FManC

0.0.1

Generated on Wed Jan 25 2023 00:30:41 for FManC by Doxygen 1.9.5

Wed Jan 25 2023 00:30:41

1 Welcome to the FManC documentation website !

Copyright

2 Todo List

Global copyFileWithoutTabAndLineBreak (char ∗sourceFilePath, char ∗pathToCopy)

Check if the path to copy has a name and an extension at the end.

Move it to code_utils.h

Global SHARED

I should change the name of "STATIC" to something like "FMANC_STATIC" to avoid any problems in case someone wants to use two libs with the same define system

3 Module Index

3.1 Modules

Here is a list of all modules:

4 File Index

4.1 File List

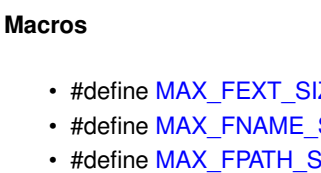
Here is a list of all files with brief descriptions:

5 Module Documentation

5.1 Constant macros

This submodule contains constant macros used by the lib.

Collaboration diagram for Constant macros:



5.1.1 Detailed Description

This submodule contains constant macros used by the lib.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.1.2 Macro Definition Documentation

5.1.2.1 MAX_FEXT_SIZE `#define MAX_FEXT_SIZE 50`

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

This the value of the maximum size of a file extension counted in characters.

Definition at line 91 of file fileMan.h.

5.1.2.2 MAX_FNAME_SIZE `#define MAX_FNAME_SIZE 256`

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

This the value of the maximum size of a file name counted in characters.

Definition at line 102 of file fileMan.h.

5.1.2.3 MAX_FPATH_SIZE `#define MAX_FPATH_SIZE 512`

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

This the value of the maximum size of a file path (full (relative or not) path without name and extension) counted in characters.

Definition at line 113 of file fileMan.h.

5.2 File management utilities

This module provides utilities to manage informations about files.

Collaboration diagram for File management utilities:

Modules

- Constant macros
  This submodule contains constant macros used by the lib.
- Functions
  This submodule contains the functions related to file management utilities.
- Macros
  This submodule contains macros related to files management facilities.
- Source files
  This submodule contains files related to file management utilities.

5.2.1 Detailed Description

This module provides utilities to manage informations about files.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.3 Functions

This submodule contains the functions related to file management utilities.

Collaboration diagram for Functions:

Functions

- char * copyFileWithoutTabAndLineBreak (char *sourceFilePath, char *pathToCopy)

  Copy a file without tab and line break.

5.3.1 Detailed Description

This submodule contains the functions related to file management utilities.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.3.2 Function Documentation

5.3.2.1 copyFileWithoutTabAndLineBreak()

```
char * copyFileWithoutTabAndLineBreak (
        char * sourceFilePath,
        char * pathToCopy )
```

Copy a file without tab and line break.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

The copied file with be renamed as <sourceFile name>_copied.<sourceFileExtension> if the param pathToCopy is set to NULL, and what you want if you specify the path with a name and extension.

Todo Check if the path to copy has a name and an extension at the end.

Move it to code_utils.h

Parameters

sourceFilePath	The source file path.
pathToCopy	The path to copy. You can set it to NULL if you want the copied file to be in the same directory as the source file.

Return values

sourceFileName	This case is when no error has occured.
NULL	If an error has occured.

Definition at line 7 of file fileMan.c.

References getFileExtension, getFileName, MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

5.4 General macros

These macros are intended to be defined if you need to modify the headers to be included, or to modify the expansion of the SHARED macro for Windows users.

Macros

- #define SHARED

  Useful to choose how to use the lib on Windows systems.

5.4.1 Detailed Description

These macros are intended to be defined if you need to modify the headers to be included, or to modify the expansion of the SHARED macro for Windows users.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.4.2 Macro Definition Documentation

5.4.2.1 SHARED `#define SHARED`

Useful to choose how to use the lib on Windows systems.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

If you want to use the lib with the dll, you don't need to add anything in the command line. If you want to use the static version of the lib, then put "-D STATIC" in your command line when compiling, so you let the compiler know that the keyword "SHARED" is set to nothing and the function declarations are not provided with the __declspec() attribute. You can also look at the full macro block below (wich is also in the source code of all of the headers) to see what I mean

```
# if (defined(_WIN32) || defined(WIN32))
#   if defined(STATIC)
#     define SHARED
#   else
#     if defined(BUILD_DLL)
#       define SHARED __declspec(dllexport)
#     else
#       define SHARED __declspec(dllimport)
#     endif
#   endif
# else
#   define SHARED
# endif // Definition of SHARED macro
```

Todo I should change the name of "STATIC" to something like "FMANC_STATIC" to avoid any problems in case someone wants to use two libs with the same define system

Definition at line 78 of file fmanc.h.

5.5 Macros

This submodule contains macros related to files management facilities.

Collaboration diagram for Macros:

Macros

- #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = ""; fgetFileExtension(source↩ FilePath, extension)

  Gives you the file extension.
- #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(source↩ FilePath, name)

  Gives you the file name.
- #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath, path)

  Gives you the file path (without name and extension).

5.5.1 Detailed Description

This submodule contains macros related to files management facilities.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.5.2 Macro Definition Documentation

5.5.2.1 getFileExtension

```
#define getFileExtension(
        sourceFilePath,
        extension ) char extension[MAX_FEXT_SIZE] = ""; fgetFileExtension(sourceFile↩
Path, extension)
```

Gives you the file extension.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

It is stored in an array of char with the name you specify (extension).

Parameters

in	*sourceFilePath*	Full path or relative path.
out	*extension*	The name of the array where you store the extension.

Returns

This doesn't really return anything, but displays an error message if no extension or invalid extension. Moreover, you should check the emptyness of the created variable.

Definition at line 137 of file fileMan.h.

5.5.2.2 getFileName

```
#define getFileName(
        sourceFilePath,
        name ) char name[MAX_FNAME_SIZE] = ""; fgetFileName(sourceFilePath, name)
```

Gives you the file name.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

It is stored in an array of char with the name you specify (name).

Parameters

in	*sourceFilePath*	Full path or relative path.
out	*name*	The name of the array where you stock the name.

Returns

This doesn't really return anything, but displays an error message if no name or invalid name. Moreover, you should check the emptyness of the created variable.

Definition at line 153 of file fileMan.h.

5.5.2.3 getFilePath

```
#define getFilePath(
        sourceFilePath,
        path ) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath, path)
```

Gives you the file path (without name and extension).

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

It is stored in an array of char with the name you specify (extension).

Parameters

in	*sourceFilePath*	Full path or relative path.
out	*path*	The name of the array where you stock the path.

Returns

This doesn't really return anything, but if the path is like "main.c", then the array will have '\0' as only character. Won't display any error message if no path. Here again, check the emptyness of the created variable.

Definition at line 169 of file fileMan.h.

5.6 Main header

This header should be included if you don't know precisely what to include.

Files

- file fmanc.h

  This is the main header of the lib, where all of the headers are included.

5.6.1 Detailed Description

This header should be included if you don't know precisely what to include.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

5.7 Source files

This submodule contains files related to file management utilities.

Collaboration diagram for Source files:

Files

- file fileMan.h

  This header contains macro definitions and function declarations that are written in this file.

5.7.1 Detailed Description

This submodule contains files related to file management utilities.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

6 File Documentation

6.1 docs/documentation_pages/main_page.dox File Reference

6.2 src/analyze.c File Reference

Include dependency graph for analyze.c:

Functions

- size_t countCharInFile (char *filePath)
- void free_stringOccurrences (stringOccurrences *toBeDeleted)
- stringOccurrences * init_stringOccurences (size_t sizeOfString)
- int replaceStringInFile (char *filePath, char *toReplaceString, char *toAddString)
- stringOccurrences * searchStringInFile (char *filePath, char *toSearch)

6.2.1 Function Documentation

6.2.1.1 countCharInFile() `size_t countCharInFile ( char * filePath )`

Definition at line 14 of file analyze.c.

Referenced by searchStringInFile().

Here is the caller graph for this function:

6.2.1.2 free_stringOccurrences() `void free_stringOccurrences ( stringOccurrences * toBeDeleted )`

Definition at line 47 of file analyze.c.

References FManC_StrOcc::pos.

Referenced by replaceStringInFile(), and searchStringInFile().

Here is the caller graph for this function:

6.2.1.3 init_stringOccurences() `stringOccurrences * init_stringOccurences ( size_t sizeOfString )`

Definition at line 35 of file analyze.c.

References FManC_StrOcc::charCount, and FManC_StrOcc::pos.

Referenced by searchStringInFile().

Here is the caller graph for this function:

6.2.1.4 replaceStringInFile()

```
int replaceStringInFile (
        char * filePath,
        char * toReplaceString,
        char * toAddString )
```

Definition at line 155 of file analyze.c.

References FManC_StrOcc::charCount, free_stringOccurrences(), getFileExtension, getFileName, getFilePath, MAX_FEXT_SIZE, MAX_FNAME_SIZE, MAX_FPATH_SIZE, FManC_StrOcc::pos, and searchStringInFile().

Here is the call graph for this function:

6.2.1.5 searchStringInFile()

```
stringOccurrences * searchStringInFile (
        char * filePath,
        char * toSearch )
```

Definition at line 54 of file analyze.c.

References countCharInFile(), free_stringOccurrences(), getFileName, init_stringOccurences(), and FManC_StrOcc::pos.

Referenced by replaceStringInFile().

Here is the call graph for this function: Here is the caller graph for this function:

6.3 analyze.c

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <errno.h>
00004 #include <string.h>
00005 #include <stdbool.h>
00006 #include <locale.h>
00007 #include <limits.h>
00008 #include <stddef.h>
00009 #include <stdint.h>
00010 #include <wchar.h>
00011 #include "analyze.h"
00012 #include "fileMan.h"
00013
00014 SHARED size_t countCharInFile(char *filePath)
00015 {
00016     errno = 0;
00017     setlocale(LC_ALL, "fr_FR.UTF8");
00018     FILE *fil = fopen(filePath, "r, ccs=UTF-8");
00019     if (fil == NULL)
00020     {
00021         fprintf(stderr, "Error :%s\n", strerror(errno));
00022         return 0;
00023     }
00024     size_t returned = 1;
00025     rewind(fil);
00026     while (fgetwc(fil) != WEOF)
00027     {
00028         returned++;
00029     }
00030
00031     fclose(fil);
00032     return returned;
00033 }
00034
00035 SHARED stringOccurrences *init_stringOccurences(size_t sizeOfString)
00036 {
00037
00038     long long int *position = (long long int *) malloc(sizeof(long long int));
00039     *position = -1;
00040     stringOccurrences *returned = (stringOccurrences*) malloc(sizeof(stringOccurrences));
00041     returned->pos = position;
00042     returned->charCount = sizeOfString;
00043
```

```
00044     return returned;
00045 }
00046
00047 SHARED void free_stringOccurrences(stringOccurrences *toBeDeleted)
00048 {
00049     free(toBeDeleted->pos);
00050     free(toBeDeleted);
00051 }
00052
00053
00054 SHARED stringOccurrences *searchStringInFile(char *filePath, char *toSearch)
00055 {
00056     errno = 0;
00057     wchar_t toSearchW[strlen(toSearch)+1];
00058
00059     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00060     {
00061         fprintf(stderr, "Error :%s\n", strerror(errno));
00062         return NULL;
00063     }
00064
00065     if (mbstowcs(toSearchW, toSearch, strlen(toSearch)) == (size_t) - 1)
00066     {
00067         fprintf(stderr, "Error :%s\n", strerror(errno));
00068         return NULL;
00069     }
00070
00071     toSearchW[strlen(toSearch)] = L'\0';
00072
00073     if (countCharInFile(filePath) > LLONG_MAX || wcslen(toSearchW) > SIZE_MAX)
00074     {
00075         getFileName(filePath, fErrorName);
00076         fprintf(stderr, "Error :  your file named \"%s\" contains too much characters\n", fErrorName);
00077         return NULL;
00078     }
00079
00080     stringOccurrences *occurencesToSearch = init_stringOccurences(wcslen(toSearchW));
00081
00082
00083     FILE *fil = fopen(filePath, "r, ccs=UTF-8");
00084     if (fil == NULL)
00085     {
00086         fprintf(stderr, "Error :%s\n", strerror(errno));
00087         free_stringOccurrences(occurencesToSearch);
00088         return NULL;
00089     }
00090     rewind(fil);
00091
00092     unsigned int cpt_occ = 0;
00093     wint_t temp[wcslen(toSearchW)+1];
00094     for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00095     {
00096         temp[i] = L'\0';
00097     }
00098     size_t cpt = 0;
00099
00100     long long int cpt2 = 0;
00101     cpt2 = ftell(fil);
00102     wint_t temp2 = fgetwc(fil);
00103     while(temp2 != WEOF)
00104     {
00105         fseek(fil, cpt2, SEEK_SET);
00106         while(cpt <= wcslen(toSearchW))
00107         {
00108             temp[cpt] = fgetwc(fil);
00109
00110             if (temp[cpt] != (wint_t)toSearchW[cpt] || temp[cpt] == WEOF)
00111             {
00112                 if (temp[cpt] == (wint_t)toSearchW[cpt])
00113                 {
00114                     cpt++;
00115                 }
00116                 break;
00117             }
00118             else
00119             {
00120                 cpt++;
00121             }
00122         }
00123
00124         if (cpt == wcslen(toSearchW))
00125         {
00126             cpt_occ++;
00127             occurencesToSearch->pos = realloc(occurencesToSearch->pos, cpt_occ*sizeof(long long));
00128             *(occurencesToSearch->pos + cpt_occ - 1) = cpt2;
00129         }
00130         cpt = 0;
```

```
00131         for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00132         {
00133             temp[i] = L'\0';
00134         }
00135         fseek(fil, cpt2, SEEK_SET);
00136         temp2 = fgetwc(fil);
00137         cpt2 = ftell(fil);
00138     }
00139     if (cpt_occ == 0)
00140     {
00141         occurencesToSearch->pos = realloc(occurencesToSearch->pos, sizeof(long long));
00142         *(occurencesToSearch->pos) = -1;
00143     }
00144     else
00145     {
00146         occurencesToSearch->pos = realloc(occurencesToSearch->pos, (cpt_occ + 1)*sizeof(long long));
00147         *(occurencesToSearch->pos + cpt_occ) = -1;
00148     }
00149 
00150     fclose(fil);
00151     return occurencesToSearch;
00152 }
00153 
00154 
00155 SHARED int replaceStringInFile(char *filePath, char *toReplaceString, char *toAddString)
00156 {
00157     stringOccurrences *toReplaceOccurrences = searchStringInFile(filePath, toReplaceString);
00158     errno = 0;
00159     wchar_t toAdd[strlen(toAddString)+1];
00160     wchar_t toReplace[strlen(toReplaceString)+1];
00161 
00162     if (toReplaceOccurrences == NULL || *(toReplaceOccurrences->pos) == -1)
00163     {
00164         return 3;
00165     }
00166 
00167     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00168     {
00169         fprintf(stderr, "Error :%s\n", strerror(errno));
00170         return -3;
00171     }
00172 
00173     if (mbstowcs(toAdd, toAddString, strlen(toAddString)) == (size_t) - 1)
00174     {
00175         fprintf(stderr, "Error :%s\n", strerror(errno));
00176         return -4;
00177     }
00178 
00179     if (mbstowcs(toReplace, toReplaceString, strlen(toReplaceString)) == (size_t) - 1)
00180     {
00181         fprintf(stderr, "Error :%s\n", strerror(errno));
00182         return -4;
00183     }
00184 
00185     toAdd[strlen(toAddString)] = L'\0';
00186     toReplace[strlen(toReplaceString)] = L'\0';
00187 
00188     FILE *filToR = fopen(filePath, "r, ccs=UTF-8");
00189     if (filToR == NULL)
00190     {
00191         fprintf(stderr, "Error :%s\n", strerror(errno));
00192         return -1;
00193     }
00194     rewind(filToR);
00195 
00196 
00197     getFilePath(filePath, sFilePath);
00198 
00199 
00200     getFileName(filePath, sFileName);
00201     if (sFileName[0] == '\0')
00202     {
00203         return -2;
00204     }
00205     getFileExtension(filePath, sFileExt);
00206     if (sFileExt[0] == '\0')
00207     {
00208         return -2;
00209     }
00210 
00211     FILE *filToW = NULL;
00212     char *replaced = "replaced";
00213     char *tempName = malloc((MAX_FNAME_SIZE + MAX_FPATH_SIZE + MAX_FEXT_SIZE)*sizeof(char));
00214     *tempName = '\0';
00215     if(sFilePath[0] != '\0')
00216     {
00217         tempName = strcat(tempName, sFilePath);
```

```
00218         tempName = strcat(tempName, replaced);
00219         tempName = strcat(tempName, sFileExt);
00220         filToW = fopen(tempName, "w+, ccs=UTF-8");
00221     }
00222     else
00223     {
00224         tempName = strcat(tempName, replaced);
00225         tempName = strcat(tempName, sFileExt);
00226         filToW = fopen(tempName, "w+, ccs=UTF-8");
00227     }
00228
00229
00230     if (filToW == NULL)
00231     {
00232         fprintf(stderr, "Error :%s\n", strerror(errno));
00233         return -1;
00234     }
00235     rewind(filToW);
00236
00237     int cpt = 0;
00238     int old_cpt = 0;
00239     wint_t temp = L'\0';
00240     wint_t temp2 = fgetwc(filToR);
00241
00242
00243     while(temp2!=WEOF)
00244     {
00245         ungetwc(temp2, filToR);
00246         while(*(toReplaceOccurrences->pos + cpt) != -1 && *(toReplaceOccurrences->pos + cpt) >= 0)
00247         {
00248             if (ftell(filToR) == *(toReplaceOccurrences->pos + cpt))
00249             {
00250                 for (size_t i = 0; i < wcslen(toAdd); ++i)
00251                 {
00252                     if(fputwc(toAdd[i], filToW) != (wint_t) toAdd[i])
00253                     {
00254                         fprintf(stderr, "ERR :%s\n", strerror(errno));
00255                         return 1;
00256                     }
00257                 }
00258                 cpt++;
00259                 for (size_t i = 0; i<toReplaceOccurrences->charCount; ++i)
00260                 {
00261                     if(fgetwc(filToR) == WEOF)
00262                         break;
00263                 }
00264             }
00265
00266             if (temp2!=WEOF && old_cpt == cpt)
00267             {
00268                 temp = fgetwc(filToR);
00269                 if(fputwc(temp, filToW) != temp)
00270                 {
00271                     fprintf(stderr, "ERR :%s\n", strerror(errno));
00272                     return 1;
00273                 }
00274             }
00275             else
00276             {
00277                 old_cpt++;
00278             }
00279         }
00280
00281         if (temp!=WEOF)
00282         {
00283             temp = fgetwc(filToR);
00284             if(fputwc(temp, filToW) != temp)
00285             {
00286                 fprintf(stderr, "ERR :%s\n", strerror(errno));
00287                 return 1;
00288             }
00289         }
00290         temp2 = fgetwc(filToR);
00291     }
00292
00293     fclose(filToR);
00294     fclose(filToW);
00295
00296     if (remove(filePath) != 0)
00297     {
00298         fprintf(stderr, "ERR :%s\n", strerror(errno));
00299         return 2;
00300     }
00301     else if (rename(tempName, filePath) != 0)
00302     {
00303         fprintf(stderr, "ERR :%s\n", strerror(errno));
00304         return 2;
```

```
00305     }
00306
00307     free(tempName);
00308     free_stringOccurrences(toReplaceOccurrences);
00309
00310     return 0;
00311 }
```

6.4 src/analyze.h File Reference

Include dependency graph for analyze.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct FManC_StrOcc

Macros

- #define SHARED

Typedefs

- typedef struct FManC_StrOcc stringOccurrences

Functions

- int copyFileWithoutStrings (const unsigned int argc, char ∗filePath,...)
- size_t countCharInFile (char ∗filePath)
- void free_stringOccurrences (stringOccurrences ∗toBeDeleted)
- stringOccurrences ∗ init_stringOccurences (size_t sizeOfString)
- int replaceStringInFile (char ∗filePath, char ∗toReplaceString, char ∗toAddString)
- stringOccurrences ∗ searchStringInFile (char ∗filePath, char ∗toSearch)

6.4.1 Data Structure Documentation

6.4.1.1 struct FManC_StrOcc Definition at line 25 of file analyze.h.

Collaboration diagram for FManC_StrOcc:

Data Fields

size_t	charCount	
long long int ∗	pos	

6.4.2 Macro Definition Documentation

6.4.2.1 SHARED `#define SHARED`

Definition at line 18 of file analyze.h.

6.4.3 Typedef Documentation

6.4.3.1 stringOccurrences `typedef struct FManC_StrOcc stringOccurrences`

Definition at line 31 of file analyze.h.

6.4.4 Function Documentation

6.4.4.1 copyFileWithoutStrings() `int copyFileWithoutStrings (`

```
const unsigned int argc,
char * filePath,
 ...  )
```

6.4.4.2 countCharInFile() `size_t countCharInFile (`

```
char * filePath )
```

Definition at line 14 of file analyze.c.

Referenced by searchStringInFile().

Here is the caller graph for this function:

6.4.4.3 free_stringOccurrences() `void free_stringOccurrences (`

```
stringOccurrences * toBeDeleted )
```

Definition at line 47 of file analyze.c.

References FManC_StrOcc::pos.

Referenced by replaceStringInFile(), and searchStringInFile().

Here is the caller graph for this function:

6.4.4.4 init_stringOccurences() `stringOccurrences * init_stringOccurences (`

```
size_t sizeOfString )
```

Definition at line 35 of file analyze.c.

References FManC_StrOcc::charCount, and FManC_StrOcc::pos.

Referenced by searchStringInFile().

Here is the caller graph for this function:

6.4.4.5 replaceStringInFile()

```
int replaceStringInFile (
        char * filePath,
        char * toReplaceString,
        char * toAddString )
```

Definition at line 155 of file analyze.c.

References FManC_StrOcc::charCount, free_stringOccurrences(), getFileExtension, getFileName, getFilePath, MAX_FEXT_SIZE, MAX_FNAME_SIZE, MAX_FPATH_SIZE, FManC_StrOcc::pos, and searchStringInFile().

Here is the call graph for this function:

6.4.4.6 searchStringInFile()

```
stringOccurrences * searchStringInFile (
        char * filePath,
        char * toSearch )
```

Definition at line 54 of file analyze.c.

References countCharInFile(), free_stringOccurrences(), getFileName, init_stringOccurences(), and FManC_StrOcc::pos.

Referenced by replaceStringInFile().

Here is the call graph for this function: Here is the caller graph for this function:

6.5 analyze.h

Go to the documentation of this file.

```
00001 #ifndef ANALYZE_H
00002 #define ANALYZE_H
00003
00004 # if (defined(_WIN32) || defined(WIN32))
00005 /***************** "-D STATIC" *****************/
00006 #   if defined(STATIC)
00007 #     define SHARED
00008 /**************** "-D BUILD_DLL" ***************/
00009 #   else
00010 #     if defined(BUILD_DLL)
00011 #       define SHARED __declspec(dllexport)
00012 #     else
00013 #       define SHARED __declspec(dllimport)
00014 #     endif
00015 #   endif
00016 /****************** DEFAULT ******************/
00017 # else
00018 #   define SHARED
00019 # endif
00020
00021
00022 #include <stddef.h>
00023
00024
00025 SHARED struct FManC_StrOcc
00026 {
00027     size_t charCount;
00028     long long int *pos;
00029 };
00030
00031 SHARED typedef struct FManC_StrOcc stringOccurrences;
00032
00033 SHARED size_t countCharInFile(char *filePath);
00034 SHARED stringOccurrences *init_stringOccurences(size_t sizeOfString);
00035 SHARED void free_stringOccurrences(stringOccurrences *toBeDeleted);
00036 SHARED stringOccurrences *searchStringInFile(char *filePath, char *toSearch);
00037 SHARED int replaceStringInFile(char *filePath, char *toReplaceString, char *toAddString);
00038 SHARED int copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00039
00040
00041 #endif
00042
```

6.6 src/code_utils.c File Reference

Include dependency graph for code_utils.c:

6.7 code_utils.c

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <errno.h>
00004 #include <string.h>
00005 #include "fileMan.h"
00006
```

6.8 src/code_utils.h File Reference

Macros

- #define SHARED

6.8.1 Macro Definition Documentation

6.8.1.1 SHARED `#define SHARED`

Definition at line 18 of file code_utils.h.

6.9 code_utils.h

Go to the documentation of this file.

```
00001 #ifndef CODE_UTILS_H
00002 #define CODE_UTILS_H
00003
00004 # if (defined(_WIN32) || defined(WIN32))
00005 /***************** "-D STATIC" ******************/
00006 #   if defined(STATIC)
00007 #     define SHARED
00008 /**************** "-D BUILD_DLL" ***************/
00009 #   else
00010 #     if defined(BUILD_DLL)
00011 #       define SHARED __declspec(dllexport)
00012 #     else
00013 #       define SHARED __declspec(dllimport)
00014 #     endif
00015 #   endif
00016 /****************** DEFAULT ******************/
00017 # else
00018 #   define SHARED
00019 # endif
00020
00021 #endif
```

6.10 src/fileMan.c File Reference

Include dependency graph for fileMan.c:

Functions

- char * copyFileWithoutTabAndLineBreak (char *sourceFilePath, char *pathToCopy)

  Copy a file without tab and line break.
- void fgetFileExtension (const char *const sourceFilePath, char *extension)
- void fgetFileName (const char *const sourceFilePath, char *fileName)
- void fgetFilePath (const char *const sourceFilePath, char *filePath)

6.10.1 Function Documentation

6.10.1.1 fgetFileExtension() `void fgetFileExtension (`

```
const char *const sourceFilePath,
char * extension )
```

Definition at line 70 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.10.1.2 fgetFileName() `void fgetFileName (`

```
const char *const sourceFilePath,
char * fileName )
```

Definition at line 118 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.10.1.3 fgetFilePath() `void fgetFilePath (`

```
const char *const sourceFilePath,
char * filePath )
```

Definition at line 191 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.11 fileMan.c

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <errno.h>
00004 #include <string.h>
00005 #include "fileMan.h"
00006
00007 SHARED char *copyFileWithoutTabAndLineBreak(char *sourceFilePath, char *pathToCopy) //not finished.
      TODO : change the return value
00008 {
00009     errno = 0;
00010     getFileName(sourceFilePath, sourceFileName);
00011     getFileExtension(sourceFilePath, sourceFileExtension);
00012
00013     FILE *sourceFile = fopen(sourceFilePath, "r");
00014
00015     if (sourceFile == NULL)
00016     {
00017         fprintf(stderr, "Error :%s\n", strerror(errno));
00018         return NULL;
00019     }
00020     rewind(sourceFile);
00021     char *copiedName = (char*) malloc((strlen(pathToCopy)+10)*sizeof(char));
00022
00023     if(copiedName == NULL) return copiedName;
00024     if (pathToCopy == NULL)
00025     {
00026         copiedName = strcat(strcat(sourceFileName,"_copied"), sourceFileExtension);
00027     }
00028     else
00029     {
00030         copiedName = strcpy(copiedName, pathToCopy);
00031     }
00032     if(copiedName == NULL) return copiedName;
00033     FILE *copiedFile = fopen(copiedName, "w");
00034     if (copiedFile == NULL)
00035     {
00036         fprintf(stderr, "Error :%s\n", strerror(errno));
00037         fclose(sourceFile);
00038         return NULL;
00039     }
00040     rewind(copiedFile);
00041
00042     while(fgetc(sourceFile) != EOF)
00043     {
00044         fseek(sourceFile, -1, SEEK_CUR);
00045         if (fgetc(sourceFile) != '\n')
00046         {
00047             fseek(sourceFile, -1, SEEK_CUR);
00048             if (fgetc(sourceFile) != '\t')
00049             {
00050                 fseek(sourceFile, -1, SEEK_CUR);
00051                 fputc(fgetc(sourceFile), copiedFile);
00052             }
00053         }
00054     }
00055     static char returnedName[MAX_FEXT_SIZE+MAX_FNAME_SIZE+MAX_FPATH_SIZE] = {'\0'}; // find a way to
      modify this
00056     int i = 0;
00057     while(sourceFileName[i] != '\0' && (size_t) i < strlen(sourceFileName))
00058     {
00059         *(returnedName + i) = sourceFileName[i];
00060         i++;
00061     }
00062     *(returnedName + i) = '\0';
00063
00064     fclose(copiedFile);
00065     fclose(sourceFile);
00066     free(copiedName);
00067     return returnedName;
00068 }
00069
00070 SHARED void fgetFileExtension(const char* const sourceFilePath, char *extension)
00071 {
00072     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00073     {
00074         fprintf(stderr, "\nError :  Full path is too big\n");
00075         return;
00076     }
00077     int cpt = strlen(sourceFilePath);
00078     char pt = *(sourceFilePath + cpt);
00079
00080
00081     while((pt != '.')  && (cpt >= 0) && (pt != '/') && (pt != '\\'))
```

```
00082     {
00083         cpt--;
00084         if (cpt>=0)
00085         {
00086             pt = *(sourceFilePath + cpt);
00087         }
00088         else break;
00089     }
00090     if (cpt < 0)
00091     {
00092         fprintf(stderr, "\nError :  incorrect file path\n");
00093         return;
00094     }
00095     else if (pt == '/' || pt == '\\')
00096     {
00097         fprintf(stderr, "\nError :  incorrect file path\n");
00098         return;
00099     }
00100
00101     else
00102     {
00103         char res[strlen(sourceFilePath)-cpt+1];
00104         for (size_t i = cpt; i < strlen(sourceFilePath); ++i)
00105         {
00106             res[i - cpt] = *(sourceFilePath + i);
00107         }
00108         res[strlen(sourceFilePath)-cpt] = '\0';
00109         for (size_t i = 0; i < strlen(res); ++i)
00110         {
00111             *(extension + i) = res[i];
00112         }
00113         *(extension + strlen(res)) = '\0';
00114
00115     }
00116 }
00117
00118 SHARED void fgetFileName(const char* const sourceFilePath, char *fileName)
00119 {
00120     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00121     {
00122         fprintf(stderr, "\nError :  Full path is too big\n");
00123         return;
00124     }
00125     int cpt = strlen(sourceFilePath);
00126     char pt = *(sourceFilePath + cpt);
00127
00128     while(cpt >= 0)
00129     {
00130         cpt--;
00131         if (cpt>=0)
00132         {
00133             pt = *(sourceFilePath + cpt);
00134         }
00135         else break;
00136         if (pt == '/' || pt == '\\' || pt == '~')
00137         {
00138             break;
00139         }
00140     }
00141     cpt++;
00142     if (cpt < 0 || (size_t) cpt == strlen(sourceFilePath))
00143     {
00144         fprintf(stderr, "\nError :  incorrect file path\n");
00145         return;
00146     }
00147     else
00148     {
00149         char res[strlen(sourceFilePath)-cpt+1];
00150         for (size_t i = cpt; i < strlen(sourceFilePath); ++i)
00151         {
00152             res[i - cpt] = *(sourceFilePath + i);
00153         }
00154         res[strlen(sourceFilePath)-cpt] = '\0';
00155         for (size_t i = 0; i < strlen(res); ++i)
00156         {
00157             *(fileName + i) = res[i];
00158         }
00159         *(fileName + strlen(res)) = '\0';
00160         cpt = strlen(fileName) - 1;
00161         char *tmp_recov = (char*) malloc((strlen(fileName)+10)*sizeof(char));
00162         int tmp_recov_char_num = strlen(fileName) + 1;
00163
00164         tmp_recov = strncpy(tmp_recov, fileName, tmp_recov_char_num);
00165         if (tmp_recov == NULL)
00166         {
00167             fprintf(stderr, "\nInternal problem into the lib\n");
00168             free(tmp_recov);
```

```
00169             return;
00170         }
00171         else
00172         {
00173             while(cpt >= 0 && fileName[cpt] != '.')
00174             {
00175                 fileName[cpt] = '\0';
00176                 cpt--;
00177             }
00178             if (cpt < 0)
00179             {
00180                 fileName = strncpy(fileName, tmp_recov, tmp_recov_char_num);
00181                 fileName[tmp_recov_char_num-1] = '\0';
00182             }
00183             else fileName[cpt] = '\0';
00184 
00185         }
00186         free(tmp_recov);
00187 
00188     }
00189 }
00190 
00191 SHARED void fgetFilePath(const char* const sourceFilePath, char *filePath)
00192 {
00193     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00194     {
00195         fprintf(stderr, "\nError :  Full path is too big\n");
00196         return;
00197     }
00198     int cpt = strlen(sourceFilePath);
00199     char pt = *(sourceFilePath + cpt); // pt = '\0'
00200 
00201     while(cpt >= 0)
00202     {
00203         cpt--;
00204         if (cpt>=0)
00205         {
00206             pt = *(sourceFilePath + cpt);
00207         }
00208         else break;
00209         if (pt == '/' || pt == '\\')
00210         {
00211             break;
00212         }
00213     }
00214     if (cpt < 0)
00215     {
00216         return;
00217     }
00218 
00219     // cpt = position in sourcefilepath of the last '/' or '\\'
00220 
00221     else
00222     {
00223         char res[cpt+1+1]; // nb of chars in the (path\{filename, ext}) + '\0' so cpt+1 due to the
    index +1 again for '\0'
00224         for (size_t i = 0; i < (size_t)cpt+1; ++i) // cpt >= 0 anyway so we can actually do this cast
    to avoid this useless gcc -Wextra warning
00225         {
00226             res[i] = *(sourceFilePath + i);
00227         }
00228         res[cpt + 1] = '\0';
00229         for (size_t i = 0; i < strlen(res); ++i)
00230         {
00231             *(filePath + i) = res[i];
00232         }
00233         if (pt == '/')
00234         {
00235             *(filePath + strlen(res)-1) = '/';
00236         }
00237         else
00238         {
00239             *(filePath + strlen(res)-1) = '\\';
00240         }
00241         *(filePath + strlen(res)) = '\0';
00242 
00243     }
00244 }
```

6.12 src/fileMan.h File Reference

This header contains macro definitions and function declarations that are written in this file.

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

Macros

- #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = ""; fgetFileExtension(source↩ FilePath, extension)

  Gives you the file extension.
- #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(source↩ FilePath, name)

  Gives you the file name.
- #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath, path)

  Gives you the file path (without name and extension).
- #define MAX_FEXT_SIZE 50
- #define MAX_FNAME_SIZE 256
- #define MAX_FPATH_SIZE 512
- #define SHARED

  Useful to choose how to use the lib on Windows systems.

Functions

- int copyFileWithoutStrings (const unsigned int argc, char ∗filePath,...)
- char ∗ copyFileWithoutTabAndLineBreak (char ∗sourceFilePath, char ∗pathToCopy)

  Copy a file without tab and line break.
- void fgetFileExtension (const char ∗const sourceFileName, char ∗extension)
- void fgetFileName (const char ∗const sourceFilePath, char ∗fileName)
- void fgetFilePath (const char ∗const sourceFilePath, char ∗filePath)

6.12.1 Detailed Description

This header contains macro definitions and function declarations that are written in this file.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

These functions are made to operate simple operation on files or file names, when there is no need to analyze something like occurrences,

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

Definition in file fileMan.h.

6.12.2 Macro Definition Documentation

6.12.2.1 SHARED `#define SHARED`

Useful to choose how to use the lib on Windows systems.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

For more informations, please see the definition of the SHARED macro in the main header

Definition at line 71 of file fileMan.h.

6.12.3 Function Documentation

6.12.3.1 copyFileWithoutStrings() `int copyFileWithoutStrings (`

```
const unsigned int argc,
char * filePath,
 ...  )
```

6.12.3.2 fgetFileExtension() `void fgetFileExtension (`

```
const char *const sourceFileName,
char * extension )
```

Definition at line 70 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.12.3.3 fgetFileName() `void fgetFileName (`

```
const char *const sourceFilePath,
char * fileName )
```

Definition at line 118 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.12.3.4 fgetFilePath() void fgetFilePath (const char *const *sourceFilePath*, char * *filePath*)

Definition at line 191 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

6.13 fileMan.h

Go to the documentation of this file.

```
00001
00046 #ifndef FILEMAN_H
00047 #define FILEMAN_H
00048
00057 # if (defined(_WIN32) || defined(WIN32))
00058 /***************** "-D STATIC" ******************/
00059 #   if defined(STATIC)
00060 #     define SHARED
00061 /***************** "-D BUILD_DLL" ***************/
00062 #   else
00063 #     if defined(BUILD_DLL)
00064 #       define SHARED __declspec(dllexport)
00065 #     else
00066 #       define SHARED __declspec(dllimport)
00067 #     endif
00068 #   endif
00069 /****************** DEFAULT *******************/
00070 # else
00071 #   define SHARED
00072 # endif
00073
00090 #ifndef MAX_FEXT_SIZE
00091 #define MAX_FEXT_SIZE 50
00092 #endif
00093
00101 #ifndef MAX_FNAME_SIZE
00102 #define MAX_FNAME_SIZE 256
00103 #endif
00104
00112 #ifndef MAX_FPATH_SIZE
00113 #define MAX_FPATH_SIZE 512
00114 #endif
00115
00116
00136 #ifndef getFileExtension
00137 #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = "";
      fgetFileExtension(sourceFilePath, extension)
00138 #endif
00139
00140
00152 #ifndef getFileName
00153 #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(sourceFilePath,
      name)
00154 #endif
00155
00156
00168 #ifndef getFilePath
00169 #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath,
      path)
00170 #endif
00171
00172
00197 SHARED char *copyFileWithoutTabAndLineBreak(char *sourceFilePath, char *pathToCopy); // copied file
      will be named like <sourceFile name>_copied
00198 SHARED int copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00199 SHARED void fgetFileExtension(const char* const sourceFileName, char *extension);
00200 SHARED void fgetFileName(const char* const sourceFilePath, char *fileName);
00201 SHARED void fgetFilePath(const char* const sourceFilePath, char *filePath);
00202
00203
00204 #endif
```

6.14 src/fmanc.h File Reference

This is the main header of the lib, where all of the headers are included.

Include dependency graph for fmanc.h:

Macros

- #define SHARED

  Useful to choose how to use the lib on Windows systems.

6.14.1 Detailed Description

This is the main header of the lib, where all of the headers are included.

Author

Axel PASCON (a.k.a. brvtalcake)

Date

2022

If you don't want to have troubles, just include this one instead of including the others one by one.

Definition in file fmanc.h.

6.15 fmanc.h

Go to the documentation of this file.

```
00001
00026 #ifndef FMANC_H
00027 #define FMANC_H
00028
00064 # if (defined(_WIN32) || defined(WIN32))
00065 /***************** "-D STATIC" ******************/
00066 #   if defined(STATIC)
00067 #     define SHARED
00068 /***************** "-D BUILD_DLL" ***************/
00069 #   else
00070 #     if defined(BUILD_DLL)
00071 #       define SHARED __declspec(dllexport)
00072 #     else
00073 #       define SHARED __declspec(dllimport)
00074 #     endif
00075 #   endif
00076 /****************** DEFAULT ******************/
00077 # else
00078 #   define SHARED
00079 # endif // Definition of SHARED macro
00080
00081
00082 #include "fileMan.h"
00083 #include "analyze.h"
00084
00102 /*
00103 #define USE_CODE_UTILS // just to make doxygen generate the doc
00104 #undef USE_CODE_UTILS
00105 */
00106
00107 #if defined(USE_CODE_UTILS)
00108 #include "code_utils.h"
00109 #include "./third_party/lex_yy.h"
00110 #endif // USE_CODE_UTILS
00111
00112
00113 #endif // fmanc.h
```

6.16 src/third_party/lex_yy.h File Reference

Macros

- #define SHARED

Functions

- int deleteCStyleComments (char *filePath)

6.16.1 Macro Definition Documentation

6.16.1.1 SHARED `#define SHARED`

Definition at line 18 of file lex_yy.h.

6.16.2 Function Documentation

6.16.2.1 deleteCStyleComments() `int deleteCStyleComments ( char * filePath )`

6.17 lex_yy.h

Go to the documentation of this file.

```
00001 #ifndef LEX_YY_H
00002 #define LEX_YY_H
00003
00004 # if (defined(_WIN32) || defined(WIN32))
00005 /***************** "-D STATIC" ******************/
00006 #   if defined(STATIC)
00007 #     define SHARED
00008 /**************** "-D BUILD_DLL" ***************/
00009 #   else
00010 #     if defined(BUILD_DLL)
00011 #       define SHARED __declspec(dllexport)
00012 #     else
00013 #       define SHARED __declspec(dllimport)
00014 #     endif
00015 #   endif
00016 /****************** DEFAULT ******************/
00017 # else
00018 #   define SHARED
00019 # endif
00020
00021 SHARED int deleteCStyleComments(char *filePath);
00022
00023
00024 #endif
```

Index