v, etc.)
4. Starting the working loop

Parameters

argc Number of command line arguments (same as for `main()`).

argv Array of all command line arguments (same as for `main()`).

Returns

EXIT_SUCCESS | EXIT_FAILURE

See also

info check

Definition at line 79 of file run_program.c.

```
    {  
        struct command command_data;  
  
        // Initialization  
        init_board(BOARD_SIZE_DEF);  
        init_known_commands();  
  
        // STDOUT must be unbuffered:  
        setbuf( stdout, NULL );  
  
        // Read command line arguments:  
        read_opts( argc, argv );  
  
        // Working loop:  
        while ( quit_program == 0 ) {  
            read_gtp_input(&command_data);  
  
            // Ignore empty lines:  
            if ( is_input_empty() == true ) {  
                continue;  
            }  
        }  
    }
```

```
    }

    if ( get_output_error() == false ) {
        select_command(&command_data);
    }

    print_output(command_data.id);
}

free_board();

return EXIT_SUCCESS;
}
```

Index

- add_output
 - io.c, [31](#)
 - io.h, [39](#)
- argc
 - command, [17](#)
- argv
 - command, [17](#)
- BLACK
 - global_const.h, [29](#)
- BLACK_STONE
 - global_const.h, [29](#)
- board
 - board.c, [24](#)
- board.c
 - board, [24](#)
 - board_size, [24](#)
 - free_board, [20](#)
 - get_board_as_string, [20](#)
 - get_board_size, [21](#)
 - get_label_x, [21](#)
 - get_label_y_left, [21](#)
 - get_label_y_right, [22](#)
 - hoshi, [24](#)
 - init_board, [22](#)
 - is_hoshi, [23](#)
 - set_vertex, [23](#)
- board.h
 - free_board, [24](#)
 - get_board_as_string, [25](#)
 - get_board_size, [26](#)
 - init_board, [26](#)
 - set_vertex, [27](#)
- board_size
 - board.c, [24](#)
- BOARD_SIZE_DEF
 - global_const.h, [29](#)
- BOARD_SIZE_MAX
 - global_const.h, [29](#)
- BOARD_SIZE_MIN
 - global_const.h, [29](#)
- command, [17](#)
 - argc, [17](#)
 - argv, [17](#)
 - command_func, [18](#)
 - id, [17](#)
 - name, [17](#)
- command_func, [18](#)
 - command, [18](#)
 - function, [18](#)
- command_input_buffer
 - io.c, [37](#)
- COUNT_KNOWN_COMMANDS
 - run_program.h, [53](#)
- drop_comment
 - io.c, [32](#)
 - io.h, [39](#)
- EMPTY
 - global_const.h, [29](#)
- FIELD_EMPTY
 - global_const.h, [29](#)
- FIELD_HOSHI
 - global_const.h, [30](#)
- free_board
 - board.c, [20](#)
 - board.h, [24](#)
- function
 - command_func, [18](#)
- get_board_as_string
 - board.c, [20](#)
 - board.h, [25](#)
- get_board_size
 - board.c, [21](#)
 - board.h, [26](#)
- get_label_x
 - board.c, [21](#)
- get_label_y_left
 - board.c, [21](#)
- get_label_y_right
 - board.c, [22](#)
- get_output_error
 - io.c, [32](#)
 - io.h, [39](#)
- global_const.h
 - BLACK, [29](#)
 - BLACK_STONE, [29](#)

- BOARD_SIZE_DEF, 29
- BOARD_SIZE_MAX, 29
- BOARD_SIZE_MIN, 29
- EMPTY, 29
- FIELD_EMPTY, 29
- FIELD_HOSHI, 30
- MAX_OUTPUT_LENGTH, 30
- MAX_TOKEN_COUNT, 30
- MAX_TOKEN_LENGTH, 30
- WHITE, 30
- WHITE_STONE, 30
- Go Text Protocol Administrative Commands, 8
- Go Text Protocol Commands, 7
- Go Text Protocol Core Play Commands, 14
- Go Text Protocol Debug Commands, 16
- Go Text Protocol Setup Commands, 12
- GTP_Administrative_Commands
 - gtp_known_command, 8
 - gtp_list_commands, 9
 - gtp_name, 10
 - gtp_protocol_version, 10
 - gtp_quit, 11
 - gtp_version, 11
- gtp_boardsize
 - GTP_Setup_Commands, 12
- gtp_clear_board
 - GTP_Setup_Commands, 13
- GTP_Core_Play_Commands
 - gtp_play, 14
- GTP_Debug_Commands
 - gtp_showboard, 16
- gtp_known_command
 - GTP_Administrative_Commands, 8
- gtp_komi
 - GTP_Setup_Commands, 13
- gtp_list_commands
 - GTP_Administrative_Commands, 9
- gtp_name
 - GTP_Administrative_Commands, 10
- gtp_play
 - GTP_Core_Play_Commands, 14
- gtp_protocol_version
 - GTP_Administrative_Commands, 10
- gtp_quit
 - GTP_Administrative_Commands, 11
- GTP_Setup_Commands
 - gtp_boardsize, 12
 - gtp_clear_board, 13
 - gtp_komi, 13
- gtp_showboard
 - GTP_Debug_Commands, 16
- GTP_VERSION
 - run_program.c, 51
- gtp_version
 - GTP_Administrative_Commands, 11
- help_message
 - run_program.c, 51
- hoshi
 - board.c, 24
- id
 - command, 17
- identify_tokens
 - io.c, 32
 - io.h, 40
- init_board
 - board.c, 22
 - board.h, 26
- init_known_commands
 - run_program.c, 47
- init_tokens
 - io.c, 33
 - io.h, 41
- input_empty
 - io.c, 37
- io.c
 - add_output, 31
 - command_input_buffer, 37
 - drop_comment, 32
 - get_output_error, 32
 - identify_tokens, 32
 - init_tokens, 33
 - input_empty, 37
 - is_input_empty, 33
 - output, 37
 - output_error, 37
 - parse_gtp_input, 34
 - print_output, 34
 - read_gtp_input, 35
 - set_output_error, 36
 - trim, 36
- io.h
 - add_output, 39
 - drop_comment, 39
 - get_output_error, 39
 - identify_tokens, 40
 - init_tokens, 41
 - is_input_empty, 41
 - parse_gtp_input, 41
 - print_output, 42
 - read_gtp_input, 42
 - set_output_error, 43
 - SIZE_INPUT_BUFFER, 39
 - trim, 43
- is_hoshi
 - board.c, 23
- is_input_empty

- io.c, [33](#)
- io.h, [41](#)
- known_commands
 - run_program.c, [51](#)
- komi
 - run_program.c, [51](#)
- main
 - main.c, [45](#)
- main.c
 - main, [45](#)
- MAX_OUTPUT_LENGTH
 - global_const.h, [30](#)
- MAX_TOKEN_COUNT
 - global_const.h, [30](#)
- MAX_TOKEN_LENGTH
 - global_const.h, [30](#)
- name
 - command, [17](#)
- output
 - io.c, [37](#)
- output_error
 - io.c, [37](#)
- parse_gtp_input
 - io.c, [34](#)
 - io.h, [41](#)
- print_help_message
 - run_program.c, [48](#)
- print_output
 - io.c, [34](#)
 - io.h, [42](#)
- print_version
 - run_program.c, [48](#)
- PROGRAM_NAME
 - run_program.c, [51](#)
- PROGRAM_VERSION
 - run_program.c, [52](#)
- quit_program
 - run_program.c, [52](#)
- read_gtp_input
 - io.c, [35](#)
 - io.h, [42](#)
- read_opts
 - run_program.c, [48](#)
- run
 - run_program.c, [49](#)
 - run_program.h, [53](#)
- run_program.c
 - GTP_VERSION, [51](#)
 - help_message, [51](#)
 - init_known_commands, [47](#)
 - known_commands, [51](#)
 - komi, [51](#)
 - print_help_message, [48](#)
 - print_version, [48](#)
 - PROGRAM_NAME, [51](#)
 - PROGRAM_VERSION, [52](#)
 - quit_program, [52](#)
 - read_opts, [48](#)
 - run, [49](#)
 - select_command, [50](#)
 - set_quit_program, [50](#)
 - str_toupper, [51](#)
- run_program.h
 - COUNT_KNOWN_COMMANDS, [53](#)
 - run, [53](#)
- select_command
 - run_program.c, [50](#)
- set_output_error
 - io.c, [36](#)
 - io.h, [43](#)
- set_quit_program
 - run_program.c, [50](#)
- set_vertex
 - board.c, [23](#)
 - board.h, [27](#)
- SIZE_INPUT_BUFFER
 - io.h, [39](#)
- src/board.c, [19](#)
- src/board.h, [24](#)
- src/global_const.h, [28](#)
- src/io.c, [30](#)
- src/io.h, [37](#)
- src/main.c, [44](#)
- src/run_program.c, [46](#)
- src/run_program.h, [52](#)
- str_toupper
 - run_program.c, [51](#)
- trim
 - io.c, [36](#)
 - io.h, [43](#)
- WHITE
 - global_const.h, [30](#)
- WHITE_STONE
 - global_const.h, [30](#)