

My Project

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 configEntry Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 keyName	5
3.1.2.2 keyValue	5
3.1.2.3 sectionName	5
4 File Documentation	7
4.1 config.c File Reference	7
4.1.1 Function Documentation	7
4.1.1.1 checkSection()	7
4.1.1.2 getNameValuePair()	8
4.1.1.3 getStrAtPos()	9
4.1.1.4 parseConfig()	9
4.2 config.c	9
4.3 config.h File Reference	12
4.3.1 Macro Definition Documentation	13
4.3.1.1 ERR_PARSECONFIG_UNKNOWN	13
4.3.1.2 ERROR_DELIMITER_NOT_FOUND	13
4.3.1.3 ERROR_MAX_LEN	13
4.3.1.4 ERROR_STR	13
4.3.1.5 FOUND_SECTION	13
4.3.1.6 MAX_LEN_SECTIONNAME	14
4.3.1.7 NO_ERROR	14
4.3.1.8 NO_SECTION	14
4.3.1.9 ST_ERROR_GETSTRATPOS	14
4.3.1.10 ST_ERROR_NOT_FOUND_LEFT_DELIMITER	14
4.3.1.11 ST_ERROR_NOT_FOUND_RIGHT_DELIMITER	14
4.3.1.12 ST_FINISH	14
4.3.1.13 ST_FOUND_LEFT_DELIMITER	14
4.3.1.14 ST_FOUND_RIGHT_DELIMITER	15
4.3.1.15 ST_FOUND_SECTION	15
4.3.1.16 ST_INIT	15
4.3.1.17 ST_SKIP_READ	15
4.3.2 Function Documentation	15
4.3.2.1 checkSection()	15

4.3.2.2	getNameValuePair()	16
4.3.2.3	getStrAtPos()	16
4.3.2.4	loadConfig()	17
4.3.2.5	parseConfig()	17
4.4	config.h	17
4.5	file.c File Reference	18
4.5.1	Function Documentation	18
4.5.1.1	getFile()	18
4.6	file.c	18
4.7	file.h File Reference	19
4.7.1	Macro Definition Documentation	19
4.7.1.1	FILE_ERROR_OPEN	19
4.7.1.2	FILE_ERROR_READ_MISMATCH	20
4.7.1.3	FILE_ERROR_STRCONTENT_TO_SMALL	20
4.7.1.4	NO_ERROR	20
4.7.2	Function Documentation	20
4.7.2.1	getFile()	20
4.8	file.h	20
4.9	test.c File Reference	20
4.9.1	Function Documentation	21
4.9.1.1	main()	21
4.10	test.c	21
Index		23

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

configEntry	5
---------------------------------------	---

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

config.c	7
config.h	12
file.c	18
file.h	19
test.c	20

Chapter 3

Class Documentation

3.1 configEntry Struct Reference

```
#include <config.h>
```

Public Attributes

- char * [sectionName](#)
- char * [keyName](#)
- char * [keyValue](#)

3.1.1 Detailed Description

Definition at line [30](#) of file [config.h](#).

3.1.2 Member Data Documentation

3.1.2.1 keyName

```
char* configEntry::keyName
```

Definition at line [33](#) of file [config.h](#).

3.1.2.2 keyValue

```
char* configEntry::keyValue
```

Definition at line [34](#) of file [config.h](#).

3.1.2.3 sectionName

```
char* configEntry::sectionName
```

Definition at line [32](#) of file [config.h](#).

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

- [config.h](#)

Chapter 4

File Documentation

4.1 config.c File Reference

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
```

Functions

- int [checkSection](#) (char *str, char delimiterLeft, char delimiterRight, char **sectionName)
Here we check if the given string contains a section for example [SECTIONName] here delimiterLeft is [and delimiterRight is] if a section is found, the section name will be written to sectionName.
- int [getStrAtPos](#) (char *str, int fromPos, int toPos, char **name, int sizeName)
- int [getNameValuePair](#) (char *str, char leftDelimiterPos, char rightDelimiterPos, char **name, char **value, int sizeName, int sizeValue)
- int [parseConfig](#) (char *originalBuffer, struct [configEntry](#) **entry, int configSizeCount, int *returnedCount)

4.1.1 Function Documentation

4.1.1.1 checkSection()

```
int checkSection (
    char * str,
    char delimiterLeft,
    char delimiterRight,
    char ** sectionName)
```

Here we check if the given string contains a section for example [SECTIONName] here delimiterLeft is [and delimiterRight is] if a section is found, the section name will be written to sectionName.

Input:

Parameters

<i>char</i>	**str: the string to check
<i>char</i>	delimiterLeft: the left delimiter to check for
<i>char</i>	delimiterRight: the right delimiter to check for

Output:

Parameters

<i>char</i>	**sectionName: if found the section Name of the section
-------------	---

Return:

Parameters

<i>int</i>	status_code: one of the following values:
------------	---

Definition at line 24 of file [config.c](#).

4.1.1.2 getNameValuePair()

```
int getNameValuePair (
    char * str,
    char leftDelimiterPos,
    char rightDelimiterPos,
    char ** name,
    char ** value,
    int sizeName,
    int sizeValue)
```

Here we get a "pair" of data, parsed from *str, witch consists of a keyname and a keyvalue. These are written to the pointers at char **name and char **value. These pair can be in the form of: ex1: NAME=VALUE ^ ^ | Here we have no delimiter this means rightDelimiterPos must be NULL This is the left delimiter ex2: name(value) ^ ^ | This is the rightDelimiterPos and must be ')' This is leftDelimiterPos which must be '('

Input: char *str: the line in the form of a string where a key value pair is stored char leftDelimiterPos: the left delimiter for example '=' char rightDelimiterPos: the right delimiter can be NULL in most cases, if NULL we assume that there is only one delimiter, which is in this case the leftDelimiterPos Output: char **name: The address where we store the name of the Key char **value: The address where we store the key value int sizeName: for size checking against memory allocated at **name int sizeValue: for size checking against memory allocated at **value

Return: will return NO_ERROR (0) if successfull. If not it will return the error of the subroutine

Definition at line 151 of file [config.c](#).

4.1.1.3 getStrAtPos()

```
int getStrAtPos (
    char * str,
    int fromPos,
    int toPos,
    char ** name,
    int sizeName)
```

Here we get / cut from fromPos to toPos and write it to an address

Input: char *str: the string to cut int fromPos: from which position we want to copy, the pos is included and zero indexed int toPos: to which position we want to copy, the pos is included and zero indexed Output: char **name: here we will write the section name to Input: int sizeName: the size of the user allocated buffer at **name

Definition at line 98 of file [config.c](#).

4.1.1.4 parseConfig()

```
int parseConfig (
    char * originalBuffer,
    struct configEntry ** entry,
    int configSizeCount,
    int * returnedCount)
```

Given a null terminated buffer of content(ex. from a file) an parse it with a default syntax of: Sections -> [SECTIONNAME] name / value -> name=value

Input: char *originalBuffer: the buffer which holds the string for parsing. struct [configEntry](#) **entry: a user allocated buffer to which we write the parsed config data. int configSizeCount: the user allocated size of **entry. Output: int *returnedCount: to this variable the function writes the count of struct, which will be required to store all available data.

Note: The function returns the maximum allowed data structures, determined by configSizeCount. All data which exceeds will be dropped. To get the whole data we run the function a second time with the reallocated configSizeCount in the size of [returnedCount] structures.

Definition at line 215 of file [config.c](#).

4.2 config.c

[Go to the documentation of this file.](#)

```
00001 #include"config.h"
00002 #include<stdio.h>
00003 #include<stdlib.h>
00004 #include<string.h>
00005 #include<errno.h>
00006
00024 int checkSection(char *str,char delimiterLeft,char delimiterRight,char **sectionName)
00025 {
00026     char section[MAX_LEN_SECTIONNAME];
00027     int i = 0; //most outer loop -> character of string
00028     if(str == NULL)
00029     {
00030         return ERROR_STR;
00031     }
00032     int len = strlen(str);
00033     int state = ST_INIT;
00034     int leftDelimiterPos=0;
```

```

00035     int rightDelimiterPos=0;
00036     char *sectionName2 = NULL;
00037     while(state != ST_FINISH)
00038     {
00039         switch(state)
00040         {
00041             case ST_INIT:
00042                 for(i=0;i<=len;i++) //find first (left) delimiter
00043                 {
00044                     if(str[i] == delimiterLeft)
00045                     {
00046                         state = ST_FOUND_LEFT_DELIMITER;
00047                         leftDelimiterPos = i;
00048                         break;
00049                     }
00050                 }
00051                 if(state != ST_FOUND_LEFT_DELIMITER)
00052                 {
00053                     return ST_ERROR_NOT_FOUND_LEFT_DELIMITER;
00054                 }
00055                 break;
00056             case ST_FOUND_LEFT_DELIMITER:
00057                 for(i=0;i<=len;i++) //find second (right) delimiter
00058                 {
00059                     if(str[i] == delimiterRight)
00060                     {
00061                         state = ST_FOUND_RIGHT_DELIMITER;
00062                         rightDelimiterPos = i;
00063                         break;
00064                     }
00065                 }
00066                 if(state != ST_FOUND_RIGHT_DELIMITER)
00067                 {
00068                     return ST_ERROR_NOT_FOUND_RIGHT_DELIMITER;
00069                 }
00070                 break;
00071             case ST_FOUND_RIGHT_DELIMITER:
00072                 int ret=0;
00073
00074                 if((ret=getStrAtPos(str,leftDelimiterPos,rightDelimiterPos,sectionName,MAX_LEN_SECTIONNAME)) ==
NO_ERROR)
00075                 {
00076                     state = ST_FINISH;
00077                 }else{
00078                     return ret;
00079                 }
00080                 case ST_FINISH:
00081                     return FOUND_SECTION;
00082             }
00083             return NO_SECTION;
00084         }
00085     }
00098 int getStrAtPos(char *str,int fromPos,int toPos,char **name,int sizeName)
00099 {
00100     if(*name == NULL)
00101     {
00102         int error = errno;
00103         printf("Pointer NULL:%d\n",error);
00104         return error;
00105     }
00106     int i=fromPos; //character iterator, which starts from specified pos
00107     int j=0; //the character iterator for the target string
00108     int diffLen=toPos-fromPos;
00109     char *ptr_name=*name;
00110     if(diffLen > MAX_LEN_SECTIONNAME || diffLen>sizeName)
00111     {
00112         return ERROR_MAX_LEN;
00113     }
00114     for(i=fromPos,j=0;i<=toPos;i++,j++)
00115     {
00116         ptr_name[j] = str[i];
00117     }
00118     ptr_name[j+1]='\0';
00119     return NO_ERROR;
00120 }
00121
00151 int getNameValuePair(char *str,char leftDelimiterPos,char rightDelimiterPos,char **name,char
**value,int sizeName,int sizeValue)
00152 {
00153     if(*name == NULL || *value == NULL)
00154     {
00155         int error = errno;
00156         printf("Pointer NULL:%d\n",error);
00157         return error;
00158     }
00159     int state=ST_INIT;

```

```

00160     char *ptr_name=*name;
00161     char *ptr_value=*value;
00162     int ret=0;
00163
00164     char *ptrDelimiter=NULL;
00165     int posDelimiter=0;
00166     int posEnd=0;
00167
00168     ptrDelimiter=strchr(str,leftDelimiterPos);
00169     if(ptrDelimiter==NULL)
00170     {
00171         return ERROR_DELIMITER_NOT_FOUND;
00172     }
00173     posDelimiter = (ptrDelimiter - str);
00174     printf("LenUntilDelimiter: %d\n",posDelimiter);
00175     if((ret=getStrAtPos(str,0,posDelimiter-1,&ptr_name,sizeName)) == NO_ERROR)
00176     {
00177         printf("ptr_name:%s\n",ptr_name);
00178     }else {
00179         printf("Error at getStrAtPos:%d\n",ret);
00180         return ret;
00181     }
00182     if(rightDelimiterPos == 0)
00183     {
00184         posEnd = strlen(str);
00185     }else{
00186         posEnd = rightDelimiterPos;
00187     }
00188
00189     if((ret=getStrAtPos(str,posDelimiter+1,posEnd,&ptr_value,sizeValue)) == NO_ERROR)
00190     {
00191         printf("ptr_value:%s\n",ptr_value);
00192     }else {
00193         return ret;
00194     }
00195
00196     return NO_ERROR;
00197 }
00198
00215 int parseConfig(char *originalBuffer,struct configEntry **entry,int configSizeCount,int
*returnedCount)
00216 {
00217     int state=ST_INIT;
00218     int ret=0;
00219     int i=0;
00220     *returnedCount=0;
00221     char *sectionName=NULL;
00222     struct configEntry *ptr_entry = *entry;
00223     char *keyName=NULL;
00224     char *keyValue=NULL;
00225     char *buffer=NULL;
00226
00227     buffer = malloc(strlen(originalBuffer)+1);
00228     if(buffer == NULL)
00229     {
00230         int error = errno;
00231         printf("ERROR MALLOC:%d",error);
00232         return ERR_PARSECONFIG_UNKNOWN;
00233     }
00234     memset(buffer,0,strlen(originalBuffer)+1);
00235     strcpy(buffer,originalBuffer);
00236
00237     //this step is necessary, because the strtok function modifies the originalBuffer
00238     //so we make a copy of it
00239     strcpy(buffer,originalBuffer);
00240
00241     printf("buffer:%s\n--",buffer);
00242
00243     sectionName = malloc(MAX_LEN_SECTIONNAME);
00244     memset(sectionName,0,MAX_LEN_SECTIONNAME);
00245
00246     //read buffer line by line
00247     char *token;
00248     token = strtok(buffer,"\n");
00249     while(token != NULL && state != ST_FINISH)
00250     {
00251
00252         switch(state)
00253         {
00254             case ST_INIT:
00255                 if((ret=checkSection(token,[' ','\n'],&sectionName))==FOUND_SECTION)
00256                 {
00257                     state = ST_FOUND_SECTION;
00258                     printf("FOUND_SECTION:%s\n",sectionName);
00259                 }
00260                 break;
00261             case ST_FOUND_SECTION:

```

```

00262         keyName = malloc(MAX_LEN_SECTIONNAME);
00263         keyValue = malloc(MAX_LEN_SECTIONNAME);
00264         if(keyName == NULL || keyValue == NULL)
00265         {
00266             int error = errno;
00267             printf("MALLOC:%d\n",error);
00268             return error;
00269         }
00270         memset(keyName,0,MAX_LEN_SECTIONNAME);
00271         memset(keyValue,0,MAX_LEN_SECTIONNAME);
00272
00273         ret=getNameValuePair(token,'=',0,&keyName,&keyValue,MAX_LEN_SECTIONNAME,MAX_LEN_SECTIONNAME);
00274         if(ret==NO_ERROR)
00275         {
00276             printf("configSizeCount: %d, i:%d\n",configSizeCount,i);
00277             if(i<configSizeCount)
00278             {
00279                 ptr_entry[i].sectionName = strdup(sectionName);
00280                 ptr_entry[i].keyValue= strdup(keyValue);
00281                 ptr_entry[i].keyName = strdup(keyName);
00282             }
00283             i++;
00284             *returnedCount = i;
00285             state = ST_FOUND_SECTION;
00286         }
00287         else {
00288             state = ST_SKIP_READ;
00289         }
00290         free(keyName);
00291         free(keyValue);
00292         break;
00293     }
00294
00295     if(state != ST_SKIP_READ)
00296     {
00297         token = strtok(NULL, "\n");
00298     }
00299     else {
00300         state = ST_INIT;
00301     }
00302 }
00303 printf("token: %s\n",token);
00304 printf("buffer:%s\n--",buffer);
00305
00306 free(sectionName);
00307 free(buffer);
00308 printf("finish exiting parsing\nSTATE:%d\n",state);
00309 return NO_ERROR;
00310 }

```

4.3 config.h File Reference

Classes

- struct [configEntry](#)

Macros

- #define [NO_SECTION](#) 0
- #define [FOUND_SECTION](#) 10
- #define [ERROR_STR](#) 1001
- #define [ERROR_MAX_LEN](#) 2
- #define [NO_ERROR](#) 0
- #define [ERROR_DELIMITER_NOT_FOUND](#) 40
- #define [ST_INIT](#) 0
- #define [ST_FOUND_LEFT_DELIMITER](#) 1
- #define [ST_FOUND_RIGHT_DELIMITER](#) 2
- #define [ST_ERROR_NOT_FOUND_RIGHT_DELIMITER](#) 3
- #define [ST_ERROR_NOT_FOUND_LEFT_DELIMITER](#) 4

- `#define ST_FOUND_SECTION 5`
- `#define ST_SKIP_READ 6`
- `#define ST_FINISH 20`
- `#define ST_ERROR_GETSTRATPOS 30`
- `#define MAX_LEN_SECTIONNAME 128`
- `#define ERR_PARSECONFIG_UNKNOWN 201`

Functions

- `int loadConfig (char *file, char **str_entry, char **host, int *intervall, int size)`
- `int checkSection (char *str, char delimiterLeft, char delimiterRight, char **sectionName)`
Here we check if the given string contains a section for example [SECTIONName] here delimiterLeft is [and delimiterRight is] if a section is found, the section name will be written to sectionName.
- `int getStrAtPos (char *str, int fromPos, int toPos, char **name, int sizeName)`
- `int getNameValuePair (char *str, char leftDelimiterPos, char rightDelimiterPos, char **name, char **value, int sizeName, int sizeValue)`
- `int parseConfig (char *buffer, struct configEntry **entry, int configSize, int *returnedCount)`

4.3.1 Macro Definition Documentation

4.3.1.1 ERR_PARSECONFIG_UNKNOWN

```
#define ERR_PARSECONFIG_UNKNOWN 201
```

Definition at line 28 of file [config.h](#).

4.3.1.2 ERROR_DELIMITER_NOT_FOUND

```
#define ERROR_DELIMITER_NOT_FOUND 40
```

Definition at line 7 of file [config.h](#).

4.3.1.3 ERROR_MAX_LEN

```
#define ERROR_MAX_LEN 2
```

Definition at line 4 of file [config.h](#).

4.3.1.4 ERROR_STR

```
#define ERROR_STR 1001
```

Definition at line 3 of file [config.h](#).

4.3.1.5 FOUND_SECTION

```
#define FOUND_SECTION 10
```

Definition at line 2 of file [config.h](#).

4.3.1.6 MAX_LEN_SECTIONNAME

```
#define MAX_LEN_SECTIONNAME 128
```

Definition at line 25 of file [config.h](#).

4.3.1.7 NO_ERROR

```
#define NO_ERROR 0
```

Definition at line 5 of file [config.h](#).

4.3.1.8 NO_SECTION

```
#define NO_SECTION 0
```

Definition at line 1 of file [config.h](#).

4.3.1.9 ST_ERROR_GETSTRATPOS

```
#define ST_ERROR_GETSTRATPOS 30
```

Definition at line 22 of file [config.h](#).

4.3.1.10 ST_ERROR_NOT_FOUND_LEFT_DELIMITER

```
#define ST_ERROR_NOT_FOUND_LEFT_DELIMITER 4
```

Definition at line 15 of file [config.h](#).

4.3.1.11 ST_ERROR_NOT_FOUND_RIGHT_DELIMITER

```
#define ST_ERROR_NOT_FOUND_RIGHT_DELIMITER 3
```

Definition at line 14 of file [config.h](#).

4.3.1.12 ST_FINISH

```
#define ST_FINISH 20
```

Definition at line 19 of file [config.h](#).

4.3.1.13 ST_FOUND_LEFT_DELIMITER

```
#define ST_FOUND_LEFT_DELIMITER 1
```

Definition at line 12 of file [config.h](#).

4.3.1.14 ST_FOUND_RIGHT_DELIMITER

```
#define ST_FOUND_RIGHT_DELIMITER 2
```

Definition at line 13 of file [config.h](#).

4.3.1.15 ST_FOUND_SECTION

```
#define ST_FOUND_SECTION 5
```

Definition at line 16 of file [config.h](#).

4.3.1.16 ST_INIT

```
#define ST_INIT 0
```

Definition at line 11 of file [config.h](#).

4.3.1.17 ST_SKIP_READ

```
#define ST_SKIP_READ 6
```

Definition at line 17 of file [config.h](#).

4.3.2 Function Documentation

4.3.2.1 checkSection()

```
int checkSection (  
    char * str,  
    char delimiterLeft,  
    char delimiterRight,  
    char ** sectionName)
```

Here we check if the given string contains a section for example [SECTIONName] here delimiterLeft is [and delimiterRight is] if a section is found, the section name will be written to sectionName.

Input:

Parameters

<i>char</i>	**str: the string to check
<i>char</i>	delimiterLeft: the left delimiter to check for
<i>char</i>	delimiterRight: the right delimiter to check for

Output:

Parameters

<i>char</i>	**sectionName : if found the section Name of the section
-------------	---

Return:

Parameters

<i>int</i>	status_code : one of the following values:
------------	---

Definition at line 24 of file [config.c](#).**4.3.2.2 getNameValuePair()**

```
int getNameValuePair (
    char * str,
    char leftDelimiterPos,
    char rightDelimiterPos,
    char ** name,
    char ** value,
    int sizeName,
    int sizeValue)
```

Here we get a "pair" of data, parsed from *str, witch consists of a keyname and a keyvalue. These are written to the pointers at char **name and char **value. These pair can be in the form of: ex1: NAME=VALUE ^ ^ | Here we have no delimiter this means rightDelimiterPos must be NULL This is the left delimiter ex2: name(value) ^ ^ | This is the rightDelimiterPos and must be ')' This is leftDelimiterPos which must be '('

Input: char *str: the line in the form of a string where a key value pair is stored char leftDelimiterPos: the left delimiter for example '=' char rightDelimiterPos: the right delimiter can be NULL in most cases, if NULL we assume that there is only one delimiter, which is in this case the leftDelimiterPos Output: char **name: The address where we store the name of the Key char **value: The address where we store the key value int sizeName: for size checking against memory allocated at **name int sizeValue: for size checking against memory allocated at **value

Return: will return NO_ERROR (0) if successfull. If not it will return the error of the subroutine

Definition at line 151 of file [config.c](#).**4.3.2.3 getStrAtPos()**

```
int getStrAtPos (
    char * str,
    int fromPos,
    int toPos,
    char ** name,
    int sizeName)
```

Here we get / cut from fromPos to toPos and write it to an address

Input: char *str: the string to cut int fromPos: from which position we want to copy, the pos is included and zero indexed int toPos: to which position we want to copy, the pos is included and zero indexed Output: char **name: here we will write the section name to Input: int sizeName: the size of the user allocated buffer at **name

Definition at line 98 of file [config.c](#).

4.3.2.4 loadConfig()

```
int loadConfig (
    char * file,
    char ** str_entry,
    char ** host,
    int * intervall,
    int size)
```

4.3.2.5 parseConfig()

```
int parseConfig (
    char * originalBuffer,
    struct configEntry ** entry,
    int configSizeCount,
    int * returnedCount)
```

Given a null terminated buffer of content(ex. from a file) an parse it with a default syntax of: Sections -> [SECTIONNAME] name / value -> name=value

Input: char *originalBuffer: the buffer which holds the string for parsing. struct [configEntry](#) **entry: a user allocated buffer to which we write the parsed config data. int configSizeCount: the user allocated size of **entry. Output: int *returnedCount: to this variable the function writes the count of struct, which will be required to store all available data.

Note: The function returns the maximum allowed data structures, determined by configSizeCount. All data which exceeds will be dropped. To get the whole data we run the function a second time with the reallocated configSizeCount in the size of [returnedCount] structures.

Definition at line 215 of file [config.c](#).

4.4 config.h

[Go to the documentation of this file.](#)

```
00001 #define NO_SECTION 0
00002 #define FOUND_SECTION 10
00003 #define ERROR_STR 1001
00004 #define ERROR_MAX_LEN 2
00005 #define NO_ERROR 0
00006 //ERROR DELIMITER
00007 #define ERROR_DELIMITER_NOT_FOUND 40
00008
00009 //State Machine
00010
00011 #define ST_INIT 0
00012 #define ST_FOUND_LEFT_DELIMITER 1
00013 #define ST_FOUND_RIGHT_DELIMITER 2
00014 #define ST_ERROR_NOT_FOUND_RIGHT_DELIMITER 3
00015 #define ST_ERROR_NOT_FOUND_LEFT_DELIMITER 4
00016 #define ST_FOUND_SECTION 5
00017 #define ST_SKIP_READ 6
00018
00019 #define ST_FINISH 20
00020
00021 //state machine ERROR
00022 #define ST_ERROR_GETSTRATPOS 30
00023
00024 //LIMITS
00025 #define MAX_LEN_SECTIONNAME 128
00026
00027 //error parseConfig
00028 #define ERR_PARSECONFIG_UNKNOWN 201
00029
```

```

00030 struct configEntry
00031 {
00032     char *sectionName;
00033     char *keyName;
00034     char *keyValue;
00035 };
00036
00037
00038 int loadConfig(char *file, char **str_entry, char **host, int *intervall, int size);
00039 int checkSection(char *str, char delimiterLeft, char delimiterRight, char **sectionName);
00040
00041 int getStrAtPos(char *str, int fromPos, int toPos, char **name, int sizeName);
00042 int getNameValuePair(char *str, char leftDelimiterPos, char rightDelimiterPos, char **name, char
    **value, int sizeName, int sizeValue);
00043
00044 int parseConfig(char *buffer, struct configEntry **entry, int configSize, int *returnedCount);

```

4.5 file.c File Reference

```

#include "file.h"
#include <stdio.h>

```

Functions

- int [getFile](#) (char *fname, char **strContent, int cbSize, long *neededSize)

4.5.1 Function Documentation

4.5.1.1 getFile()

```

int getFile (
    char * fname,
    char ** strContent,
    int cbSize,
    long * neededSize)

```

Definition at line 17 of file [file.c](#).

4.6 file.c

[Go to the documentation of this file.](#)

```

00001 #include "file.h"
00002 #include <stdio.h>
00003 /*
00004  * Here we load a file and write the file as string to a memory address:
00005  * Input:
00006  *     char* fname: the name of the file to open
00007  *     int cbSize: the size of the strContent
00008  * Output:
00009  *     char **strContent: a pointer to a address where you have to allocate enough memory to store
the whole file
00010  *     int neededSize: this returns the needed size of the buffer strContent
00011  *
00012  * Note:
00013  *     to get the size define strContent as NULL, then the needed size will return the needed
buffersize
00014  *
00015  *
00016  */

```

```

00017 int getFile(char *fname, char **strContent, int cbSize, long *neededSize)
00018 {
00019     FILE *hfile = fopen(fname, "r");
00020     long file_size=0;
00021     int ret=0;
00022     if(hfile == NULL)
00023     {
00024         return FILE_ERROR_OPEN;
00025     }
00026
00027     printf("file openend:%s", fname);
00028     fseek(hfile, 0, SEEK_END);
00029     file_size = ftell(hfile);
00030     if(strContent == NULL) //we must determine the size and return to neededSize
00031     {
00032         *neededSize = file_size+1;
00033         fclose(hfile);
00034         return NO_ERROR;
00035     }
00036
00037     if(cbSize < file_size)
00038     {
00039         *neededSize = file_size+1; //null terminator
00040         fclose(hfile);
00041         return FILE_ERROR_STRCONTENT_TO_SMALL;
00042     }
00043
00044     fseek(hfile, 0, SEEK_SET);
00045     ret=fread(*strContent, file_size, 1, hfile);
00046     if(ret != 1)
00047     {
00048         printf("Reading error read and should read mismatch\nnumber written elements:%d\n",
00049             ret);
00050         fclose(hfile);
00051         return FILE_ERROR_READ_MISMATCH;
00052     }
00053     //buffer end-1 = \0
00054     //buffer end-2 = \n
00055     //buffer end-3 = last character
00056     printf("strcontent: %ld, [%c]\n", *neededSize, *(*strContent+ *neededSize-3));
00057     *(*strContent + *neededSize -1) = '\0';
00058     printf("after zero assign: %ld, %s\n", *neededSize, (*strContent+ *neededSize-1));
00059     //printf("content:%s", *strContent);
00060     fclose(hfile);
00061     return NO_ERROR;
00062 }
00063
00064

```

4.7 file.h File Reference

Macros

- `#define NO_ERROR 0`
- `#define FILE_ERROR_OPEN 11`
- `#define FILE_ERROR_STRCONTENT_TO_SMALL 12`
- `#define FILE_ERROR_READ_MISMATCH 13`

Functions

- `int getFile (char *fname, char **strContent, int cbSize, long *neededSize)`

4.7.1 Macro Definition Documentation

4.7.1.1 FILE_ERROR_OPEN

```
#define FILE_ERROR_OPEN 11
```

Definition at line 4 of file [file.h](#).

4.7.1.2 FILE_ERROR_READ_MISMATCH

```
#define FILE_ERROR_READ_MISMATCH 13
```

Definition at line 6 of file [file.h](#).

4.7.1.3 FILE_ERROR_STRCONTENT_TO_SMALL

```
#define FILE_ERROR_STRCONTENT_TO_SMALL 12
```

Definition at line 5 of file [file.h](#).

4.7.1.4 NO_ERROR

```
#define NO_ERROR 0
```

Definition at line 1 of file [file.h](#).

4.7.2 Function Documentation

4.7.2.1 getFile()

```
int getFile (  
    char * fname,  
    char ** strContent,  
    int cbSize,  
    long * neededSize)
```

Definition at line 17 of file [file.c](#).

4.8 file.h

[Go to the documentation of this file.](#)

```
00001 #define NO_ERROR 0  
00002  
00003 //ERRORS  
00004 #define FILE_ERROR_OPEN 11  
00005 #define FILE_ERROR_STRCONTENT_TO_SMALL 12  
00006 #define FILE_ERROR_READ_MISMATCH 13  
00007  
00008 int getFile(char *fname, char **strContent,int cbSize,long *neededSize);
```

4.9 test.c File Reference

```
#include "config.h"  
#include <errno.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include "file.h"
```


Functions

- int [main](#) (void)

4.9.1 Function Documentation

4.9.1.1 main()

```
int main (
    void )
```

Definition at line 8 of file [test.c](#).

4.10 test.c

[Go to the documentation of this file.](#)

```
00001 #include "config.h"
00002 #include <errno.h>
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <string.h>
00006 #include "file.h"
00007
00008 int main(void)
00009 {
00010     char teststr[] =
00011         "sdafsd sdf12345asdf ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd";
00012     char *name = NULL;
00013     char *sectionName=NULL;
00014     int ret=0;
00015     name = malloc(MAX_LEN_SECTIONNAME*sizeof(char));
00016     getStrAtPos(teststr, 11, 15, &name, MAX_LEN_SECTIONNAME);
00017
00018     sectionName = malloc(MAX_LEN_SECTIONNAME);
00019     memset(sectionName,0,MAX_LEN_SECTIONNAME);
00020     ret=checkSection(teststr,[' ','\n'], &sectionName);
00021     if(ret == FOUND_SECTION)
00022     {
00023         printf("checkSection sucessfull\nsectionName=%s\n",sectionName);
00024     }
00025     else
00026     {
00027         free(sectionName);
00028         printf("an error ocured:%d\n",ret);
00029         return 1;
00030     }
00031
00032     char testpair[] = "asifdsfo=s1254124";
00033     char *keyName=NULL;
00034     char *keyValue=NULL;
00035     char *content=NULL;
00036
00037     keyName = malloc(MAX_LEN_SECTIONNAME);
00038     keyValue = malloc(MAX_LEN_SECTIONNAME);
00039     memset(keyName,0,MAX_LEN_SECTIONNAME);
00040     memset(keyValue,0,MAX_LEN_SECTIONNAME);
00041
00042     if((ret=getNameValuePair(testpair,
00043                             ' ',0,
00044                             &keyName,&keyValue,
00045                             MAX_LEN_SECTIONNAME,MAX_LEN_SECTIONNAME))==NO_ERROR)
00046     {
00047         printf("keyname:%s Value: %s\n",keyName,keyValue);
00048     }else{
00049         printf("error getNameValuePair: %d\n",ret);
00050     }
00051
00052     long neededSize=0;
00053     if((ret=getFile("config-segfault.cfg",NULL,0,&neededSize))==NO_ERROR)
00054     {
00055         printf("sucessfull retrieved size:%ld\n",neededSize);
00056     }
```

```

00056     else {
00057         printf("Error on getFile:%d\n",ret);
00058         return 1;
00059     }
00060     content = malloc(neededSize);
00061     printf("allocate buffer for filecontent size:%ld\n",neededSize);
00062     if(content == NULL)
00063     {
00064         int error = errno;
00065         printf("MALLOC: %d\n",error);
00066         return error;
00067     }
00068     memset(content,0,neededSize);
00069     if((ret=getFile("config-segfault.cfg",&content,neededSize,&neededSize))==NO_ERROR)
00070     {
00071         //printf("Sucessfull read file into buffer:%s|\n---\n",content);
00072     }
00073     else {
00074         printf("Error on getFile:%d\n",ret);
00075         free(content);
00076         return 1;
00077     }
00078     struct configEntry *entry=NULL;
00079     int returnedCount=0;
00080     int i=0;
00081     int count=10;
00082
00083     entry = malloc(count*sizeof(struct configEntry));
00084     if(entry == NULL)
00085     {
00086         int error = errno;
00087         printf("MALLOC: %d\n",error);
00088         return 0;
00089     }
00090     ret = parseConfig(content,&entry,count,&returnedCount);
00091     if(ret!=NO_ERROR)
00092     {
00093         printf("Error on parseConfig:%d\nReallocate from size %d, to %d\n",ret,count,returnedCount);
00094     }
00095     if(returnedCount > count)
00096     {
00097         for(i=0;i<count;i++)
00098         {
00099             free(entry[i].keyName);
00100             free(entry[i].keyValue);
00101             free(entry[i].sectionName);
00102         }
00103
00104         printf("Error on parseConfig:%d\nReallocate from size %d, to %d\n",ret,count,returnedCount);
00105         if((entry=realloc(entry,returnedCount*sizeof(struct configEntry)))==NULL)
00106         {
00107             printf("error could not reallocate from size %d to %d \n",count,returnedCount);
00108             return 1;
00109         }
00110         ret = parseConfig(content,&entry,returnedCount,&returnedCount);
00111         if(ret!=NO_ERROR)
00112         {
00113             printf("Error on parseConfig:%d\n",ret);
00114             return ret;
00115         }
00116         printf("returnedCount:%d\n",returnedCount);
00117     }
00118
00119
00120
00121     free(content);
00122
00123     for(i=0;i<returnedCount;i++)
00124     {
00125         printf("i:%d/%d, struct section: %s, keyname: %s, keyvalue:
00126 %s\n",i,returnedCount,entry[i].sectionName,entry[i].keyName,entry[i].keyValue);
00127         free(entry[i].keyName);
00128         free(entry[i].keyValue);
00129         free(entry[i].sectionName);
00130     }
00131
00132     free(keyValue);
00133     free(keyName);
00134     free(name);
00135     free(entry);
00136     free(sectionName);
00137     return 0;
00138 }
00139

```

Index

- checkSection
 - config.c, [7](#)
 - config.h, [15](#)
- config.c, [7](#)
 - checkSection, [7](#)
 - getNameValuePair, [8](#)
 - getStrAtPos, [8](#)
 - parseConfig, [9](#)
- config.h, [12](#)
 - checkSection, [15](#)
 - ERR_PARSECONFIG_UNKNOWN, [13](#)
 - ERROR_DELIMITER_NOT_FOUND, [13](#)
 - ERROR_MAX_LEN, [13](#)
 - ERROR_STR, [13](#)
 - FOUND_SECTION, [13](#)
 - getNameValuePair, [16](#)
 - getStrAtPos, [16](#)
 - loadConfig, [16](#)
 - MAX_LEN_SECTIONNAME, [13](#)
 - NO_ERROR, [14](#)
 - NO_SECTION, [14](#)
 - parseConfig, [17](#)
 - ST_ERROR_GETSTRATPOS, [14](#)
 - ST_ERROR_NOT_FOUND_LEFT_DELIMITER, [14](#)
 - ST_ERROR_NOT_FOUND_RIGHT_DELIMITER, [14](#)
 - ST_FINISH, [14](#)
 - ST_FOUND_LEFT_DELIMITER, [14](#)
 - ST_FOUND_RIGHT_DELIMITER, [14](#)
 - ST_FOUND_SECTION, [15](#)
 - ST_INIT, [15](#)
 - ST_SKIP_READ, [15](#)
- configEntry, [5](#)
 - keyName, [5](#)
 - keyValue, [5](#)
 - sectionName, [5](#)
- ERR_PARSECONFIG_UNKNOWN
 - config.h, [13](#)
- ERROR_DELIMITER_NOT_FOUND
 - config.h, [13](#)
- ERROR_MAX_LEN
 - config.h, [13](#)
- ERROR_STR
 - config.h, [13](#)
- file.c, [18](#)
 - getFile, [18](#)
- file.h, [19](#)
 - FILE_ERROR_OPEN, [19](#)
 - FILE_ERROR_READ_MISMATCH, [19](#)
 - FILE_ERROR_STRCONTENT_TO_SMALL, [20](#)
 - getFile, [20](#)
 - NO_ERROR, [20](#)
- FILE_ERROR_OPEN
 - file.h, [19](#)
- FILE_ERROR_READ_MISMATCH
 - file.h, [19](#)
- FILE_ERROR_STRCONTENT_TO_SMALL
 - file.h, [20](#)
- FOUND_SECTION
 - config.h, [13](#)
- getFile
 - file.c, [18](#)
 - file.h, [20](#)
- getNameValuePair
 - config.c, [8](#)
 - config.h, [16](#)
- getStrAtPos
 - config.c, [8](#)
 - config.h, [16](#)
- keyName
 - configEntry, [5](#)
- keyValue
 - configEntry, [5](#)
- loadConfig
 - config.h, [16](#)
- main
 - test.c, [21](#)
- MAX_LEN_SECTIONNAME
 - config.h, [13](#)
- NO_ERROR
 - config.h, [14](#)
 - file.h, [20](#)
- NO_SECTION
 - config.h, [14](#)
- parseConfig
 - config.c, [9](#)
 - config.h, [17](#)
- sectionName
 - configEntry, [5](#)
- ST_ERROR_GETSTRATPOS
 - config.h, [14](#)

ST_ERROR_NOT_FOUND_LEFT_DELIMITER
 [config.h](#), [14](#)
ST_ERROR_NOT_FOUND_RIGHT_DELIMITER
 [config.h](#), [14](#)
ST_FINISH
 [config.h](#), [14](#)
ST_FOUND_LEFT_DELIMITER
 [config.h](#), [14](#)
ST_FOUND_RIGHT_DELIMITER
 [config.h](#), [14](#)
ST_FOUND_SECTION
 [config.h](#), [15](#)
ST_INIT
 [config.h](#), [15](#)
ST_SKIP_READ
 [config.h](#), [15](#)

[test.c](#), [20](#)
 [main](#), [21](#)