# Source to Flow

0.2

Generated by Doxygen 1.9.8

# Chapter 1

# Specification

Programozás 1 - Nagy Házi Specifikáció - Szihalmi Botond L1U7KJ

## 1.1 Source to Flow specifikáció

Én egy saját ötlet alapján kezdtem el dolgozni a nagy házimon. \ A célja hogy egy beolvasott c source fileból egy megjeleníthető folyamat ábrát hozzon létre.\ A programot mind parancssorból mind grafikus felülettel lehet irányítani.

### 1.1.1 Parancssori irányítás

Itt nem tényleges irányítás történik, csak adott lehetőségek vannak a meghívás alatt:

- help
- theme
- output file
- input file

Ha semmilyen meghívási paraméter nincs megadva csak megnyitja a program grafikus felületét. \ Ha meg van adva a bemeneti file, azt a file-ot nyitja meg a grafikus felületen \ Ha bemeneti és kimeneti file is meg van adva, rögtön kimenti a folyamat ábra képét. \ A téma paraméter ezeknek a működését nem érinti. \ A help paraméter csak kiírja hogyan kell a parancssori irányítást használni.

### 1.1.2 Grafikus Irányítás

`Felső menü\ Folyamat ábra`

Egér irányítás:

- görgővel lehet a folyamat ábrán belül nagyítani, kisebbíteni.
- lenyomva tartva lehet vele mozogni jobbra, balra, fel, le.
- bal kattintással lehet mozgatni a folyamat ábra pontjait.

### 1.1.3 Témák

Saját témát lehet megadni `.ini` file-ként. \ Mindegyik típusú objektumhoz (beleértve a fő képernyőt is) külön témát kell megadni. \ Egy objektumhoz két színérték tartozik:

- háttér
- szöveg

### 1.1.4 File mentés

A létrehozott folyamati ábrát vagy `.png` vagy `.jpg`-ként lehet elmenteni. \ A mentett file nevét és helyét a felhasználó adja meg. \ A mentett file a használt téma alapján legyen színezve.

### 1.1.5 Kilépés

Kilépésnél rákérdezünk a felhasználóra hogy biztos meg szeretné e tenni, de automatikusan nem mentünk semmit.

# 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

# Data Structure Documentation

## 4.1 colour_t Struct Reference

colouring struct for theme

`#include <ini_reader.h>`

### Data Fields

- SDL_Colour background
- SDL_Colour text

### 4.1.1 Detailed Description

colouring struct for theme

Definition at line 34 of file ini_reader.h.

### 4.1.2 Field Documentation

#### 4.1.2.1 background

`SDL_Colour background`

Definition at line 35 of file ini_reader.h.

#### 4.1.2.2 text

`SDL_Colour text`

Definition at line 36 of file ini_reader.h.

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

- ini_reader.h

## 4.2 conditional_type_t Struct Reference

`#include <source_reader.h>`

### Data Fields

- char * condition

### 4.2.1 Detailed Description

Definition at line 26 of file source_reader.h.

### 4.2.2 Field Documentation

#### 4.2.2.1 condition

`char* condition`

Definition at line 27 of file source_reader.h.

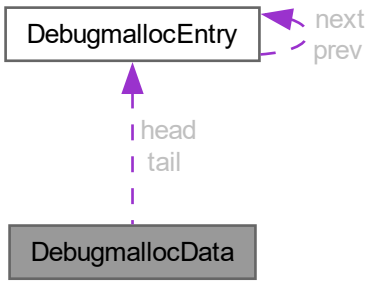
The documentation for this struct was generated from the following file:

- source_reader.h

## 4.3 DebugmallocData Struct Reference

`#include <debugmalloc.h>`

Collaboration diagram for DebugmallocData:



**Data Fields**

- char logfile [256]
- long max_block_size
- long alloc_count
- long long alloc_bytes
- long all_alloc_count
- long long all_alloc_bytes
- DebugmallocEntry head [debugmalloc_tablesize]
- DebugmallocEntry tail [debugmalloc_tablesize]

## 4.3.1 Detailed Description

Definition at line 64 of file debugmalloc.h.

## 4.3.2 Field Documentation

### 4.3.2.1 all_alloc_bytes

```
long long all_alloc_bytes
```

Definition at line 70 of file debugmalloc.h.

### 4.3.2.2 all_alloc_count

```
long all_alloc_count
```

Definition at line 69 of file debugmalloc.h.

### 4.3.2.3 alloc_bytes

```
long long alloc_bytes
```

Definition at line 68 of file debugmalloc.h.

### 4.3.2.4 alloc_count

```
long alloc_count
```

Definition at line 67 of file debugmalloc.h.

### 4.3.2.5 head

```
DebugmallocEntry head[debugmalloc_tablesize]
```

Definition at line 71 of file debugmalloc.h.

#### 4.3.2.6 logfile

```
char logfile[256]
```

Definition at line 65 of file debugmalloc.h.

#### 4.3.2.7 max_block_size

```
long max_block_size
```

Definition at line 66 of file debugmalloc.h.

#### 4.3.2.8 tail

```
DebugmallocEntry tail[debugmalloc_tablesize]
```

Definition at line 71 of file debugmalloc.h.

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

- debugmalloc.h

## 4.4 DebugmallocEntry Struct Reference

```
#include <debugmalloc.h>
```

Collaboration diagram for DebugmallocEntry:


### Data Fields

- void * real_mem
- void * user_mem
- size_t size
- char file [64]
- unsigned line
- char func [32]
- char expr [128]
- struct DebugmallocEntry * prev
- struct DebugmallocEntry * next

### 4.4.1 Detailed Description

Definition at line 49 of file debugmalloc.h.

### 4.4.2 Field Documentation

#### 4.4.2.1 expr

```
char expr[128]
```

Definition at line 57 of file debugmalloc.h.

#### 4.4.2.2 file

```
char file[64]
```

Definition at line 54 of file debugmalloc.h.

#### 4.4.2.3 func

```
char func[32]
```

Definition at line 56 of file debugmalloc.h.

#### 4.4.2.4 line

```
unsigned line
```

Definition at line 55 of file debugmalloc.h.

#### 4.4.2.5 next

```
struct DebugmallocEntry * next
```

Definition at line 59 of file debugmalloc.h.

#### 4.4.2.6 prev

```
struct DebugmallocEntry* prev
```

Definition at line 59 of file debugmalloc.h.

#### 4.4.2.7 real_mem

```
void* real_mem
```

Definition at line 50 of file debugmalloc.h.

#### 4.4.2.8 size

```
size_t size
```

Definition at line 52 of file debugmalloc.h.

#### 4.4.2.9 user_mem

```
void* user_mem
```

Definition at line 51 of file debugmalloc.h.

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

- debugmalloc.h

## 4.5 func_type_t Struct Reference

```
#include <source_reader.h>
```

### Data Fields

- char * return_type
- char ** args

### 4.5.1 Detailed Description

Definition at line 13 of file source_reader.h.

### 4.5.2 Field Documentation

#### 4.5.2.1 args

```
char** args
```

Definition at line 15 of file source_reader.h.

#### 4.5.2.2 return_type

```
char* return_type
```

Definition at line 14 of file source_reader.h.

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

- source_reader.h

## 4.6 loop_type_t Struct Reference

```
#include <source_reader.h>
```

### Data Fields

- char * condition

### 4.6.1 Detailed Description

Definition at line 30 of file source_reader.h.

### 4.6.2 Field Documentation

#### 4.6.2.1 condition

```
char* condition
```

Definition at line 31 of file source_reader.h.

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

- source_reader.h

## 4.7 mapping_t Struct Reference

hash-map like struct for mapping strings to anything (not very safe)

```
#include <types.h>
```

### Data Fields

- const char * key
- const void * value

### 4.7.1 Detailed Description

hash-map like struct for mapping strings to anything (not very safe)

Definition at line 21 of file types.h.

### 4.7.2 Field Documentation

#### 4.7.2.1 key

```
const char* key
```

Definition at line 22 of file types.h.

#### 4.7.2.2 value

```
const void* value
```

Definition at line 23 of file types.h.

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

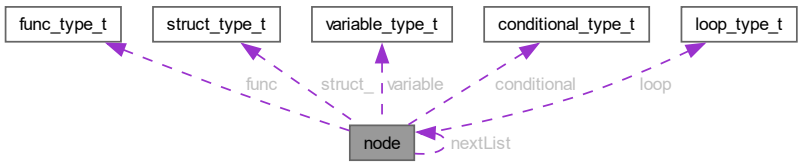
- types.h

## 4.8 node Struct Reference

linked list structure

```
#include <source_reader.h>
```

Collaboration diagram for node:



**Data Fields**

- context_e type
- char name [100]

  *type of the node*
- union {
  func_type_t func
  struct_type_t struct_
  *function*
  variable_type_t variable
  *struct*
  conditional_type_t conditional
  *variable*
  loop_type_t loop
  *conditional*
  };

  *name of the node*
- struct node ** nextList

## 4.8.1 Detailed Description

linked list structure

Definition at line 36 of file source_reader.h.

## 4.8.2 Field Documentation

### 4.8.2.1 [union]

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

name of the node

### 4.8.2.2 conditional

```
conditional_type_t conditional
```

variable

Definition at line 43 of file source_reader.h.

### 4.8.2.3 func

```
func_type_t func
```

Definition at line 40 of file source_reader.h.

#### 4.8.2.4 loop

```
loop_type_t loop
```

conditional

Definition at line 44 of file source_reader.h.

#### 4.8.2.5 name

```
char name[100]
```

type of the node

Definition at line 38 of file source_reader.h.

#### 4.8.2.6 nextList

```
struct node** nextList
```

Definition at line 46 of file source_reader.h.

#### 4.8.2.7 struct_

```
struct_type_t struct_
```

function

Definition at line 41 of file source_reader.h.

#### 4.8.2.8 type

```
context_e type
```

Definition at line 37 of file source_reader.h.

#### 4.8.2.9 variable

```
variable_type_t variable
```

struct

Definition at line 42 of file source_reader.h.

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

- source_reader.h

## 4.9 struct_type_t Struct Reference

`#include <source_reader.h>`

### Data Fields

- char * args

### 4.9.1 Detailed Description

Definition at line 18 of file source_reader.h.

### 4.9.2 Field Documentation

#### 4.9.2.1 args

`char* args`

Definition at line 19 of file source_reader.h.

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

- source_reader.h

## 4.10 test Struct Reference

### Data Fields

- int a

### 4.10.1 Detailed Description

Definition at line 6 of file test.c.

### 4.10.2 Field Documentation

#### 4.10.2.1 a

`int a`

Definition at line 7 of file test.c.

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

- test.c

## 4.11 test_t Struct Reference

### Data Fields

- int b

### 4.11.1 Detailed Description

Definition at line 9 of file test.c.

### 4.11.2 Field Documentation

#### 4.11.2.1 b

```
int b
```

Definition at line 10 of file test.c.

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

- test.c

## 4.12 theme_t Struct Reference

struct for theme

```
#include <ini_reader.h>
```

Collaboration diagram for theme_t:

**Data Fields**

- colour_t functions
- colour_t structs
- colour_t variables
- colour_t conditionals
- colour_t loops
- colour_t main_

## 4.12.1 Detailed Description

struct for theme

Definition at line 42 of file ini_reader.h.

## 4.12.2 Field Documentation

### 4.12.2.1 conditionals

```
colour_t conditionals
```

Definition at line 46 of file ini_reader.h.

### 4.12.2.2 functions

```
colour_t functions
```

Definition at line 43 of file ini_reader.h.

### 4.12.2.3 loops

```
colour_t loops
```

Definition at line 47 of file ini_reader.h.

### 4.12.2.4 main_

```
colour_t main_
```

Definition at line 48 of file ini_reader.h.

### 4.12.2.5 structs

```
colour_t structs
```

Definition at line 44 of file ini_reader.h.

#### 4.12.2.6 variables

```
colour_t variables
```

Definition at line 45 of file ini_reader.h.

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

- ini_reader.h

## 4.13 variable_type_t Struct Reference

```
#include <source_reader.h>
```

### Data Fields

- char * value

### 4.13.1 Detailed Description

Definition at line 22 of file source_reader.h.

### 4.13.2 Field Documentation

#### 4.13.2.1 value

```
char* value
```

Definition at line 23 of file source_reader.h.

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

- source_reader.h

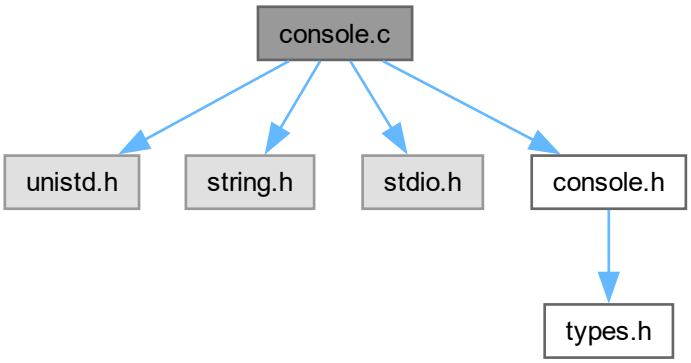
# Chapter 5

# File Documentation

## 5.1 console.c File Reference

```
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include "console.h"
```

Include dependency graph for console.c:




### Functions

- int init_console (int argc, char **argv, char *theme_file, char *output_file, char *in_file)

  *This function is used to initialize from the console.*
- file_type_e ends_with (char *file)

  *This function is used to get the file type.*

### 5.1.1 Function Documentation

#### 5.1.1.1 ends_with()

```
file_type_e ends_with (
            char * file )
```

This function is used to get the file type.

**Parameters**

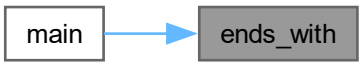
| | |
|---|---|
| *file* | file path |

**Returns**

file_type

Definition at line 39 of file console.c.

Here is the caller graph for this function:




#### 5.1.1.2 init_console()

```
int init_console (
            int argc,
            char ** argv,
            char * theme_file,
            char * output_file,
            char * in_file )
```

This function is used to initialize from the console.

**Parameters**

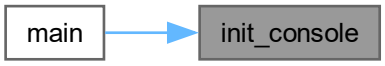
| | |
|---|---|
| *argc* | argument_cont from main |
| *argv* | arguments list from main |
| *theme_file* | theme file path |
| *output_file* | output file path |

**Returns**

0 if success, -1 if error

Definition at line 8 of file console.c.

Here is the caller graph for this function:




## 5.2 console.c

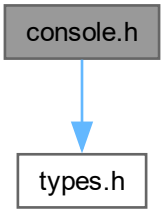
Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 16/10/2023.
00003 //
00004 #include <unistd.h>
00005 #include <string.h>
00006 #include <stdio.h>
00007 #include "console.h"
00008 int init_console(int argc, char** argv, char* theme_file, char* output_file, char* in_file) {
00009     int c;
00010     while ((c = getopt(argc,argv,":o:t:h")) != -1) {
00011         switch (c) {
00012             case 't':
00013                 strcpy(theme_file, optarg);
00014                 break;
00015             case 'h':
00016                 printf("Usage: console -t <theme_file> -o <output_file> -i <input_file>\n");
00017                 return  -1;
00018             case 'o':
00019                 strcpy(output_file, optarg);
00020                 break;
00021             case ':':
00022                 strcpy(in_file, optarg);
00023                 break;
00024             case '?':
00025                 if (optopt == 'o') {
00026                     fprintf(stderr, "Option -%c requires an argument.\n", optopt);
00027                 } else if (optopt == 't') {
00028                     fprintf(stderr, "Option -%c requires an argument.\n", optopt);
00029                 } else {
00030                     fprintf(stderr, "Unknown option `-%c'.\n", optopt);
00031                 }
00032             default:
00033                 break;
00034         }
00035     }
00036     return 0;
00037 }
00038
00039 file_type_e ends_with(char* file)
00040 {
00041     char* fileExt = strrchr(file, '.');
00042     if (strcmp(fileExt, file) != 0 || fileExt != NULL)
00043     {
00044         if (strcmp(fileExt, ".jpg") == 0)
00045             return file_type_jpg;
00046         else if (strcmp(fileExt, ".png") == 0)
00047             return file_type_png;
00048         else if (strcmp(fileExt, ".c") == 0)
00049             return file_type_c;
00050         else if (strcmp(fileExt, ".h") == 0)
00051             return file_type_h;
00052         /*else if (strcmp(fileExt, "md") == 0)
00053             return file_type_md;*/
00054     }
00055     return DEFAULT_FILE_TYPE;
00056 }
```

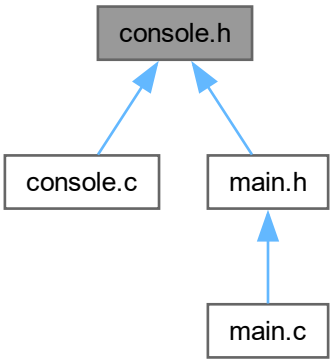
## 5.3 console.h File Reference

```
#include "types.h"
```

Include dependency graph for console.h:




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




### Functions

- int init_console (int argc, char **argv, char *theme_file, char *output_file, char *in_file)

  *This function is used to initialize from the console.*
- file_type_e ends_with (char *file)

  *This function is used to get the file type.*

### 5.3.1 Function Documentation

#### 5.3.1.1 ends_with()

```
file_type_e ends_with (
            char * file )
```

This function is used to get the file type.

**Parameters**

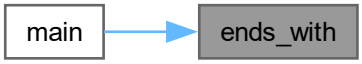
| | |
|---|---|
| *file* | file path |

**Returns**

file_type

Definition at line 39 of file console.c.

Here is the caller graph for this function:




#### 5.3.1.2 init_console()

```
int init_console (
            int argc,
            char ** argv,
            char * theme_file,
            char * output_file,
            char * in_file )
```

This function is used to initialize from the console.

**Parameters**

| | |
|---|---|
| *argc* | argument_cont from main |
| *argv* | arguments list from main |
| *theme_file* | theme file path |
| *output_file* | output file path |

**Returns**

0 if success, -1 if error

Definition at line 8 of file console.c.

Here is the caller graph for this function:


## 5.4 console.h

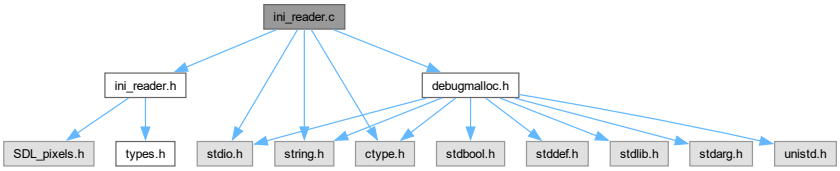
Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 16/10/2023.
00003 //
00004
00005 #ifndef NHF_CONSOLE_H
00006 #define NHF_CONSOLE_H
00007 #include "types.h"
00016 int init_console(int argc, char** argv, char* theme_file, char* output_file, char* in_file);
00022 file_type_e ends_with(char* file);
00023 #endif //NHF_CONSOLE_H
```

## 5.5 ini_reader.c File Reference

```
#include "ini_reader.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "debugmalloc.h"
```

Include dependency graph for ini_reader.c:




### Functions

- int read_ini (const char *filename, theme_t *theme)

  *reads the theme ini file for custom themes*
- void stoLower (char *str)
- void set_rgba (char *hex, SDL_Colour *colour)

  *Sets the rgba of an SDL_Colour.*

### 5.5.1 Function Documentation

#### 5.5.1.1 read_ini()

```
int read_ini (
            const char * filename,
            theme_t * theme )
```

reads the theme ini file for custom themes

**Parameters**

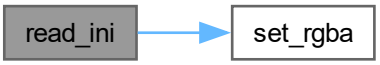
| | |
|---|---|
| *filename* | name of the ini file (including the .ini) |
| *theme* | pointer to the theme variable |

**Returns**

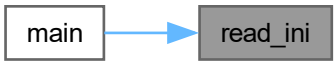
0 if ok, 1 if error

Definition at line 11 of file ini_reader.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.5.1.2 set_rgba()

```
void set_rgba (
            char * hex,
            SDL_Colour * colour )
```

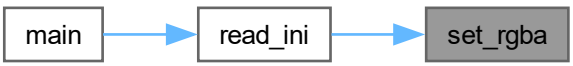
Sets the rgba of an SDL_Colour.

**Parameters**

| | |
|---|---|
| *hex* | hexadecimal string beginning with an # |
| *colour* | pointer to the SDL_Colour |

Definition at line 88 of file ini_reader.c.

Here is the caller graph for this function:




#### 5.5.1.3 stoLower()

```
void stoLower (
            char * str )
```

Definition at line 81 of file ini_reader.c.

## 5.6 ini_reader.c

Go to the documentation of this file.

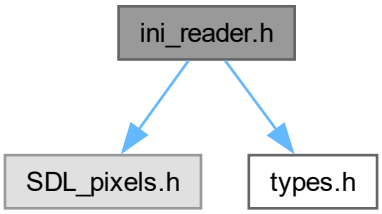
```
00001 //
00002 // Created by sziha on 16/10/2023.
00003 //
00004
00005 #include "ini_reader.h"
00006 #include <stdio.h>
00007 #include <string.h>
00008 #include <ctype.h>
00009 #include "debugmalloc.h"
00010
00011 int read_ini(const char *filename, theme_t *theme) {
00012     FILE *fp = fopen(filename, "r");
00013     if (fp == NULL) {
00014         fprintf(stderr,"Couldn't open file");
00015         return -1;
00016     }
00017     mapping_t mappings_sc[2];
00018     mapping_t mappings_c[] = {
00019             {"functions", &(theme->functions)},
00020             {"structs", &(theme->structs)},
00021             {"vars", &(theme->variables)},
00022             {"conds", &(theme->conditionals)},
00023             {"loops", &(theme->loops)},
00024             {"main", &(theme->main_)},
00025     };
00026     //printf("file open\n") ;
00027     char line[256]; //if context == -1 go to next line else check subcontext and add to theme
00028     colour_t *context = NULL;
00029     SDL_Colour *subContext = NULL;
00030     while (fgets(line, 256, fp) != NULL) {
00031         //printf("%s", line);
00032         if (line[0] == '[') {
00033             char *name = line + 1;
```

```
00034             name = strtok(name, "]");
00035             name = strlwr(name);
00036             //printf("%s\n", name);
00037             for (int i = 0; i < 6; i++) {
00038                 if (strcmp(name, mappings_c[i].key) == 0) {
00039                     context = (colour_t *) mappings_c[i].value;
00040                     break;
00041                 }
00042             }
00043             if (context == NULL) {
00044                 fprintf(stderr, "Unknown context: %s\n", name);
00045             }
00046             mappings_sc[0].key = "background";
00047             mappings_sc[0].value = &(context->background);
00048             mappings_sc[1].key = "text";
00049             mappings_sc[1].value = &(context->text);
00050             continue;
00051         }
00052         else if (line[0] != ';' && context != NULL)
00053         {
00054             char *value = strtok(line, "=");
00055             unsigned long long int val_len = strlen(value);
00056             //printf("%s", value);
00057             char *valend = value+val_len-1;
00058             while (isspace(*valend)){
00059                 *valend = '\0';
00060                 valend--;
00061             }
00062             for (int i = 0; i < 2; i++) {
00063                 if (strcmp(value, mappings_sc[i].key) == 0) {
00064                     subContext = (SDL_Color *) mappings_sc[i].value;
00065                     break;
00066                 }
00067             }
00068             if (subContext == NULL) {
00069                 fprintf(stderr, "Unknown sub context: %s\n", value);
00070             }
00071             value = strtok(NULL, "=");
00072             while (isspace(*value))
00073                 value++;
00074             set_rgba(value, subContext);
00075         }
00076     }
00077     fclose(fp);
00078     return 0;
00079 }
00080
00081 void stoLower(char *str) {
00082     while (*str != '\0') {
00083         *str = (char) tolower(*str);
00084         str++;
00085     }
00086 }
00087
00088 void set_rgba(char *hex, SDL_Colour *colour) {
00089     char rgba[3] = {hex[1], hex[2],'\0'};
00090     colour->r = strtoul(rgba, NULL, 16);
00091     rgba[0] = hex[3]; rgba[1] = hex[4];
00092     colour->g = strtoul(rgba, NULL, 16);
00093     rgba[0] = hex[5]; rgba[1] = hex[6];
00094     colour->b = strtoul(rgba, NULL, 16);
00095     rgba[0] = hex[7]; rgba[1] = hex[8];
00096     colour->a = strtoul(rgba, NULL, 16);
00097 }
```

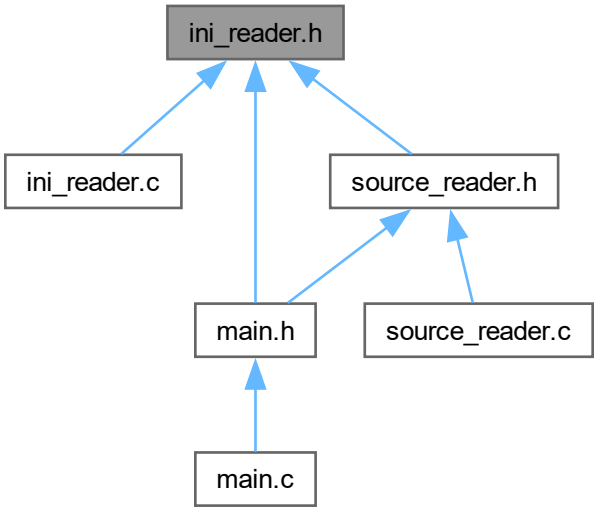
## 5.7 ini_reader.h File Reference

```
#include <SDL_pixels.h>
#include "types.h"
```

Include dependency graph for ini_reader.h:




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




### Data Structures

- struct colour_t
  *colouring struct for theme*
- struct theme_t
  *struct for theme*

### Enumerations

- enum context_e {
  function , structs , variable , conditional ,
  loop , main_ }

*enum for ini file context*

- enum sub_context_e { background , text }

  *enum for ini file sub_context (in ini documentation is named value, but i use it like another context so it doesnt matter)*

### Functions

- int read_ini (const char *filename, theme_t *theme)

  *reads the theme ini file for custom themes*

- void set_rgba (char *hex, SDL_Colour *colour)

  *Sets the rgba of an SDL_Colour.*

## 5.7.1 Enumeration Type Documentation

### 5.7.1.1 context_e

```
enum context_e
```

enum for ini file context

**Enumerator**

| | |
|---|---|
| function | |
| structs | |
| variable | |
| conditional | |
| loop | |
| main_ | |

Definition at line 14 of file ini_reader.h.

### 5.7.1.2 sub_context_e

```
enum sub_context_e
```

enum for ini file sub_context (in ini documentation is named value, but i use it like another context so it doesnt matter)

**Enumerator**

| | |
|---|---|
| background | |
| text | |

Definition at line 26 of file ini_reader.h.

### 5.7.2 Function Documentation

#### 5.7.2.1 read_ini()

```
int read_ini (
            const char * filename,
            theme_t * theme )
```

reads the theme ini file for custom themes

**Parameters**

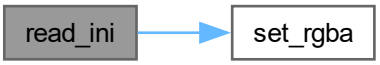
| | |
|---|---|
| *filename* | name of the ini file (including the .ini) |
| *theme* | pointer to the theme variable |

**Returns**

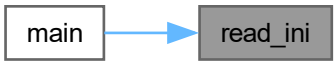
0 if ok, 1 if error

Definition at line 11 of file ini_reader.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.7.2.2 set_rgba()

```
void set_rgba (
            char * hex,
            SDL_Colour * colour )
```

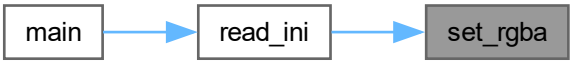
Sets the rgba of an SDL_Colour.

**Parameters**

| | |
|---|---|
| *hex* | hexadecimal string beginning with an # |
| *colour* | pointer to the SDL_Colour |

Definition at line 88 of file ini_reader.c.

Here is the caller graph for this function:




## 5.8 ini_reader.h

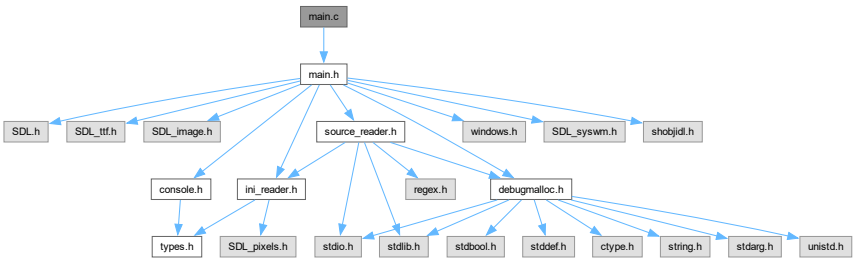
Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 16/10/2023.
00003 //
00004
00005 #ifndef NHF_INI_READER_H
00006 #define NHF_INI_READER_H
00007
00008 #include <SDL_pixels.h>
00009 #include "types.h"
00010
00014 typedef enum {
00015     function,
00016     structs,
00017     variable,
00018     conditional,
00019     loop,
00020     main_
00021 } context_e;
00026 typedef enum {
00027     background,
00028     text
00029 } sub_context_e;
00030
00034 typedef struct {
00035     SDL_Colour background;
00036     SDL_Colour text;
00037 } colour_t;
00038
00042 typedef struct {
00043     colour_t functions;
00044     colour_t structs;
00045     colour_t variables;
00046     colour_t conditionals;
00047     colour_t loops;
00048     colour_t main_;
00049 } theme_t;
00050
00057 int read_ini(const char *filename, theme_t *theme);
00063 void set_rgba(char *hex, SDL_Colour *colour);
00064 #endif //NHF_INI_READER_H
```

## 5.9 main.c File Reference

```
#include "main.h"
```

Include dependency graph for main.c:




### Functions

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

  *Obvious.*
- HWND GetHwnd (SDL_Window *window)

  *Gets win32 window handle.*
- void ActivateMenu (HWND windowRef)

  *Creates a menu for the given window handle.*
- char * file_open_dialog (HWND windowRef, const wchar_t *name, const wchar_t *file_spec)

  *Creates a file open dialog for opening source files.*
- char * file_save_dialog (HWND windowRef)

  *Creates a file save dialog for saving image files.*

### 5.9.1 Function Documentation

#### 5.9.1.1 ActivateMenu()

```
void ActivateMenu (
            HWND windowRef )
```

Creates a menu for the given window handle.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |

Definition at line 121 of file main.c.

Here is the caller graph for this function:


#### 5.9.1.2 file_open_dialog()

```
char * file_open_dialog (
            HWND windowRef,
            const wchar_t * name,
            const wchar_t * file_spec )
```

Creates a file open dialog for opening source files.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |
| *name* | filter name |
| *file_spec* | filter spec |

**Returns**

file to open

Definition at line 143 of file main.c.

Here is the caller graph for this function:


#### 5.9.1.3 file_save_dialog()

```
char * file_save_dialog (
            HWND windowRef )
```

Creates a file save dialog for saving image files.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |

**Returns**

file to save

Definition at line 190 of file main.c.

Here is the caller graph for this function:


#### 5.9.1.4 GetHwnd()

```
HWND GetHwnd (
            SDL_Window * window )
```

Gets win32 window handle.

**Parameters**

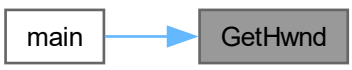
| | |
|---|---|
| *window* | sdl window |

**Returns**

win32 window handle

Definition at line 115 of file main.c.

Here is the caller graph for this function:



#### 5.9.1.5 main()

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

Obvious.

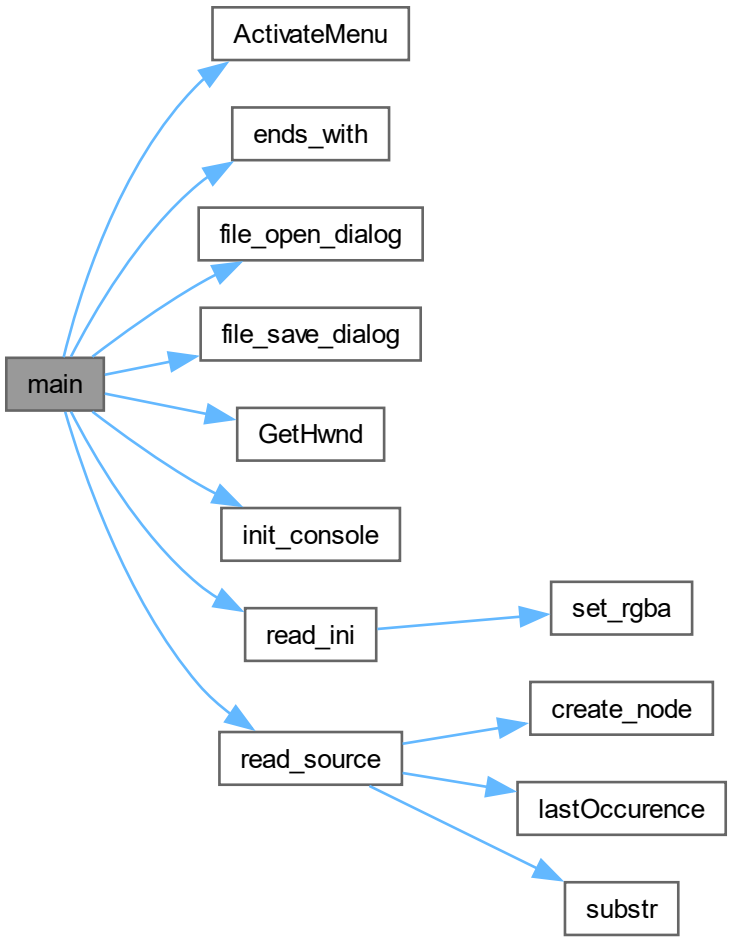
**Parameters**

| | |
|---|---|
| *argc* | |
| *argv* | |

**Returns**

Definition at line 3 of file main.c.

Here is the call graph for this function:



## 5.10 main.c

Go to the documentation of this file.

```
00001 #include "main.h"
00002
00003 int main(int argc, char** argv) {
00004     char *theme_file = "theme.ini";
00005     char *input_file = "test.c";
00006     char *output_file = "test";
00007     theme_t *theme = &default_theme;
00008     printf("%u %u %u", theme->main_.background.r, theme->main_.background.g,
      theme->main_.background.b);
00009     file_type_e output_type;
00010     file_type_e input_type;
00011     node_t *root = NULL;
00012     int quit = 0;
00013     HWND windowRef;
00014     SDL_Event event;
00015
00016     if (argc != 1){
00017         if(init_console(argc, argv, theme_file, output_file, input_file) == -1) return -1;
00018         output_type = ends_with(output_file);
00019         input_type = ends_with(input_file);
00020     }
00021     if (read_ini(theme_file, theme) == -1) theme = &default_theme;
00022     SDL_Window *window = SDL_CreateWindow("Source to Flow", SDL_WINDOWPOS_UNDEFINED,
      SDL_WINDOWPOS_UNDEFINED, 640, 480, SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE);
00023     SDL_Renderer *renderer = SDL_CreateRenderer(window, -1, 0);
00024     SDL_Surface *surface = SDL_CreateRGBSurface(1,640,480,32,0,0,0,0);
00025     SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);
00026     SDL_FreeSurface(surface);
00027     windowRef = GetHwnd(window);
00028     ActivateMenu(windowRef);
00029     SDL_EventState(SDL_SYSWMEVENT, SDL_ENABLE);
00030     if (input_type == file_type_c){
00031         root = read_source(input_file);
00032     }
00033     if (output_type == file_type_c){
00034         fprintf(stderr,"HOW");
00035     }
00036     while (!quit) {
00037         SDL_PollEvent(&event);
00038         switch (event.type) {
00039             case SDL_WINDOWEVENT_CLOSE:
00040                 event.type = SDL_QUIT;
00041                 SDL_PushEvent(&event);
00042                 break;
00043             case SDL_QUIT:
00044                 quit = 1;
00045                 break;
00046             case SDL_SYSWMEVENT:
00047                 if (event.syswm.msg->msg.win.msg == WM_COMMAND){
00048                     char* temp;
00049                     switch (event.syswm.msg->msg.win.wParam){
00050                         case ID_EXIT:
00051                             quit = 1;
00052                             break;
00053                         case ID_OPEN_FILE:
00054                             ;
00055                             temp = file_open_dialog(windowRef,L"Source file",L"*.c");
00056                             if (temp == NULL) break;
00057                             //free(input_file);
00058                             input_file = temp;
00059                             input_type = ends_with(input_file);
00060                             root = read_source(input_file);
00061                             //printf("%s", input_file);
00062                             break;
00063                         case ID_SAVE_FLOW:
00064                             ;
00065                             temp = file_save_dialog(windowRef);
00066                             if (temp == NULL) break;
00067                             //free(output_file);
00068                             output_file = temp;
00069                             output_type = ends_with(output_file);
00070                             //printf("%s", output_file);
00071                             break;
00072                         case ID_LOAD_THEME:
00073                             temp = file_open_dialog(windowRef,L"Theme file",L"*.ini");
00074                             if (temp == NULL) break;
00075                             free(theme_file);
00076                             theme_file = temp;
00077                             if (read_ini(theme_file, theme) == -1){
00078                                 theme = &default_theme;
00079                                 SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR, "Theme error",
      "Couldn't load theme", window);
```

```
00080                                 break;
00081                             }
00082                             break;
00083                         case ID_RESET_THEME:
00084                             theme = &default_theme;
00085                             //TODO: redraw call
00086                             break;
00087                         case ID_ZOOM_IN:
00088                             break;
00089                         case ID_ZOOM_OUT:
00090                             break;
00091                         case ID_ZOOM_RESET:
00092                             break;
00093                         default:
00094                             break;
00095                     }
00096                 }
00097                 break;
00098         }
00099
00100         //SDL_SetRenderDrawColor(renderer, theme->main_.background.r, theme->main_.background.g,
      theme->main_.background.b, theme->main_.background.a);
00101         //SDL_RenderClear(renderer);
00102         SDL_RenderPresent(renderer);
00103     }
00104     SDL_DestroyTexture(texture);
00105     SDL_DestroyRenderer(renderer);
00106     SDL_DestroyWindow(window);
00107     SDL_Quit();
00108     //free(theme);
00109     //free(input_file);
00110     //free(output_file);
00111     free(root);
00112     return 0;
00113 }
00114
00115 HWND GetHwnd(SDL_Window *window){
00116     SDL_SysWMinfo windowInfo;
00117     if(!SDL_GetWindowWMInfo(window,&windowInfo)) return NULL;
00118     return windowInfo.info.win.window;
00119 }
00120
00121 void ActivateMenu(HWND windowRef)
00122 {
00123     HMENU hMenuBar = CreateMenu();
00124     HMENU hFile = CreateMenu();
00125     HMENU hView = CreateMenu();
00126
00127     AppendMenu(hMenuBar, MF_POPUP, (UINT_PTR)hFile, "File");
00128     AppendMenu(hMenuBar, MF_POPUP, (UINT_PTR)hView, "View");
00129     AppendMenu(hMenuBar, MF_STRING, ID_EXIT, "Exit");
00130
00131     AppendMenu(hFile, MF_STRING, ID_OPEN_FILE, "Open File");
00132     AppendMenu(hFile, MF_STRING, ID_SAVE_FLOW, "Save flowchart");
00133
00134     AppendMenu(hView, MF_STRING, ID_LOAD_THEME, "Load Theme");
00135     AppendMenu(hView, MF_STRING, ID_RESET_THEME, "Reset Theme");
00136     AppendMenu(hView, MF_STRING, ID_ZOOM_IN, "Zoom In");
00137     AppendMenu(hView, MF_STRING, ID_ZOOM_OUT, "Zoom Out");
00138     AppendMenu(hView, MF_STRING, ID_ZOOM_RESET, "Zoom Reset");
00139
00140     SetMenu(windowRef, hMenuBar);
00141 }
00142
00143 char* file_open_dialog(HWND windowRef, const wchar_t *name, const wchar_t *file_spec)
00144 {
00145     char *file_path = NULL;
00146     HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
00147                                       COINIT_DISABLE_OLE1DDE);
00148     COMDLG_FILTERSPEC filterspec = {name, file_spec};
00149     if (SUCCEEDED(hr))
00150     {
00151         IFileOpenDialog *pFileOpen;
00152
00153         // Create the FileOpenDialog object.
00154         hr = CoCreateInstance(&CLSID_FileOpenDialog, NULL, CLSCTX_ALL,
00155                               &IID_IFileOpenDialog, (void**)(&pFileOpen));
00156         if (SUCCEEDED(hr))
00157         {
00158             pFileOpen->lpVtbl->SetFileTypes(pFileOpen, 1, &filterspec);
00159             // Show the Open dialog box.
00160             hr = pFileOpen->lpVtbl->Show(pFileOpen, windowRef);
00161
00162             // Get the file name from the dialog box.
00163             if (SUCCEEDED(hr))
00164             {
00165                 IShellItem *pItem;
```

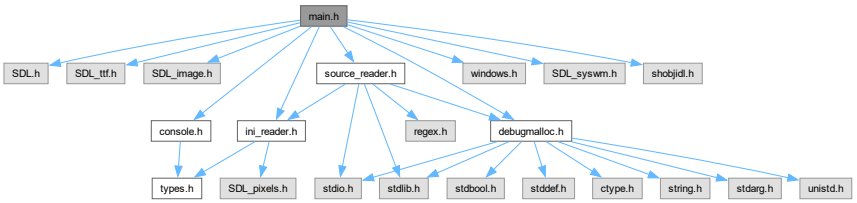
```
00166                 hr = pFileOpen->lpVtbl->GetResult(pFileOpen, &pItem);
00167                 if (SUCCEEDED(hr))
00168                 {
00169                     PWSTR pszFilePath;
00170                     hr = pItem->lpVtbl->GetDisplayName(pItem,SIGDN_FILESYSPATH,&pszFilePath);
00171 
00172                     // Display the file name to the user.
00173                     if (SUCCEEDED(hr))
00174                     {
00175                         //MessageBoxW(NULL, pszFilePath, L"File Path", MB_OK);
00176                         file_path = (char *)malloc(lstrlenW(pszFilePath) + 1);
00177                         wcstombs(file_path, pszFilePath, lstrlenW(pszFilePath) + 1);
00178                         CoTaskMemFree(pszFilePath);
00179                     }
00180                     pItem->lpVtbl->Release((IShellItem *) &pItem);
00181                 }
00182             }
00183             pFileOpen->lpVtbl->Release((IFileOpenDialog *) &pFileOpen);
00184         }
00185         CoUninitialize();
00186     }
00187     return file_path;
00188 }
00189 
00190 char* file_save_dialog(HWND windowRef)
00191 {
00192     char *file_path = NULL;
00193     HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
00194                                       COINIT_DISABLE_OLE1DDE);
00195     COMDLG_FILTERSPEC filterspec[2] = {{L"png", L"*.png"}, {L"jpg", L"*.jpg"}}; //TODO: add md later
00196     if (SUCCEEDED(hr))
00197     {
00198         IFileSaveDialog *pFileSave;
00199 
00200         // Create the FileOpenDialog object.
00201         hr = CoCreateInstance(&CLSID_FileSaveDialog, NULL, CLSCTX_ALL,
00202                               &IID_IFileSaveDialog, (void**)(&pFileSave));
00203         if (SUCCEEDED(hr))
00204         {
00205             pFileSave->lpVtbl->SetFileTypes(pFileSave, 2, filterspec);
00206             pFileSave->lpVtbl->SetFileName(pFileSave, L"Flowchart.png");
00207             // Show the Open dialog box.
00208             hr = pFileSave->lpVtbl->Show(pFileSave, windowRef);
00209             // Get the file name from the dialog box.
00210             if (SUCCEEDED(hr))
00211             {
00212                 IShellItem *pItem;
00213                 unsigned int i;
00214                 pFileSave->lpVtbl->GetFileTypeIndex(pFileSave, &i);
00215                 hr = pFileSave->lpVtbl->GetResult(pFileSave, &pItem);
00216                 if (SUCCEEDED(hr))
00217                 {
00218                     PWSTR pszFilePath;
00219                     hr = pItem->lpVtbl->GetDisplayName(pItem,SIGDN_FILESYSPATH,&pszFilePath);
00220 
00221                     // Display the file name to the user.
00222                     if (SUCCEEDED(hr))
00223                     {
00224                         //MessageBoxW(NULL, pszFilePath, L"File Path", MB_OK);
00225                         file_path = (char *)malloc(lstrlenW(pszFilePath) + 5);
00226                         wcstombs(file_path, pszFilePath, lstrlenW(pszFilePath) + 1);
00227                         strcat(file_path, i == 1 ? ".png" : ".jpg");
00228                         CoTaskMemFree(pszFilePath);
00229                     }
00230                     pItem->lpVtbl->Release((IShellItem *) &pItem);
00231                 }
00232             }
00233             pFileSave->lpVtbl->Release((IFileSaveDialog *) &pFileSave);
00234         }
00235         CoUninitialize();
00236     }
00237     return file_path;
00238 }
```

## 5.11 main.h File Reference

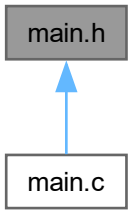
```
#include <SDL.h>
#include <SDL_ttf.h>
#include <SDL_image.h>
#include "console.h"
```

```
#include "ini_reader.h"
#include <windows.h>
#include <SDL_syswm.h>
#include <shobjidl.h>
#include "source_reader.h"
#include "debugmalloc.h"
```

Include dependency graph for main.h:




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




### Macros

- #define ID_OPEN_FILE 1
- #define ID_SAVE_FLOW 2
- #define ID_LOAD_THEME 3
- #define ID_RESET_THEME 4
- #define ID_ZOOM_IN 5
- #define ID_ZOOM_OUT 6
- #define ID_ZOOM_RESET 7
- #define ID_EXIT 8

### Functions

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

  *Obvious.*
- HWND GetHwnd (SDL_Window *window)

  *Gets win32 window handle.*
- void ActivateMenu (HWND windowRef)

*Creates a menu for the given window handle.*

- char * file_open_dialog (HWND windowRef, const wchar_t *name, const wchar_t *file_spec)

  *Creates a file open dialog for opening source files.*

- char * file_save_dialog (HWND windowRef)

  *Creates a file save dialog for saving image files.*

### 5.11.1 Macro Definition Documentation

#### 5.11.1.1 ID_EXIT

```
#define ID_EXIT 8
```

Definition at line 35 of file main.h.

#### 5.11.1.2 ID_LOAD_THEME

```
#define ID_LOAD_THEME 3
```

Definition at line 30 of file main.h.

#### 5.11.1.3 ID_OPEN_FILE

```
#define ID_OPEN_FILE 1
```

Definition at line 28 of file main.h.

#### 5.11.1.4 ID_RESET_THEME

```
#define ID_RESET_THEME 4
```

Definition at line 31 of file main.h.

#### 5.11.1.5 ID_SAVE_FLOW

```
#define ID_SAVE_FLOW 2
```

Definition at line 29 of file main.h.

#### 5.11.1.6 ID_ZOOM_IN

```
#define ID_ZOOM_IN 5
```

Definition at line 32 of file main.h.

#### 5.11.1.7 ID_ZOOM_OUT

```
#define ID_ZOOM_OUT 6
```

Definition at line 33 of file main.h.

#### 5.11.1.8 ID_ZOOM_RESET

```
#define ID_ZOOM_RESET 7
```

Definition at line 34 of file main.h.

### 5.11.2 Function Documentation

#### 5.11.2.1 ActivateMenu()

```
void ActivateMenu (
            HWND windowRef )
```

Creates a menu for the given window handle.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |

Definition at line 121 of file main.c.

Here is the caller graph for this function:


#### 5.11.2.2 file_open_dialog()

```
char * file_open_dialog (
            HWND windowRef,
            const wchar_t * name,
            const wchar_t * file_spec )
```

Creates a file open dialog for opening source files.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |
| *name* | filter name |
| *file_spec* | filter spec |

**Returns**

file to open

Definition at line 143 of file main.c.

Here is the caller graph for this function:


#### 5.11.2.3 file_save_dialog()

```
char * file_save_dialog (
            HWND windowRef )
```

Creates a file save dialog for saving image files.

**Parameters**

| | |
|---|---|
| *windowRef* | win32 window handle |

**Returns**

file to save

Definition at line 190 of file main.c.

Here is the caller graph for this function:

#### 5.11.2.4 GetHwnd()

```
HWND GetHwnd (
            SDL_Window * window )
```

Gets win32 window handle.

**Parameters**

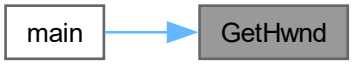
| | |
|---|---|
| *window* | sdl window |

**Returns**

win32 window handle

Definition at line 115 of file main.c.

Here is the caller graph for this function:




#### 5.11.2.5 main()

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

Obvious.

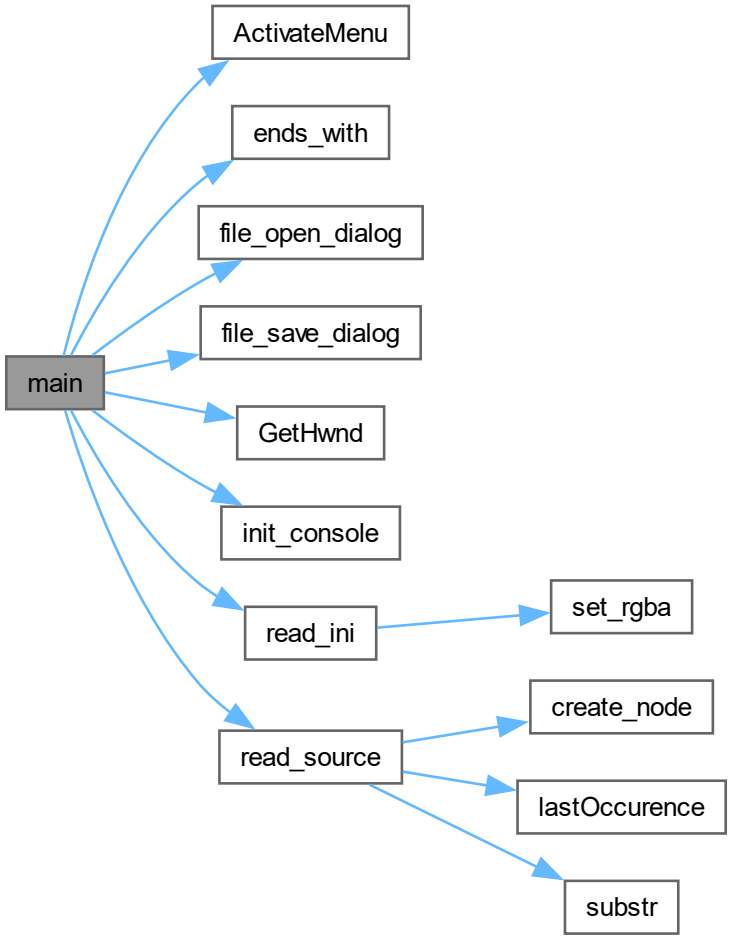
**Parameters**

| | |
|---|---|
| *argc* | |
| *argv* | |

**Returns**

Definition at line 3 of file main.c.

Here is the call graph for this function:




## 5.12 main.h

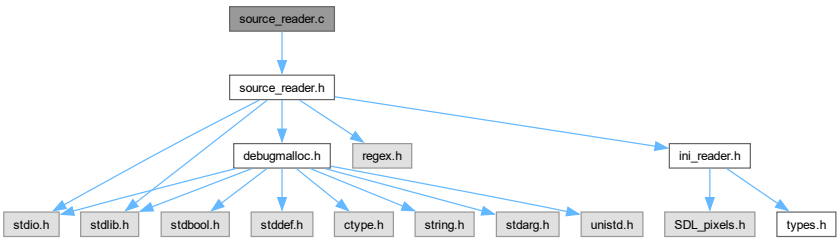
Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 18/10/2023.
00003 //
00004
00005 #ifndef NHF_MAIN_H
00006 #define NHF_MAIN_H
00007 #endif//NHF_MAIN_H
00008
00009 #include <SDL.h>
00010 #include <SDL_ttf.h>
00011 #include <SDL_image.h>
00012 #include "console.h"
00013 #include "ini_reader.h"
00014 #include <windows.h>
00015 #include <SDL_syswm.h>
00016 #include <shobjidl.h>
00017 #include "source_reader.h"
00018 #include "debugmalloc.h"
00019
00026 int main(int argc, char** argv);
```

```
00027
00028 #define ID_OPEN_FILE 1
00029 #define ID_SAVE_FLOW 2
00030 #define ID_LOAD_THEME 3
00031 #define ID_RESET_THEME 4
00032 #define ID_ZOOM_IN 5
00033 #define ID_ZOOM_OUT 6
00034 #define ID_ZOOM_RESET 7
00035 #define ID_EXIT 8
00036
00042 HWND GetHwnd(SDL_Window *window);
00047 void ActivateMenu(HWND windowRef);
00048 //code from
      https://stackoverflow.com/questions/51250046/sdl2-win32-api-menubar-click-event-not-working
00056 char* file_open_dialog(HWND windowRef, const wchar_t *name, const wchar_t *file_spec);
00062 char* file_save_dialog(HWND windowRef);
00063
00067 static theme_t default_theme = {
00068         .main_ = {
00069             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00070             .text = {.r = 0,.g = 0,.b = 0,.a = 255}},
00071         .functions = {
00072             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00073             .text = {.r = 0,.g = 0,.b = 0,.a = 255}},
00074         .structs = {
00075             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00076             .text = {.r = 0,.g = 0,.b = 0,.a = 255}},
00077         .variables = {
00078             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00079             .text = {.r = 0,.g = 0,.b = 0,.a = 255}},
00080         .conditionals = {
00081             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00082             .text = {.r = 0,.g = 0,.b = 0,.a = 255}},
00083         .loops = {
00084             .background = {.r = 255,.g = 255,.b = 255,.a = 255},
00085             .text = {.r = 0,.g = 0,.b = 0,.a = 255}}
00086 };
```

## 5.13 source_reader.c File Reference

`#include "source_reader.h"`
Include dependency graph for source_reader.c:




### Functions

- node_t * read_source (char *filename)

  *NOT FULLY IMPLEMENTED YET.*
- char * substr (char *str, char start, char end)

  *Returns a substirng from the first appearance of start until the first appearance of end.*
- void truncate (char *str, int len)

  *Truncates a string from spaces and endlines*
  *if len < 0, string length doesnt change*
  *if len > 0, truncates to len.*

- int isValidIdentifier (const char *str)
- void extractFunctionCallParameters (const char *str)
- int lastOccurence (char *str, char c)
- node_t * create_node (context_e type)

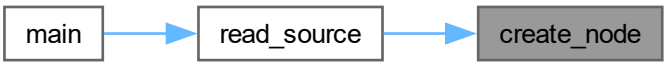
### 5.13.1 Function Documentation

#### 5.13.1.1 create_node()

```
node_t * create_node (
            context_e type )
```

Definition at line 290 of file source_reader.c.

Here is the caller graph for this function:




#### 5.13.1.2 extractFunctionCallParameters()

```
void extractFunctionCallParameters (
            const char * str )
```

Definition at line 249 of file source_reader.c.

Here is the call graph for this function:

#### 5.13.1.3 isValidIdentifier()

```
int isValidIdentifier (
            const char * str )
```

Definition at line 211 of file source_reader.c.

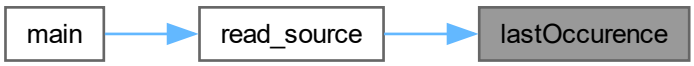
Here is the caller graph for this function:


#### 5.13.1.4 lastOccurence()

```
int lastOccurence (
            char * str,
            char c )
```

Definition at line 282 of file source_reader.c.

Here is the caller graph for this function:




#### 5.13.1.5 read_source()

```
node_t * read_source (
            char * filename )
```

NOT FULLY IMPLEMENTED YET.

**Parameters**

| | |
|---|---|
| *source_file* | to read |

**Returns**

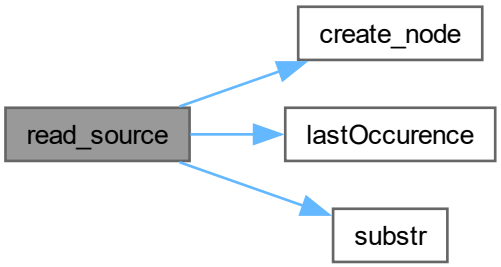
linked_list of all the lines

is full struct

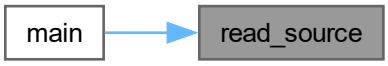
is function

Definition at line 8 of file source_reader.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.13.1.6 substr()

```
char * substr (
            char * str,
            char start,
            char end )
```

Returns a substirng from the first appearance of start until the first appearance of end.

**Parameters**

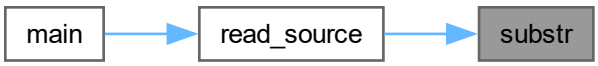
| | |
|---|---|
| *str* | string to get substring from |
| *start* | starting char |
| *end* | ending char |

**Returns**

Substring from start char (inclusive) to end char (non inclusive)

Definition at line 181 of file source_reader.c.

Here is the caller graph for this function:




#### 5.13.1.7 truncate()

```
void truncate (
            char * str,
            int len )
```

Truncates a string from spaces and endlines
if len $<$ 0, string length doesnt change
if len $>$ 0, truncates to len.

**Parameters**

| | |
|---|---|
| *str* | string to get truncated |
| *len* | truncation length |

Definition at line 195 of file source_reader.c.

## 5.14 source_reader.c

Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 21/10/2023.
00003 //
00004
00005 #include "source_reader.h"
00006
00007
00008 node_t *read_source(char *filename) {
00009     FILE *fp = fopen(filename, "r");
00010     if (fp == NULL) {
00011         fprintf(stderr, "Unable to open file %s\n", filename);
00012         return NULL;
00013     }
00014     node_t *first_node = (node_t *) malloc(sizeof(node_t));
00015     node_t *current_node = first_node;
00016     current_node->type = -1;
00017     regex_t re;
00018     regmatch_t match;
00019     int regi;
```

```
00020     char buffer[10000];
00021     int skip = 0;
00022     int c = 0;
00023     int c_bracket[2] = {0,0};
00024     int n_bracket[2] = {0,0};
00025     for (int i = 0; c != EOF; i = (i == 999) ? 0 : i + 1) {
00026         c = getc(fp);
00027         if (( skip == 1  && c!='\n' )) {
00028             i--;
00029             continue;
00030         }
00031         if (c == '\n') {
00032             skip = 0;
00033             i--;
00034             continue;
00035         }
00036         switch (c) {
00037             case '{':
00038                 c_bracket[1] = c_bracket[0];
00039                 c_bracket[0]++;
00040                 break;
00041             case '}':
00042                 c_bracket[1] = c_bracket[0];
00043                 c_bracket[0]--;
00044                 break;
00045             case '(':
00046                 n_bracket[1] = n_bracket[0];
00047                 n_bracket[0]++;
00048                 break;
00049             case ')':
00050                 n_bracket[1] = n_bracket[0];
00051                 n_bracket[0]--;
00052                 break;
00053             default:
00054                 break;
00055         }
00056         skip = 0;
00057         buffer[i] = (char) c;
00058         buffer[i + 1] = '\0';
00059         if (buffer[i] == '#' || (i >= 1 && (buffer[i-1] == '/' && buffer[i] == '/'))) {
00060             i--;
00061             skip = 1;
00062             if (i >= 1 && (buffer[i-1] == '/' && buffer[i] == '/')) i--;
00063             continue;
00064         }
00065         fprintf(stderr,"%s", buffer);
00066         char *arg;
00067         switch (current_node->type) {
00068             default:
00069                 regcomp(&re, "struct *\\w*\\s*\\{(.+)\\}.*;", REG_EXTENDED);
00070                 regi = regexec(&re, buffer, 1, &match, 0);
00071                 if (!regi && c_bracket[0] == 0) {
00072                     current_node->type = structs;
00073                     current_node->struct_.args = (char *) malloc(sizeof(char) * (match.rm_eo -
    match.rm_so + 1));
00074                     strncpy(current_node->struct_.args, buffer + match.rm_so, match.rm_eo -
    match.rm_so);
00075                     current_node->struct_.args[match.rm_eo - match.rm_so] = '\0'; //args
00076                     break;
00077                 }
00078                 regcomp(&re, "\\w+\\s+\\w+\\s*\\(.*\\)\\s*\\{", REG_EXTENDED);
00079                 regi = regexec(&re, buffer, 1, &match, 0);
00080                 if (!regi && n_bracket[0] == 0) {
00081                     current_node->type = function;
00082                     current_node->func.args = (char **) malloc(sizeof(char *)*2);
00083                     current_node->func.args[0] = (char *) malloc(sizeof(char) * (match.rm_eo -
    match.rm_so + 1));
00084                     current_node->func.args[1] = NULL;
00085                     strncpy(current_node->func.args[0], buffer + match.rm_so, match.rm_eo -
    match.rm_so);
00086                     current_node->func.args[0][match.rm_eo - match.rm_so] = '\0';
00087                     break;
00088                 }
00089                 break;
00090             case structs:
00091                 regcomp(&re, "}\\ *\\w+\\ *;", REG_EXTENDED); //name
00092                 regi = regexec(&re, current_node->struct_.args, 1, &match, 0);
00093                 if (regi) {
00094                     fprintf(stderr, "No match\n");
00095                     regcomp(&re, "(struct)\\ \\w+", REG_EXTENDED);
00096                     regi = regexec(&re, current_node->struct_.args, 1, &match, 0); //name 2
00097                     if (regi) {
00098                         fprintf(stderr, "No match\n");
00099                         strcpy(current_node->name, "Default name");
00100                     } else {
00101                         strncpy(current_node->name, buffer + match.rm_so + 8, match.rm_eo -
    match.rm_so - 7);
```

```
00102                         current_node->name[match.rm_eo - match.rm_so - 7] = '\0';
00103                     }
00104                 } else {
00105                     if (*(buffer + match.rm_so + 1) == ' ') {
00106                         strncpy(current_node->name, buffer + match.rm_so + 1, match.rm_eo -
      match.rm_so - 2);
00107                         current_node->name[match.rm_eo - match.rm_so - 2] = '\0';
00108                     } else {
00109                         strncpy(current_node->name, buffer + match.rm_so + 2, match.rm_eo -
      match.rm_so - 3);
00110                         current_node->name[match.rm_eo - match.rm_so - 3] = '\0';
00111                     }
00112                 }
00113                 char *ar = current_node->struct_.args;
00114                 current_node->struct_.args = substr(current_node->struct_.args, '{', ';');
00115                 free(ar);
00116                 *(current_node->struct_.args) = '\0';
00117                 current_node->struct_.args++;
00118                 int new_len = lastOccurence(current_node->struct_.args, '}');
00119                 current_node->struct_.args = (char *) realloc(current_node->struct_.args, new_len);
00120                 current_node->struct_.args[new_len] = '\0';
00121                 node_t *new_node = create_node(-1);
00122                 current_node->nextList = (node_t **) malloc(sizeof(node_t) * 2);
00123                 current_node->nextList[0] = new_node;
00124                 current_node->nextList[1] = NULL;
00125                 current_node = new_node;
00126                 break;
00127             case function:
00128                 arg = strtok(current_node->func.args[0], " ");
00129                 size_t arg_len = 3;
00130                 if (arg != NULL) {
00131                     arg_len = strlen(arg);
00132                     current_node->func.return_type = (char *) malloc(sizeof(char) * (arg_len + 1));
00133                     strcpy(current_node->func.return_type, arg);
00134                 } else {
00135                     current_node->func.return_type = (char *) malloc(sizeof(char) * (arg_len + 1));
00136                     strcpy(current_node->func.return_type, "int");
00137                 }
00138                 arg = strtok(NULL, " ");
00139                 if (arg != NULL)
00140                     strcpy(current_node->name, arg);
00141                 else
00142                     strcpy(current_node->name, "Default name");
00143                 char *arg1 = strtok(NULL, ",(");
00144                 if (arg1 != NULL && arg1[0] != ')') {
00145                     arg1++;
00146                     arg = arg1;
00147                 }
00148                 else{
00149                     free(arg1);
00150                     free(current_node->func.args[0]);
00151                     current_node->func.args = (char **) realloc(current_node->func.args, 1);
00152                     current_node->func.args[0] = NULL;
00153                     break;
00154                 }
00155                 int n_args = 1;
00156                 while (arg != NULL && arg[0] != ')'){
00157                     current_node->func.args = (char **) realloc(current_node->func.args, n_args+2);
00158                     current_node->func.args[n_args] = (char *) malloc(sizeof(char) * (strlen(arg) +
      1));
00159                     strcpy(current_node->func.args[n_args], arg);
00160                     n_args++;
00161                     current_node->func.args[n_args] = NULL;
00162                     arg = strtok(NULL, ",(");
00163                 }
00164                 free(current_node->func.args[0]);
00165                 current_node->func.args[0] = malloc(strlen(arg1)+1);
00166                 strcpy(current_node->func.args[0], arg1);
00167
00168                 break;
00169             case variable:
00170                 break;
00171             case conditional:
00172                 break;
00173             case loop:
00174                 break;
00175         }
00176     }
00177     fclose(fp);
00178     return first_node;
00179 }
00180
00181 char* substr(char *str, char start, char end) {
00182     char* s = str;
00183     if (start >= 0) s = strchr(str, start);
00184     char* e = strchr(str, end);
00185     if (s == NULL || e == NULL)
```

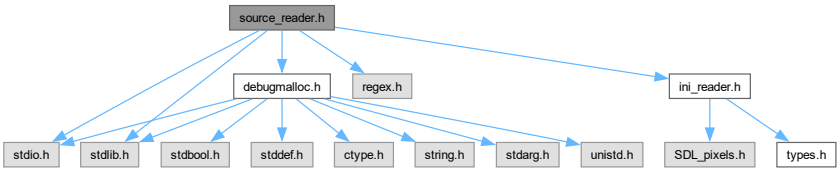
```
00186         return NULL;
00187     char* result = malloc((e - s) + 1);
00188     if (result == NULL)
00189         return NULL;
00190     strncpy(result, s, e-s);
00191     result[e-s+1] = '\0';
00192     return result;
00193 }
00194
00195 void truncate(char *str, int len){
00196     char *s = str;
00197     char *e = str + strlen(str)-1;
00198     while (*s != '\0')
00199     {
00200         if (*s == ' ' || *s == '\t') s++;
00201     }
00202     while (*e != '\0')
00203     {
00204         *e = '\0';
00205         if (*e == ' ' || *e == '\t' || *e == '\n' || *e == '\r') e--;
00206     }
00207     if (len < 0) strcpy(str,s);
00208     else strncpy(str, s, len);
00209 }
00210
00211 int isValidIdentifier(const char* str) {
00212     // Check if the string is empty
00213     if (strlen(str) == 0) {
00214         return 0;
00215     }
00216
00217     // Check if the first character is a letter or underscore
00218     if (!isalpha(str[0]) && str[0] != '_') {
00219         return 0;
00220     }
00221
00222     // Check the remaining characters
00223     for (size_t i = 1; i < strlen(str); i++) {
00224         // Check if the character is a letter, digit, or underscore
00225         if (!isalnum(str[i]) && str[i] != '_') {
00226             return 0;
00227         }
00228     }
00229
00230     // Check if the string is a reserved keyword
00231     // Add more keywords as needed
00232     const char* keywords[] = {
00233             "auto", "break", "case", "char", "const", "continue", "default",
00234             "do", "double", "else", "enum", "extern", "float", "for", "goto",
00235             "if", "int", "long", "register", "return", "short", "signed",
00236             "sizeof", "static", "struct", "switch", "typedef", "union",
00237             "unsigned", "void", "volatile", "while", NULL
00238     };
00239
00240     for (int i = 0; keywords[i] != NULL; i++) {
00241         if (strcmp(str, keywords[i]) == 0) {
00242             return 0;
00243         }
00244     }
00245     // The string is a valid identifier
00246     return 1;
00247 }
00248
00249 void extractFunctionCallParameters(const char* str) {
00250     // Check if the string starts with a valid C identifier followed by an opening parenthesis
00251     int len = strlen(str);
00252     if (len > 0 && isalpha(str[0]) && isValidIdentifier(str)) {
00253         int i = 1;
00254         while (i < len && str[i] != '(') {
00255             i++;
00256         }
00257         if (i < len && str[i] == '(') {
00258             i++;
00259             int start = i;
00260             int depth = 1;
00261             while (i < len && depth > 0) {
00262                 if (str[i] == '(') {
00263                     depth++;
00264                 } else if (str[i] == ')') {
00265                     depth--;
00266                 }
00267                 i++;
00268             }
00269             if (depth == 0) {
00270                 // Extract the function call parameters
00271                 char parameters[100];  // Adjust the size as per your requirements
00272                 strncpy(parameters, str + start, i - start - 1);
```

```
00273                 parameters[i - start - 1] = '\0';
00274
00275                 printf("Function call parameters: %s\n", parameters);
00276                 return;
00277             }
00278         }
00279     }
00280 }
00281
00282 int lastOccurence(char *str, char c) {
00283     int i = strlen(str) - 1;
00284     while (i >= 0 && str[i] != c) {
00285         i--;
00286     }
00287     return i;
00288 }
00289
00290 node_t *create_node(context_e type) {
00291     node_t *n = malloc(sizeof(node_t));
00292     n->type = type;
00293     n->nextList = NULL;
00294     n->name[0] = '\0';
00295     return n;
00296 }
```

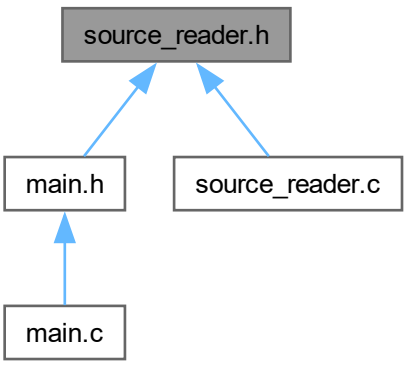
## 5.15 source_reader.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>
#include "ini_reader.h"
#include "debugmalloc.h"
```

Include dependency graph for source_reader.h:




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



### Data Structures

- struct func_type_t
- struct struct_type_t
- struct variable_type_t
- struct conditional_type_t
- struct loop_type_t
- struct node

  *linked list structure*

### Typedefs

- typedef struct node node_t

  *linked list structure*

### Functions

- node_t * read_source (char *filename)

  *NOT FULLY IMPLEMENTED YET.*
- char * substr (char *str, char start, char end)

  *Returns a substirng from the first appearance of start until the first appearance of end.*
- void truncate (char *str, int len)

  *Truncates a string from spaces and endlines*
  *if len < 0, string length doesnt change*
  *if len > 0, truncates to len.*
- int isValidIdentifier (const char *str)
- int lastOccurence (char *str, char c)
- node_t * create_node (context_e type)

## 5.15.1 Typedef Documentation

### 5.15.1.1 node_t

```
typedef struct node node_t
```

linked list structure

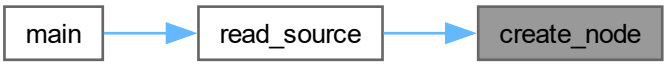
## 5.15.2 Function Documentation

### 5.15.2.1 create_node()

```
node_t * create_node (
            context_e type )
```

Definition at line 290 of file source_reader.c.

Here is the caller graph for this function:



#### 5.15.2.2 isValidIdentifier()

```
int isValidIdentifier (
            const char * str )
```

Definition at line 211 of file source_reader.c.

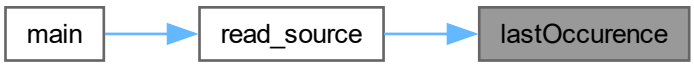
Here is the caller graph for this function:


#### 5.15.2.3 lastOccurence()

```
int lastOccurence (
            char * str,
            char c )
```

Definition at line 282 of file source_reader.c.

Here is the caller graph for this function:




#### 5.15.2.4 read_source()

```
node_t * read_source (
            char * filename )
```

NOT FULLY IMPLEMENTED YET.

**Parameters**

| | |
|---|---|
| *source_file* | to read |

**Returns**

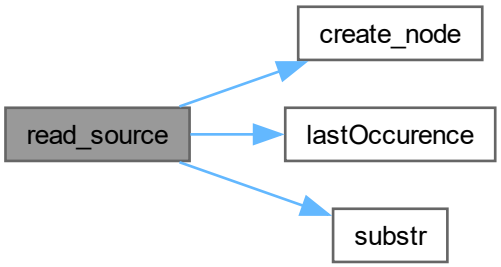
linked_list of all the lines

is full struct

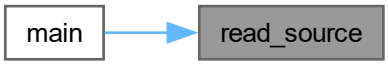
is function

Definition at line 8 of file source_reader.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.15.2.5 substr()

```
char * substr (
            char * str,
            char start,
            char end )
```

Returns a substirng from the first appearance of start until the first appearance of end.

**Parameters**

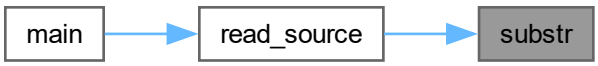
| | |
|---|---|
| *str* | string to get substring from |
| *start* | starting char |
| *end* | ending char |

**Returns**

Substring from start char (inclusive) to end char (non inclusive)

Definition at line 181 of file source_reader.c.

Here is the caller graph for this function:




#### 5.15.2.6 truncate()

```
void truncate (
            char * str,
            int len )
```

Truncates a string from spaces and endlines
if len $<$ 0, string length doesnt change
if len $>$ 0, truncates to len.

**Parameters**

| | |
|---|---|
| *str* | string to get truncated |
| *len* | truncation length |

Definition at line 195 of file source_reader.c.

## 5.16 source_reader.h

Go to the documentation of this file.

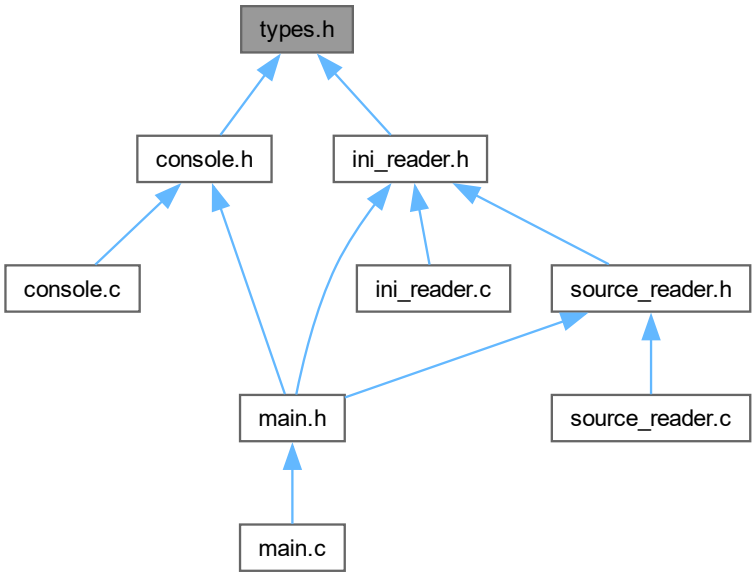
```
00001 //
00002 // Created by sziha on 21/10/2023.
00003 //
00004
00005 #ifndef NHF_SOURCE_READER_H
00006 #define NHF_SOURCE_READER_H
00007 #include <stdio.h>
00008 #include <stdlib.h>
00009 #include <regex.h>
00010 #include "ini_reader.h"
00011 #include "debugmalloc.h"
00012
00013 typedef struct {
00014     char *return_type;
00015     char **args;
00016 } func_type_t;
00017
00018 typedef struct {
00019     char *args;
```

```
00020 } struct_type_t;
00021
00022 typedef struct {
00023     char *value;
00024 } variable_type_t;
00025
00026 typedef struct {
00027     char *condition;
00028 } conditional_type_t;
00029
00030 typedef struct {
00031     char *condition;
00032 } loop_type_t;
00036 typedef struct node{
00037     context_e type;
00038     char name[100];
00039     union {
00040         func_type_t func;
00041         struct_type_t struct_;
00042         variable_type_t variable;
00043         conditional_type_t conditional;
00044         loop_type_t loop;
00045     };
00046     struct node **nextList;
00047 } node_t;
00053 node_t *read_source(char *filename);
00061 char* substr(char *str, char start, char end);
00062
00070 void truncate(char *str, int len);
00071
00072
00073 int isValidIdentifier(const char* str);
00074
00075 int lastOccurence(char *str, char c);
00076 node_t *create_node(context_e type);
00077 #endif //NHF_SOURCE_READER_H
```

## 5.17 Specification.md File Reference

## 5.18 types.h File Reference

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



### Data Structures

- struct mapping_t

  *hash-map like struct for mapping strings to anything (not very safe)*

### Macros

- #define DEFAULT_FILE_TYPE file_type_png

### Enumerations

- enum file_type_e { file_type_h , file_type_c , file_type_jpg , file_type_png }

  *Input file enum.*

## 5.18.1 Macro Definition Documentation

#### 5.18.1.1 DEFAULT_FILE_TYPE

```
#define DEFAULT_FILE_TYPE file_type_png
```

Definition at line 26 of file types.h.

## 5.18.2 Enumeration Type Documentation

#### 5.18.2.1 file_type_e

```
enum file_type_e
```

Input file enum.

**Enumerator**

| | |
|---|---|
| file_type_h | |
| file_type_c | header file |
| file_type_jpg | c file |
| file_type_png | jpg file |

Definition at line 10 of file types.h.

# 5.19 types.h

Go to the documentation of this file.

```
00001 //
00002 // Created by sziha on 16/10/2023.
00003 //
00004
```

```
00005 #ifndef NHF_TYPES_H
00006 #define NHF_TYPES_H
00010 typedef enum {
00011     file_type_h,
00012     file_type_c,
00013     file_type_jpg,
00014     file_type_png,
00015     //file_type_md, /// <markdown file
00016 } file_type_e;
00017
00021 typedef struct {
00022     const char *key;
00023     const void *value;
00024 } mapping_t;
00025
00026 #define DEFAULT_FILE_TYPE file_type_png
00027
00028 #endif//NHF_TYPES_H
```

# Index