haigo

0.1

Generated by Doxygen 1.7.1

Tue Jan 17 2012 16:51:52

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

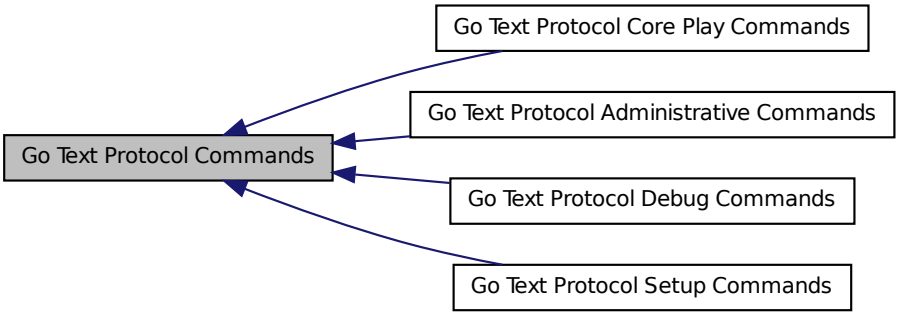
Here is a list of all files with brief descriptions:

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

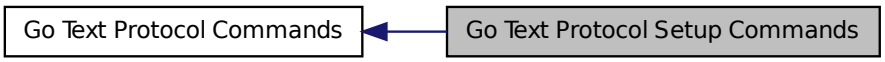
Go Text Protokol version 2, 6.3.1 Administrative Commands

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

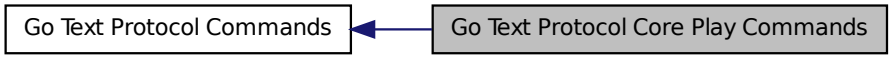
Go Text Protokol version 2, 6.3.2 Setup Commands

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

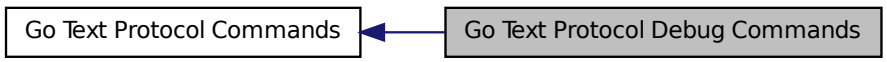
Go Text Protokol version 2, 6.3.3 Core Play Commands

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

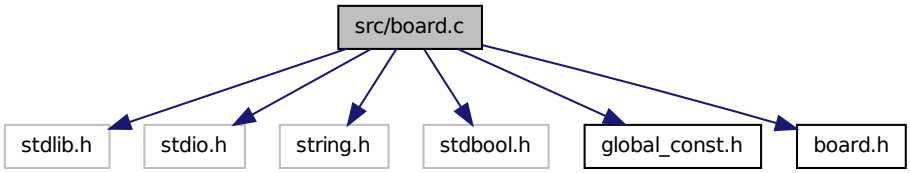
Chapter 6

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

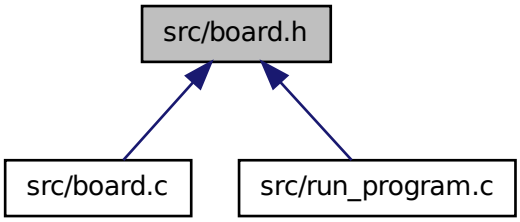
Definition at line 12 of file board.c.

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;
}
```

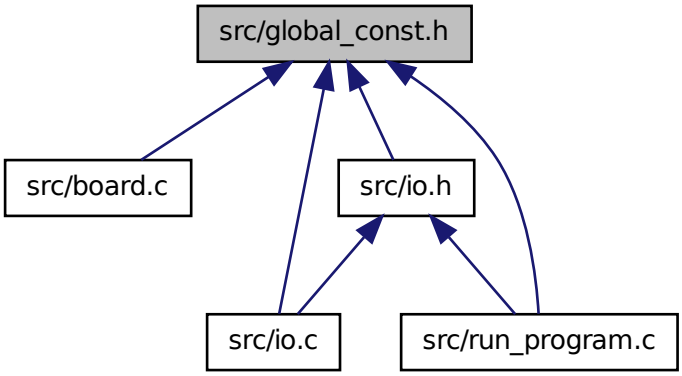
6.2.1.5 void set_vertex (int *color*, int *i*, int *j*)

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.

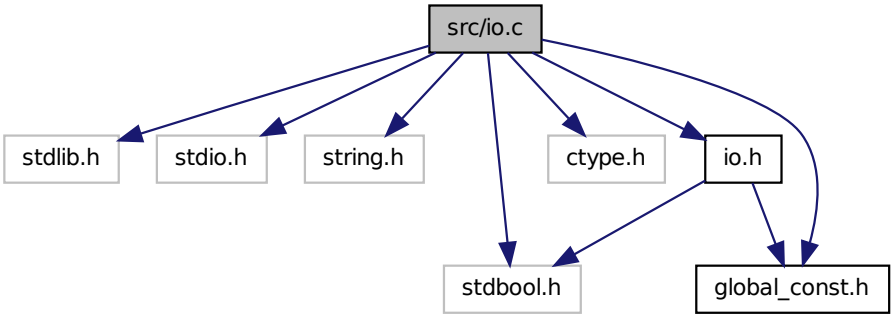
Definition at line 24 of file global_const.h.

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.

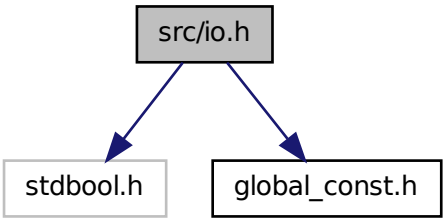
6.4.2.4 bool output_error = false

Definition at line 13 of file io.c.

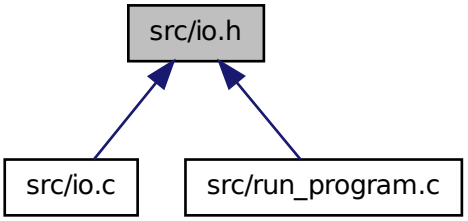
6.5 src/io.h File Reference

```
#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*)

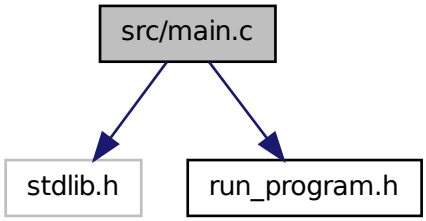
Definition at line 186 of file io.c.

```
                        {
    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

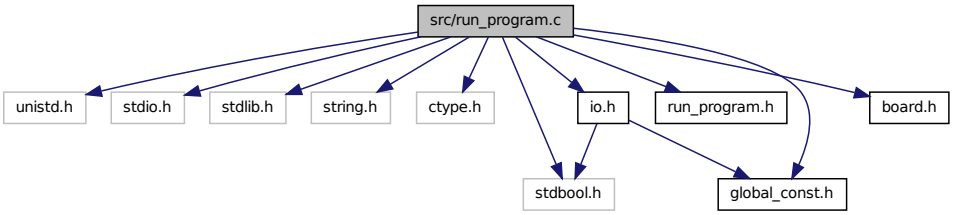
Definition at line 17 of file main.c.

```
                                    {
    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.

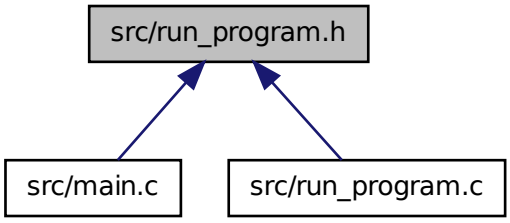
6.7.2.7 int quit_program = 0

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