FManC

1.0.0

Generated on Mon Jan 16 2023 03:09:40 for FManC by Doxygen 1.9.5

Mon Jan 16 2023 03:09:40

1 File Index

1.1 File List

Here is a list of all files with brief descriptions:

2 File Documentation

2.1 src/analyze.c File Reference

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)

2.1.1 Function Documentation

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

Definition at line 14 of file analyze.c.

Referenced by searchStringInFile().

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

Definition at line 47 of file analyze.c.

References FMANC_SO::pos.

Referenced by replaceStringInFile(), and searchStringInFile().

2.1.1.3 init_StringOccurences()

```
stringOccurrences * init_StringOccurences (
        size_t sizeOfString )
```

Definition at line 35 of file analyze.c.

References FMANC_SO::charCount, and FMANC_SO::pos.

Referenced by searchStringInFile().

2.1.1.4 replaceStringInFile()

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

Definition at line 155 of file analyze.c.

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

2.1.1.5 searchStringInFile()

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

Definition at line 54 of file analyze.c.

References countCharInFile(), free_stringOccurrences(), getFileName, init_StringOccurences(), and FMANC_SO::pos.

Referenced by replaceStringInFile().

2.2 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 -1;
00023     }
00024     size_t returned = 0;
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 = malloc(sizeof(long long int));
00039     *position = -1;
00040     stringOccurrences *returned = 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     wchar_t temp[wcslen(toSearchW)+1];
00094     for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00095     {
00096         temp[i] = '\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] != toSearchW[cpt] || temp[cpt] == WEOF)
00111             {
00112                 if (temp[cpt] == 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] = '\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     wchar_t temp = L'\0';
00240     wchar_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) != 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 }
```

2.3 src/analyze.h File Reference

Data Structures

- struct FMANC_SO

Macros

- #define SHARED

Typedefs

- typedef struct FMANC_SO stringOccurrences

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)

2.3.1 Data Structure Documentation

2.3.1.1 struct FMANC_SO Definition at line 25 of file analyze.h.

Data Fields

size_t	charCount	
long long int *	pos	

2.3.2 Macro Definition Documentation

2.3.2.1 SHARED `#define SHARED`

Definition at line 18 of file analyze.h.

2.3.3 Typedef Documentation

2.3.3.1 stringOccurrences `typedef struct FMANC_SO stringOccurrences`

Definition at line 31 of file analyze.h.

2.3.4 Function Documentation

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

Definition at line 14 of file analyze.c.

Referenced by searchStringInFile().

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

Definition at line 47 of file analyze.c.

References FMANC_SO::pos.

Referenced by replaceStringInFile(), and searchStringInFile().

2.3.4.3 init_StringOccurences() `stringOccurrences * init_StringOccurences ( size_t sizeOfString )`

Definition at line 35 of file analyze.c.

References FMANC_SO::charCount, and FMANC_SO::pos.

Referenced by searchStringInFile().

2.3.4.4 replaceStringInFile()

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

Definition at line 155 of file analyze.c.

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

2.3.4.5 searchStringInFile()

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

Definition at line 54 of file analyze.c.

References countCharInFile(), free_stringOccurrences(), getFileName, init_StringOccurences(), and FMANC_SO::pos.

Referenced by replaceStringInFile().

2.4 analyze.h

Go to the documentation of this file.

```
00001 #ifndef ANALYZE_H
00002 #define ANALYZE_H
00003
00004 # if 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_SO
00026 {
00027     size_t charCount;
00028     long long int *pos;
00029 };
00030
00031 SHARED typedef struct FMANC_SO 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
00039
00040 #endif
00041
```

2.5 src/fcmx.c File Reference

2.6 fcmx.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
```

2.7 src/fcmx.h File Reference

Macros

- #define SHARED

Functions

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

2.7.1 Macro Definition Documentation

2.7.1.1 SHARED `#define SHARED`

Definition at line 18 of file fcmx.h.

2.7.2 Function Documentation

2.7.2.1 copyFileWithoutStrings()

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

2.8 fcmx.h

Go to the documentation of this file.

```
00001 #ifndef FCMX_H
00002 #define FCMX_H
00003
00004 # if 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 copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00022
00023 #endif
```

2.9 src/fileMan.c File Reference

Functions

- char * copyFileWithoutTabAndLineBreak (char *sourceFilePath, char **pathToCopy)
- void fgetFileExtension (char *sourceFilePath, char *extension)
- void fgetFileName (char *sourceFilePath, char *fileName)
- void fgetFilePath (char *sourceFilePath, char *filePath)

2.9.1 Function Documentation

2.9.1.1 copyFileWithoutTabAndLineBreak()

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

Definition at line 7 of file fileMan.c.

References getFileExtension, and getFileName.

2.9.1.2 fgetFileExtension()

```
void fgetFileExtension (
        char * sourceFilePath,
        char * extension )
```

Definition at line 67 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.9.1.3 fgetFileName()

```
void fgetFileName (
        char * sourceFilePath,
        char * fileName )
```

Definition at line 104 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.9.1.4 fgetFilePath()

```
void fgetFilePath (
        char * sourceFilePath,
        char * filePath )
```

Definition at line 151 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.10 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
00008 {
00009
00010     errno = 0;
00011     getFileName(sourceFilePath, sourceFileName);
00012     getFileExtension(sourceFilePath, sourceFileExtension);
00013
00014     FILE *sourceFile = fopen(sourceFilePath, "r");
00015
00016     if (sourceFile == NULL)
00017     {
00018         fprintf(stderr, "Error :%s\n", strerror(errno));
00019         return NULL;
00020     }
00021     rewind(sourceFile);
00022     char *copiedName = NULL;
00023     if (pathToCopy == NULL)
00024     {
00025         copiedName = strcat(strcat(sourceFileName,"_copied"), sourceFileExtension); //modify here
00026     }
00027     else
00028     {
00029         copiedName = *pathToCopy;
00030     }
00031
00032     FILE *copiedFile = fopen(copiedName, "w");
00033     if (copiedFile == NULL)
00034     {
00035         fprintf(stderr, "Error :%s\n", strerror(errno));
00036         fclose(sourceFile);
00037         return NULL;
00038     }
00039     rewind(copiedFile);
00040
00041     while(fgetc(sourceFile) != EOF)
00042     {
00043         fseek(sourceFile, -1, SEEK_CUR);
00044         if (fgetc(sourceFile) != '\n')
00045         {
00046             fseek(sourceFile, -1, SEEK_CUR);
00047             if (fgetc(sourceFile) != '\t')
00048             {
00049                 fseek(sourceFile, -1, SEEK_CUR);
00050                 fputc(fgetc(sourceFile), copiedFile);
00051             }
00052         }
```

```
00053     }
00054     char *returnedName = NULL;
00055     int i = 0;
00056     while(sourceFileName[i] != '\0')
00057     {
00058         *(returnedName + i) = sourceFileName[i];
00059         i++;
00060     }
00061     *(returnedName + i) = '\0';
00062     fclose(copiedFile);
00063     fclose(sourceFile);
00064     return returnedName;
00065 }
00066
00067 SHARED void fgetFileExtension(char *sourceFilePath, char *extension)
00068 {
00069     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00070     {
00071         fprintf(stderr, "\nError :  Full path is too big\n");
00072         return;
00073     }
00074     int cpt = strlen(sourceFilePath);
00075     char pt = *(sourceFilePath + cpt);
00076
00077
00078     while((pt != '.')  && (cpt >= 0))
00079     {
00080         cpt--;
00081         pt = *(sourceFilePath + cpt);
00082     }
00083     if (cpt < 0)
00084     {
00085         fprintf(stderr, "\nError :  incorrect file path\n");
00086     }
00087     else
00088     {
00089         char res[strlen(sourceFilePath)-cpt+1];
00090         for (size_t i = cpt; i < strlen(sourceFilePath); ++i)
00091         {
00092             res[i - cpt] = *(sourceFilePath + i);
00093         }
00094         res[strlen(sourceFilePath)-cpt] = '\0';
00095         for (size_t i = 0; i < strlen(res); ++i)
00096         {
00097             *(extension + i) = res[i];
00098         }
00099         *(extension + strlen(res)) = '\0';
00100
00101     }
00102 }
00103
00104 SHARED void fgetFileName(char *sourceFilePath, char *fileName)
00105 {
00106     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00107     {
00108         fprintf(stderr, "\nError :  Full path is too big\n");
00109         return;
00110     }
00111     int cpt = strlen(sourceFilePath);
00112     char pt = *(sourceFilePath + cpt);
00113
00114     while(cpt >= 0)
00115     {
00116         cpt--;
00117         pt = *(sourceFilePath + cpt);
00118         if (pt == '/' || pt == '\\')
00119         {
00120             break;
00121         }
00122     }
00123     cpt++;
00124     if (cpt < 0)
00125     {
00126         fprintf(stderr, "\nError :  incorrect file path\n");
00127     }
00128     else
00129     {
00130         char res[strlen(sourceFilePath)-cpt+1];
00131         for (size_t i = cpt; i < strlen(sourceFilePath); ++i)
00132         {
00133             res[i - cpt] = *(sourceFilePath + i);
00134         }
00135         res[strlen(sourceFilePath)-cpt] = '\0';
00136         for (size_t i = 0; i < strlen(res); ++i)
00137         {
00138             *(fileName + i) = res[i];
00139         }
```

```
00140         *(fileName + strlen(res)) = '\0';
00141         cpt = strlen(fileName) - 1;
00142         while(fileName[cpt] != '.')
00143         {
00144             fileName[cpt] = '\0';
00145             cpt--;
00146         }
00147         fileName[cpt] = '\0';
00148     }
00149 }
00150
00151 SHARED void fgetFilePath(char *sourceFilePath, char *filePath)
00152 {
00153     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00154     {
00155         fprintf(stderr, "\nError :  Full path is too big\n");
00156         return;
00157     }
00158     int cpt = strlen(sourceFilePath);
00159     char pt = *(sourceFilePath + cpt);
00160
00161     while(cpt >= 0)
00162     {
00163         cpt--;
00164         pt = *(sourceFilePath + cpt);
00165         if (pt == '/' || pt == '\\')
00166         {
00167             break;
00168         }
00169     }
00170
00171     if (cpt < 0)
00172     {
00173         return;
00174     }
00175
00176     else
00177     {
00178         char res[cpt+1];
00179         for (size_t i = 0; i < (size_t)cpt; ++i) // cpt >= 0 anyway so we can actually do this to
      avoid this useless gcc -Wextra warning
00180         {
00181             res[i] = *(sourceFilePath + i);
00182         }
00183         res[cpt + 1] = '\0';
00184         for (size_t i = 0; i < strlen(res); ++i)
00185         {
00186             *(filePath + i) = res[i];
00187         }
00188         if (pt == '/')
00189         {
00190             *(filePath + strlen(res)-1) = '/';
00191         }
00192         else
00193         {
00194             *(filePath + strlen(res)-1) = '\\';
00195         }
00196         *(filePath + strlen(res)) = '\0';
00197
00198     }
00199 }
```

2.11 src/fileMan.h File Reference

Macros

- #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = ""; fgetFileExtension(source↩ FilePath, extension)
- #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(source↩ FilePath, name)
- #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath, path)
- #define MAX_FEXT_SIZE 50
- #define MAX_FNAME_SIZE 256
- #define MAX_FPATH_SIZE 512
- #define SHARED

Functions

- int copyFileWithoutStrings (const unsigned int argc, char *filePath,...)
- char * copyFileWithoutTabAndLineBreak (char *sourceFilePath, char **pathToCopy)
- void fgetFileExtension (char *sourceFileName, char *extension)
- void fgetFileName (char *sourceFilePath, char *fileName)
- void fgetFilePath (char *sourceFilePath, char *filePath)

2.11.1 Macro Definition Documentation

2.11.1.1 getFileExtension

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

Definition at line 34 of file fileMan.h.

2.11.1.2 getFileName

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

Definition at line 38 of file fileMan.h.

2.11.1.3 getFilePath

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

Definition at line 42 of file fileMan.h.

2.11.1.4 MAX_FEXT_SIZE

```
#define MAX_FEXT_SIZE 50
```

Definition at line 22 of file fileMan.h.

2.11.1.5 MAX_FNAME_SIZE

```
#define MAX_FNAME_SIZE 256
```

Definition at line 26 of file fileMan.h.

2.11.1.6 MAX_FPATH_SIZE `#define MAX_FPATH_SIZE 512`

Definition at line 30 of file fileMan.h.

2.11.1.7 SHARED `#define SHARED`

Definition at line 18 of file fileMan.h.

2.11.2 Function Documentation

2.11.2.1 copyFileWithoutStrings() `int copyFileWithoutStrings (`

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

2.11.2.2 copyFileWithoutTabAndLineBreak() `char * copyFileWithoutTabAndLineBreak (`

```
char * sourceFilePath,
char ** pathToCopy )
```

Definition at line 7 of file fileMan.c.

References getFileExtension, and getFileName.

2.11.2.3 fgetFileExtension() `void fgetFileExtension (`

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

Definition at line 67 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.11.2.4 fgetFileName() `void fgetFileName (`

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

Definition at line 104 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.11.2.5 fgetFilePath() void fgetFilePath (char * *sourceFilePath*, char * *filePath*)

Definition at line 151 of file fileMan.c.

References MAX_FEXT_SIZE, MAX_FNAME_SIZE, and MAX_FPATH_SIZE.

2.12 fileMan.h

Go to the documentation of this file.

```
00001 #ifndef FILEMAN_H
00002 #define FILEMAN_H
00003
00004 # if 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 #ifndef MAX_FEXT_SIZE
00022 #define MAX_FEXT_SIZE 50
00023 #endif
00024
00025 #ifndef MAX_FNAME_SIZE
00026 #define MAX_FNAME_SIZE 256
00027 #endif
00028
00029 #ifndef MAX_FPATH_SIZE
00030 #define MAX_FPATH_SIZE 512
00031 #endif
00032
00033 #ifndef getFileExtension
00034 #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = "";
      fgetFileExtension(sourceFilePath, extension)
00035 #endif
00036
00037 #ifndef getFileName
00038 #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(sourceFilePath,
      name)
00039 #endif
00040
00041 #ifndef getFilePath
00042 #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath,
      path)
00043 #endif
00044
00045
00046 SHARED char *copyFileWithoutTabAndLineBreak(char *sourceFilePath, char **pathToCopy); // copied file
      will be named like <sourceFile name>_copied
00047 SHARED int copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00048 SHARED void fgetFileExtension(char *sourceFileName, char *extension);
00049 SHARED void fgetFileName(char *sourceFilePath, char *fileName);
00050 SHARED void fgetFilePath(char *sourceFilePath, char *filePath);
00051
00052
00053 #endif
```

2.13 src/fmanc.h File Reference

Macros

- #define SHARED

2.13.1 Macro Definition Documentation

2.13.1.1 SHARED `#define SHARED`

Definition at line 19 of file fmanc.h.

2.14 fmanc.h

Go to the documentation of this file.

```
00001 #ifndef FMANC_H
00002 #define FMANC_H
00003
00004
00005 # if defined(_WIN32)
00006 /***************** "-D STATIC" ******************/
00007 #   if defined(STATIC)
00008 #     define SHARED
00009 /***************** "-D BUILD_DLL" ***************/
00010 #   else
00011 #     if defined(BUILD_DLL)
00012 #       define SHARED __declspec(dllexport)
00013 #     else
00014 #       define SHARED __declspec(dllimport)
00015 #     endif
00016 #   endif
00017 /****************** DEFAULT *******************/
00018 # else
00019 #   define SHARED
00020 # endif
00021
00022
00023 #include "fileMan.h"
00024 #include "analyze.h"
00025
00026 #if defined(USE_FCMX)
00027 #include "fcmx.h"
00028 #include "./third_party/lex_yy.h"
00029 #endif
00030
00031
00032
00033
00034
00035
00036
00037 #endif
```

2.15 src/third_party/lex_yy.h File Reference

Macros

- #define SHARED

Functions

- int deleteCStyleComments (char *filePath)

2.15.1 Macro Definition Documentation

2.15.1.1 SHARED `#define SHARED`

Definition at line 18 of file lex_yy.h.

2.15.2 Function Documentation

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

2.16 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)
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