FManC

1.0.0

Generated on Mon Jan 16 2023 02:41:41 for FManC by Doxygen 1.9.5

Mon Jan 16 2023 02:41:41

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 50 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 38 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 163 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 57 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
00036
00037
00038 SHARED stringOccurrences *init_StringOccurences(size_t sizeOfString)
00039 {
00040
00041     long long int *position = malloc(sizeof(long long int));
00042     *position = -1;
00043     stringOccurrences *returned = malloc(sizeof(stringOccurrences));
00044     returned->pos = position;
00045     returned->charCount = sizeOfString;
00046
00047     return returned;
00048 }
00049
00050 SHARED void free_stringOccurrences(stringOccurrences *toBeDeleted)
00051 {
00052     free(toBeDeleted->pos);
00053     free(toBeDeleted);
00054 }
00055
00056
00057 SHARED stringOccurrences *searchStringInFile(char *filePath, char *toSearch)
00058 {
00059     errno = 0;
00060     wchar_t toSearchW[strlen(toSearch)+1];
00061
00062     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00063     {
00064         fprintf(stderr, "Error :%s\n", strerror(errno));
00065         return NULL;
00066     }
00067
00068
00069     if (mbstowcs(toSearchW, toSearch, strlen(toSearch)) == (size_t) - 1)
00070     {
00071         fprintf(stderr, "Error :%s\n", strerror(errno));
00072         return NULL;
00073     }
00074
00075
00076
00077     toSearchW[strlen(toSearch)] = L'\0';
00078
00079     if (countCharInFile(filePath) > LLONG_MAX || wcslen(toSearchW) > SIZE_MAX)
00080     {
00081         getFileName(filePath, fErrorName);
00082         fprintf(stderr, "Error :  your file named \"%s\" contains too much characters\n", fErrorName);
00083         return NULL;
00084     }
00085
00086
00087     stringOccurrences *occurencesToSearch = init_StringOccurences(wcslen(toSearchW));
00088
00089
00090     FILE *fil = fopen(filePath, "r, ccs=UTF-8");
00091     if (fil == NULL)
00092     {
00093         fprintf(stderr, "Error :%s\n", strerror(errno));
00094         free_stringOccurrences(occurencesToSearch);
00095         return NULL;
00096     }
00097     rewind(fil);
00098
00099
00100     unsigned int cpt_occ = 0;
00101     wchar_t temp[wcslen(toSearchW)+1];
00102     for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00103     {
00104         temp[i] = '\0';
00105     }
00106     size_t cpt = 0;
00107
00108     long long int cpt2 = 0;
00109     cpt2 = ftell(fil);
00110     wint_t temp2 = fgetwc(fil);
00111     while(temp2 != WEOF)
00112     {
00113         fseek(fil, cpt2, SEEK_SET);
00114         while(cpt <= wcslen(toSearchW))
00115         {
```

```
00116             temp[cpt] = fgetwc(fil);
00117
00118             if (temp[cpt] != toSearchW[cpt] || temp[cpt] == WEOF)
00119             {
00120                 if (temp[cpt] == toSearchW[cpt])
00121                 {
00122                     cpt++;
00123                 }
00124                 break;
00125             }
00126             else
00127             {
00128                 cpt++;
00129             }
00130         }
00131
00132         if (cpt == wcslen(toSearchW))
00133         {
00134             cpt_occ++;
00135             occurencesToSearch->pos = realloc(occurencesToSearch->pos, cpt_occ*sizeof(long long));
00136             *(occurencesToSearch->pos + cpt_occ - 1) = cpt2;
00137         }
00138         cpt = 0;
00139         for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00140         {
00141             temp[i] = '\0';
00142         }
00143         fseek(fil, cpt2, SEEK_SET);
00144         temp2 = fgetwc(fil);
00145         cpt2 = ftell(fil);
00146     }
00147     if (cpt_occ == 0)
00148     {
00149         occurencesToSearch->pos = realloc(occurencesToSearch->pos, sizeof(long long));
00150         *(occurencesToSearch->pos) = -1;
00151     }
00152     else
00153     {
00154         occurencesToSearch->pos = realloc(occurencesToSearch->pos, (cpt_occ + 1)*sizeof(long long));
00155         *(occurencesToSearch->pos + cpt_occ) = -1;
00156     }
00157
00158     fclose(fil);
00159     return occurencesToSearch;
00160 }
00161
00162
00163 SHARED int replaceStringInFile(char *filePath, char *toReplaceString, char *toAddString)
00164 {
00165     stringOccurrences *toReplaceOccurrences = searchStringInFile(filePath, toReplaceString);
00166     errno = 0;
00167     wchar_t toAdd[strlen(toAddString)+1];
00168     wchar_t toReplace[strlen(toReplaceString)+1];
00169
00170     if (toReplaceOccurrences == NULL || *(toReplaceOccurrences->pos) == -1)
00171     {
00172         return 3;
00173     }
00174
00175     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00176     {
00177         fprintf(stderr, "Error :%s\n", strerror(errno));
00178         return -3;
00179     }
00180
00181     if (mbstowcs(toAdd, toAddString, strlen(toAddString)) == (size_t) - 1)
00182     {
00183         fprintf(stderr, "Error :%s\n", strerror(errno));
00184         return -4;
00185     }
00186
00187     if (mbstowcs(toReplace, toReplaceString, strlen(toReplaceString)) == (size_t) - 1)
00188     {
00189         fprintf(stderr, "Error :%s\n", strerror(errno));
00190         return -4;
00191     }
00192
00193     toAdd[strlen(toAddString)] = L'\0';
00194     toReplace[strlen(toReplaceString)] = L'\0';
00195
00196     FILE *filToR = fopen(filePath, "r, ccs=UTF-8");
00197     if (filToR == NULL)
00198     {
00199         fprintf(stderr, "Error :%s\n", strerror(errno));
00200         return -1;
00201     }
00202     rewind(filToR);
```

```
00203
00204
00205     getFilePath(filePath, sFilePath);
00206
00207
00208     getFileName(filePath, sFileName);
00209     if (sFileName[0] == '\0')
00210     {
00211         return -2;
00212     }
00213     getFileExtension(filePath, sFileExt);
00214     if (sFileExt[0] == '\0')
00215     {
00216         return -2;
00217     }
00218
00219     FILE *filToW = NULL;
00220     char *replaced = "replaced";
00221     char *tempName = malloc((MAX_FNAME_SIZE + MAX_FPATH_SIZE + MAX_FEXT_SIZE)*sizeof(char));
00222     *tempName = '\0';
00223     if(sFilePath[0] != '\0')
00224     {
00225         tempName = strcat(tempName, sFilePath);
00226         tempName = strcat(tempName, replaced);
00227         tempName = strcat(tempName, sFileExt);
00228         filToW = fopen(tempName, "w+, ccs=UTF-8");
00229     }
00230     else
00231     {
00232         tempName = strcat(tempName, replaced);
00233         tempName = strcat(tempName, sFileExt);
00234         filToW = fopen(tempName, "w+, ccs=UTF-8");
00235     }
00236
00237
00238     if (filToW == NULL)
00239     {
00240         fprintf(stderr, "Error :%s\n", strerror(errno));
00241         return -1;
00242     }
00243     rewind(filToW);
00244
00245     int cpt = 0;
00246     int old_cpt = 0;
00247     wchar_t temp = L'\0';
00248     wchar_t temp2 = fgetwc(filToR);
00249
00250
00251     while(temp2!=WEOF)
00252     {
00253         ungetwc(temp2, filToR);
00254         while(*(toReplaceOccurrences->pos + cpt) != -1 && *(toReplaceOccurrences->pos + cpt) >= 0)
00255         {
00256             if (ftell(filToR) == *(toReplaceOccurrences->pos + cpt))
00257             {
00258                 for (size_t i = 0; i < wcslen(toAdd); ++i)
00259                 {
00260                     if(fputwc(toAdd[i], filToW) != toAdd[i])
00261                     {
00262                         fprintf(stderr, "ERR :%s\n", strerror(errno));
00263                         return 1;
00264                     }
00265                 }
00266                 cpt++;
00267                 for (size_t i = 0; i<toReplaceOccurrences->charCount; ++i)
00268                 {
00269                     if(fgetwc(filToR) == WEOF)
00270                         break;
00271                 }
00272             }
00273
00274             if (temp2!=WEOF && old_cpt == cpt)
00275             {
00276                 temp = fgetwc(filToR);
00277                 if(fputwc(temp, filToW) != temp)
00278                 {
00279                     fprintf(stderr, "ERR :%s\n", strerror(errno));
00280                     return 1;
00281                 }
00282             }
00283             else
00284             {
00285                 old_cpt++;
00286             }
00287
00288
00289         }
```

```
00290
00291         if (temp!=WEOF)
00292         {
00293             temp = fgetwc(filToR);
00294             if(fputwc(temp, filToW) != temp)
00295             {
00296                 fprintf(stderr, "ERR :%s\n", strerror(errno));
00297                 return 1;
00298             }
00299         }
00300         temp2 = fgetwc(filToR);
00301     }
00302
00303     fclose(filToR);
00304     fclose(filToW);
00305
00306
00307
00308     if (remove(filePath) != 0)
00309     {
00310         fprintf(stderr, "ERR :%s\n", strerror(errno));
00311         return 2;
00312     }
00313     else if (rename(tempName, filePath) != 0)
00314     {
00315         fprintf(stderr, "ERR :%s\n", strerror(errno));
00316         return 2;
00317     }
00318
00319
00320
00321
00322
00323
00324     free(tempName);
00325     free_stringOccurrences(toReplaceOccurrences);
00326
00327     return 0;
00328 }
```

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 50 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 38 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 163 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 57 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
00024
00025 #endif
00026
```

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 69 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 106 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 154 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
00015
00016     FILE *sourceFile = fopen(sourceFilePath, "r");
00017
00018     if (sourceFile == NULL)
00019     {
00020         fprintf(stderr, "Error :%s\n", strerror(errno));
00021         return NULL;
00022     }
00023     rewind(sourceFile);
00024     char *copiedName = NULL;
00025     if (pathToCopy == NULL)
00026     {
00027         copiedName = strcat(strcat(sourceFileName,"_copied"), sourceFileExtension); //modify here
00028     }
00029     else
00030     {
00031         copiedName = *pathToCopy;
00032     }
00033
00034     FILE *copiedFile = fopen(copiedName, "w");
00035     if (copiedFile == NULL)
00036     {
00037         fprintf(stderr, "Error :%s\n", strerror(errno));
00038         fclose(sourceFile);
00039         return NULL;
00040     }
00041     rewind(copiedFile);
00042
00043     while(fgetc(sourceFile) != EOF)
00044     {
00045         fseek(sourceFile, -1, SEEK_CUR);
00046         if (fgetc(sourceFile) != '\n')
00047         {
00048             fseek(sourceFile, -1, SEEK_CUR);
00049             if (fgetc(sourceFile) != '\t')
00050             {
00051                 fseek(sourceFile, -1, SEEK_CUR);
00052                 fputc(fgetc(sourceFile), copiedFile);
```

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

```
00140         {
00141             *(fileName + i) = res[i];
00142         }
00143         *(fileName + strlen(res)) = '\0';
00144         cpt = strlen(fileName) - 1;
00145         while(fileName[cpt] != '.')
00146         {
00147             fileName[cpt] = '\0';
00148             cpt--;
00149         }
00150         fileName[cpt] = '\0';
00151     }
00152 }
00153
00154 SHARED void fgetFilePath(char *sourceFilePath, char *filePath)
00155 {
00156     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00157     {
00158         fprintf(stderr, "\nError :  Full path is too big\n");
00159         return;
00160     }
00161     int cpt = strlen(sourceFilePath);
00162     char pt = *(sourceFilePath + cpt);
00163
00164
00165     while(cpt >= 0)
00166     {
00167         cpt--;
00168         pt = *(sourceFilePath + cpt);
00169         if (pt == '/' || pt == '\\')
00170         {
00171             break;
00172         }
00173     }
00174
00175
00176     if (cpt < 0)
00177     {
00178         return;
00179     }
00180
00181     else
00182     {
00183         char res[cpt+1];
00184         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
00185         {
00186             res[i] = *(sourceFilePath + i);
00187         }
00188         res[cpt + 1] = '\0';
00189         for (size_t i = 0; i < strlen(res); ++i)
00190         {
00191             *(filePath + i) = res[i];
00192         }
00193         if (pt == '/')
00194         {
00195             *(filePath + strlen(res)-1) = '/';
00196         }
00197         else
00198         {
00199             *(filePath + strlen(res)-1) = '\\';
00200         }
00201         *(filePath + strlen(res)) = '\0';
00202
00203     }
00204 }
```

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 69 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 106 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 154 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
00054
00055
```

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