# FManC

Generated by Doxygen 1.9.5

# Chapter 1

# NOTBYME

The "abc2.l" file was completely inspired by `this website`

# Chapter 2

# Todo List

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

Do it (lol).

**Global copyFileWithoutTabAndLineBreak (char ∗sourceFilePath, char ∗∗pathToCopy)**

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

# Chapter 3

# Module Index

## 3.1 Modules

Here is a list of all modules:

# Chapter 4

# Data Structure Index

## 4.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 6

# Module Documentation

## 6.1 Core C source code

### Files

- file analyze.c
- file fileMan.c

  *These functions are made to do operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...*

### 6.1.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.2 Core C headers

### Modules

- Core lib C headers
- Main C header

### 6.2.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.3 Core lib C headers

### Files

- file analyze.h

  *This header contains type definitions and function declarations that are written in this file .*
- file fileMan.h

  *This header contains macro definitions and function declarations that are written in this file. These functions are made to operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...*

### 6.3.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.4 C source code for my personnal project

### Modules

- C source code for my personnal project, made by me
- C source code for my personnal project, made by me and flex

### 6.4.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.5 C source code for my personnal project, made by me

### Files

- file fcmx.c

  *This header contains function declarations that I will use for one of my project (which isn't still on gitHub)*

### 6.5.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.6 C Headers for my personnal project

### Modules

- C Headers for my personnal project, made by me
- C Headers for my personnal project, made by me and flex

### 6.6.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.7 C Headers for my personnal project, made by me

### Files

- file fcmx.h

  *This header contains function declarations that are wittent on this file and that I will use for one of my project (which isn't still on gitHub)*

### 6.7.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.8 Main C header

### Files

- file fmanc.h

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

### 6.8.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.9 C source code for my personnal project, made by me and flex

### Files

- file lex_yy.c

  *This is the file with the lexical scanner.*

### 6.9.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.10 C Headers for my personnal project, made by me and flex

### Files

- file lex_yy.h

  *This header contains a function written by flex to delete C-style comments in a file.*

### 6.10.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

## 6.11 Flex source files

### Files

- file noComments.l

### 6.11.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

# Chapter 7

# Data Structure Documentation

## 7.1 FMANC_SO Struct Reference

### Data Fields

- size_t charCount
- long long int * pos

### 7.1.1 Detailed Description

Definition at line 80 of file analyze.h.

### 7.1.2 Field Documentation

#### 7.1.2.1 charCount

```
size_t charCount
```

Definition at line 82 of file analyze.h.

#### 7.1.2.2 pos

```
long long int* pos
```

Definition at line 83 of file analyze.h.

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

- docs/src_documented/analyze.h

# Chapter 8

# File Documentation

## 8.1 docs/src_documented/analyze.c File Reference

### Functions

- SHARED size_t countCharInFile (char ∗filePath)
- SHARED stringOccurrences ∗ init_StringOccurences (size_t sizeOfString)
- SHARED void free_stringOccurrences (stringOccurrences ∗toBeDeleted)
- SHARED stringOccurrences ∗ searchStringInFile (char ∗filePath, char ∗toSearch)
- SHARED int replaceStringInFile (char ∗filePath, char ∗toReplaceString, char ∗toAddString)

  *This function replace a wide-character string by another one.*

### 8.1.1 Detailed Description

These functions are made to analyze files content and make operations on it.

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file analyze.c.

### 8.1.2 Function Documentation

#### 8.1.2.1 countCharInFile()

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

Definition at line 37 of file analyze.c.

#### 8.1.2.2 free_stringOccurrences()

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

Definition at line 73 of file analyze.c.

#### 8.1.2.3 init_StringOccurences()

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

Definition at line 61 of file analyze.c.

#### 8.1.2.4 replaceStringInFile()

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

This function replace a wide-character string by another one.

If there is a problem during the call of the function, it will print strerr(errno) on stderr, and/or return an appropriate value

**Parameters**

| | |
|---|---|
| *filePath* | The file path of the document (for example : "./src/example.txt", or "exemple.txt" if the program is called from the same folder). For paths with backslashes, you will need to double it ("\\"), but no need to do this if you use a path that comes from argv arg of your main func. |
| *toReplaceString* | The string to replace |
| *toAddString* | The string to add |

**Returns**

Returns 0 if all good.

**Return values**

| | |
|---|---|
| *-4* | Problem when converting the char string to wide char string |
| *-3* | Problem when calling setlocale() |
| *-2* | Problem when trying to get the ext, name or path of the file |
| *-1* | Problem when opening files |
| *1* | Problem when writing into the new file |
| *2* | Problem when renaming or removing files |
| *3* | The string to replace isn't in the file |

Definition at line 186 of file analyze.c.

#### 8.1.2.5 searchStringInFile()

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

Definition at line 80 of file analyze.c.

## 8.2 analyze.c

Go to the documentation of this file.

```
00001
00024 #include <stdio.h>
00025 #include <stdlib.h>
00026 #include <errno.h>
00027 #include <string.h>
00028 #include <stdbool.h>
00029 #include <locale.h>
00030 #include <limits.h>
00031 #include <stddef.h>
00032 #include <stdint.h>
00033 #include <wchar.h>
00034 #include "analyze.h"
00035 #include "fileMan.h"
00036
00037 SHARED size_t countCharInFile(char *filePath)
00038 {
00039     errno = 0;
00040     setlocale(LC_ALL, "fr_FR.UTF8");
00041     FILE *fil = fopen(filePath, "r, ccs=UTF-8");
00042     if (fil == NULL)
00043     {
00044         fprintf(stderr, "Error :%s\n", strerror(errno));
00045         return -1;
00046     }
00047     size_t returned = 0;
00048     rewind(fil);
00049     while (fgetwc(fil) != WEOF)
00050     {
00051         returned++;
00052     }
00053
00054     fclose(fil);
00055     return returned;
00056 }
00057
00058
00059
00060
00061 SHARED stringOccurrences *init_StringOccurences(size_t sizeOfString)
00062 {
```

```
00063
00064     long long int *position = malloc(sizeof(long long int));
00065     *position = -1;
00066     stringOccurrences *returned = malloc(sizeof(stringOccurrences));
00067     returned->pos = position;
00068     returned->charCount = sizeOfString;
00069
00070     return returned;
00071 }
00072
00073 SHARED void free_stringOccurrences(stringOccurrences *toBeDeleted)
00074 {
00075     free(toBeDeleted->pos);
00076     free(toBeDeleted);
00077 }
00078
00079
00080 SHARED stringOccurrences *searchStringInFile(char *filePath, char *toSearch)
00081 {
00082     errno = 0;
00083     wchar_t toSearchW[strlen(toSearch)+1];
00084
00085     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00086     {
00087         fprintf(stderr, "Error :%s\n", strerror(errno));
00088         return NULL;
00089     }
00090
00091
00092     if (mbstowcs(toSearchW, toSearch, strlen(toSearch)) == (size_t) - 1)
00093     {
00094         fprintf(stderr, "Error :%s\n", strerror(errno));
00095         return NULL;
00096     }
00097
00098
00099
00100     toSearchW[strlen(toSearch)] = L'\0';
00101
00102     if (countCharInFile(filePath) > LLONG_MAX || wcslen(toSearchW) > SIZE_MAX)
00103     {
00104         getFileName(filePath, fErrorName);
00105         fprintf(stderr, "Error :  your file named \"%s\" contains too much characters\n", fErrorName);
00106         return NULL;
00107     }
00108
00109
00110     stringOccurrences *occurencesToSearch = init_StringOccurences(wcslen(toSearchW));
00111
00112
00113     FILE *fil = fopen(filePath, "r, ccs=UTF-8");
00114     if (fil == NULL)
00115     {
00116         fprintf(stderr, "Error :%s\n", strerror(errno));
00117         free_stringOccurrences(occurencesToSearch);
00118         return NULL;
00119     }
00120     rewind(fil);
00121
00122
00123     unsigned int cpt_occ = 0;
00124     wchar_t temp[wcslen(toSearchW)+1];
00125     for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00126     {
00127         temp[i] = '\0';
00128     }
00129     size_t cpt = 0;
00130
00131     long long int cpt2 = 0;
00132     cpt2 = ftell(fil);
00133     wint_t temp2 = fgetwc(fil);
00134     while(temp2 != WEOF)
00135     {
00136         fseek(fil, cpt2, SEEK_SET);
00137         while(cpt <= wcslen(toSearchW))
00138         {
00139             temp[cpt] = fgetwc(fil);
00140
00141             if (temp[cpt] != toSearchW[cpt] || temp[cpt] == WEOF)
00142             {
00143                 if (temp[cpt] == toSearchW[cpt])
00144                 {
00145                     cpt++;
00146                 }
00147                 break;
00148             }
00149             else
```

```
00150             {
00151                 cpt++;
00152             }
00153         }
00154 
00155         if (cpt == wcslen(toSearchW))
00156         {
00157             cpt_occ++;
00158             occurencesToSearch->pos = realloc(occurencesToSearch->pos, cpt_occ*sizeof(long long));
00159             *(occurencesToSearch->pos + cpt_occ - 1) = cpt2;
00160         }
00161         cpt = 0;
00162         for (size_t i = 0; i < wcslen(toSearchW)+1; ++i)
00163         {
00164             temp[i] = '\0';
00165         }
00166         fseek(fil, cpt2, SEEK_SET);
00167         temp2 = fgetwc(fil);
00168         cpt2 = ftell(fil);
00169     }
00170     if (cpt_occ == 0)
00171     {
00172         occurencesToSearch->pos = realloc(occurencesToSearch->pos, sizeof(long long));
00173         *(occurencesToSearch->pos) = -1;
00174     }
00175     else
00176     {
00177         occurencesToSearch->pos = realloc(occurencesToSearch->pos, (cpt_occ + 1)*sizeof(long long));
00178         *(occurencesToSearch->pos + cpt_occ) = -1;
00179     }
00180 
00181     fclose(fil);
00182     return occurencesToSearch;
00183 }
00184 
00185 
00186 SHARED int replaceStringInFile(char *filePath, char *toReplaceString, char *toAddString)
00187 {
00188     stringOccurrences *toReplaceOccurrences = searchStringInFile(filePath, toReplaceString);
00189     errno = 0;
00190     wchar_t toAdd[strlen(toAddString)+1];
00191     wchar_t toReplace[strlen(toReplaceString)+1];
00192 
00193     if (toReplaceOccurrences == NULL || *(toReplaceOccurrences->pos) == -1)
00194     {
00195         return 3;
00196     }
00197 
00198     if (setlocale(LC_ALL, "fr_FR.UTF8") == NULL)
00199     {
00200         fprintf(stderr, "Error :%s\n", strerror(errno));
00201         return -3;
00202     }
00203 
00204     if (mbstowcs(toAdd, toAddString, strlen(toAddString)) == (size_t) - 1)
00205     {
00206         fprintf(stderr, "Error :%s\n", strerror(errno));
00207         return -4;
00208     }
00209 
00210     if (mbstowcs(toReplace, toReplaceString, strlen(toReplaceString)) == (size_t) - 1)
00211     {
00212         fprintf(stderr, "Error :%s\n", strerror(errno));
00213         return -4;
00214     }
00215 
00216     toAdd[strlen(toAddString)] = L'\0';
00217     toReplace[strlen(toReplaceString)] = L'\0';
00218 
00219     FILE *filToR = fopen(filePath, "r, ccs=UTF-8");
00220     if (filToR == NULL)
00221     {
00222         fprintf(stderr, "Error :%s\n", strerror(errno));
00223         return -1;
00224     }
00225     rewind(filToR);
00226 
00227 
00228     getFilePath(filePath, sFilePath);
00229 
00230 
00231     getFileName(filePath, sFileName);
00232     if (sFileName[0] == '\0')
00233     {
00234         return -2;
00235     }
00236     getFileExtension(filePath, sFileExt);
```

```
00237     if (sFileExt[0] == '\0')
00238     {
00239         return -2;
00240     }
00241 
00242     FILE *filToW = NULL;
00243     char *replaced = "replaced";
00244     char *tempName = malloc((MAX_FNAME_SIZE + MAX_FPATH_SIZE + MAX_FEXT_SIZE)*sizeof(char));
00245     *tempName = '\0';
00246     if(sFilePath[0] != '\0')
00247     {
00248         tempName = strcat(tempName, sFilePath);
00249         tempName = strcat(tempName, replaced);
00250         tempName = strcat(tempName, sFileExt);
00251         filToW = fopen(tempName, "w+, ccs=UTF-8");
00252     }
00253     else
00254     {
00255         tempName = strcat(tempName, replaced);
00256         tempName = strcat(tempName, sFileExt);
00257         filToW = fopen(tempName, "w+, ccs=UTF-8");
00258     }
00259 
00260 
00261     if (filToW == NULL)
00262     {
00263         fprintf(stderr, "Error :%s\n", strerror(errno));
00264         return -1;
00265     }
00266     rewind(filToW);
00267 
00268     int cpt = 0;
00269     int old_cpt = 0;
00270     wchar_t temp = L'\0';
00271     wchar_t temp2 = fgetwc(filToR);
00272 
00273 
00274     while(temp2!=WEOF)
00275     {
00276         ungetwc(temp2, filToR);
00277         while(*(toReplaceOccurrences->pos + cpt) != -1 && *(toReplaceOccurrences->pos + cpt) >= 0)
00278         {
00279             if (ftell(filToR) == *(toReplaceOccurrences->pos + cpt))
00280             {
00281                 for (size_t i = 0; i < wcslen(toAdd); ++i)
00282                 {
00283                     if(fputwc(toAdd[i], filToW) != toAdd[i])
00284                     {
00285                         fprintf(stderr, "ERR :%s\n", strerror(errno));
00286                         return 1;
00287                     }
00288                 }
00289                 cpt++;
00290                 for (size_t i = 0; i<toReplaceOccurrences->charCount; ++i)
00291                 {
00292                     if(fgetwc(filToR) == WEOF)
00293                         break;
00294                 }
00295             }
00296 
00297             if (temp2!=WEOF && old_cpt == cpt)
00298             {
00299                 temp = fgetwc(filToR);
00300                 if(fputwc(temp, filToW) != temp)
00301                 {
00302                     fprintf(stderr, "ERR :%s\n", strerror(errno));
00303                     return 1;
00304                 }
00305             }
00306             else
00307             {
00308                 old_cpt++;
00309             }
00310 
00311 
00312         }
00313 
00314         if (temp!=WEOF)
00315         {
00316             temp = fgetwc(filToR);
00317             if(fputwc(temp, filToW) != temp)
00318             {
00319                 fprintf(stderr, "ERR :%s\n", strerror(errno));
00320                 return 1;
00321             }
00322         }
00323         temp2 = fgetwc(filToR);
```

```
00324     }
00325
00326     fclose(filToR);
00327     fclose(filToW);
00328
00329
00330
00331     if (remove(filePath) != 0)
00332     {
00333         fprintf(stderr, "ERR :%s\n", strerror(errno));
00334         return 2;
00335     }
00336     else if (rename(tempName, filePath) != 0)
00337     {
00338         fprintf(stderr, "ERR :%s\n", strerror(errno));
00339         return 2;
00340     }
00341
00342
00343
00344
00345
00346
00347     free(tempName);
00348     free_stringOccurrences(toReplaceOccurrences);
00349
00350     return 0;
00351 }
```

## 8.3 docs/src_documented/analyze.h File Reference

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

### Data Structures

- struct FMANC_SO

### Macros

- #define SHARED

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

### Typedefs

- typedef typedefSHARED struct FMANC_SO stringOccurrences

### Functions

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

  *This function replace a wide-character string by another one.*

### 8.3.1 Detailed Description

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

These functions are made to analyze files content and make operations on it.

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file analyze.h.

### 8.3.2 Macro Definition Documentation

#### 8.3.2.1 SHARED

```
#define SHARED
```

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

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)
#   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 at line 73 of file analyze.h.

### 8.3.3 Typedef Documentation

#### 8.3.3.1 stringOccurrences

```
typedef typedefSHARED struct FMANC_SO stringOccurrences
```

Definition at line 86 of file analyze.h.

### 8.3.4 Function Documentation

#### 8.3.4.1 countCharInFile()

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

Definition at line 37 of file analyze.c.

#### 8.3.4.2 free_stringOccurrences()

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

Definition at line 73 of file analyze.c.

#### 8.3.4.3 init_StringOccurences()

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

Definition at line 61 of file analyze.c.

#### 8.3.4.4 replaceStringInFile()

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

This function replace a wide-character string by another one.

If there is a problem during the call of the function, it will print strerr(errno) on stderr, and/or return an appropriate value

**Parameters**

| | |
|---|---|
| *filePath* | The file path of the document (for example : "./src/example.txt", or "exemple.txt" if the program is called from the same folder). For paths with backslashes, you will need to double it ("\\"), but no need to do this if you use a path that comes from argv arg of your main func. |
| *toReplaceString* | The string to replace |
| *toAddString* | The string to add |

**Returns**

Returns 0 if all good.

**Return values**

| | |
|---|---|
| *-4* | Problem when converting the char string to wide char string |
| *-3* | Problem when calling setlocale() |
| *-2* | Problem when trying to get the ext, name or path of the file |
| *-1* | Problem when opening files |
| *1* | Problem when writing into the new file |
| *2* | Problem when renaming or removing files |
| *3* | The string to replace isn't in the file |

Definition at line 186 of file analyze.c.

#### 8.3.4.5 searchStringInFile()

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

Definition at line 80 of file analyze.c.

## 8.4 analyze.h

Go to the documentation of this file.

```
00001
00034 #ifndef ANALYZE_H
00035 #define ANALYZE_H
00036
00059 # if defined(_WIN32)
00060 /****************** "-D STATIC" ******************/
00061 #   if defined(STATIC)
00062 #     define SHARED
00063 /***************** "-D BUILD_DLL" ****************/
00064 #   else
00065 #     if defined(BUILD_DLL)
00066 #       define SHARED __declspec(dllexport)
00067 #     else
00068 #       define SHARED __declspec(dllimport)
00069 #     endif
00070 #   endif
00071 /******************* DEFAULT *******************/
00072 # else
```

```
00073 #   define SHARED
00074 # endif
00075
00076
00077 #include <stddef.h>
00078
00079
00080 SHARED struct FMANC_SO
00081 {
00082     size_t charCount;
00083     long long int *pos;
00084 };
00085
00086 SHARED typedef struct FMANC_SO stringOccurrences;
00087
00088 SHARED size_t countCharInFile(char *filePath);
00089 SHARED stringOccurrences *init_StringOccurences(size_t sizeOfString);
00090 SHARED void free_stringOccurrences(stringOccurrences *toBeDeleted);
00091 SHARED stringOccurrences *searchStringInFile(char *filePath, char *toSearch);
00092
00110 SHARED int replaceStringInFile(char *filePath, char *toReplaceString, char *toAddString);
00111
00112
00113 #endif
00114
```

## 8.5 docs/src_documented/fcmx.c File Reference

This header contains function declarations that I will use for one of my project (which isn't still on gitHub)

### 8.5.1 Detailed Description

This header contains function declarations that I will use for one of my project (which isn't still on gitHub)

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file fcmx.c.

## 8.6 fcmx.c

Go to the documentation of this file.

```
00001
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035 #include <errno.h>
00036 #include <string.h>
00037 #include "fileMan.h"
00038
```

## 8.7 docs/src_documented/fcmx.h File Reference

This header contains function declarations that are wittent on this file and that I will use for one of my project (which isn't still on gitHub)

## Macros

- #define SHARED

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

## Functions

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

  *Function just declared and still not done.*

### 8.7.1 Detailed Description

This header contains function declarations that are wittent on this file and that I will use for one of my project (which isn't still on gitHub)

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file fcmx.h.

### 8.7.2 Macro Definition Documentation

#### 8.7.2.1 SHARED

```
#define SHARED
```

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

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)
#   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 at line 72 of file fcmx.h.

### 8.7.3 Function Documentation

#### 8.7.3.1 copyFileWithoutStrings()

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

Function just declared and still not done.

**Todo** Do it (lol).

**Parameters**

| | | |
|---|---|---|
| in | *argc* | The count of arguments |
| | *filePath* | The file path |
| in | <*unnamed*> | { parameter_description } |

**Returns**

{ description_of_the_return_value }

## 8.8 fcmx.h

Go to the documentation of this file.

```
00001
00033 #ifndef FCMX_H
00034 #define FCMX_H
00035
00058 # if defined(_WIN32)
00059 /***************** "-D STATIC" ******************/
00060 #   if defined(STATIC)
00061 #     define SHARED
00062 /***************** "-D BUILD_DLL" ****************/
00063 #   else
00064 #     if defined(BUILD_DLL)
00065 #       define SHARED __declspec(dllexport)
00066 #     else
00067 #       define SHARED __declspec(dllimport)
00068 #     endif
00069 #   endif
00070 /****************** DEFAULT *******************/
00071 # else
00072 #   define SHARED
00073 # endif
00074
00075 SHARED int copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00076
00077
00078
00079 #endif
00080
00081
```

## 8.9 docs/src_documented/fileMan.c File Reference

These functions are made to do operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...

### Functions

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

### 8.9.1 Detailed Description

These functions are made to do operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file fileMan.c.

### 8.9.2 Function Documentation

#### 8.9.2.1 copyFileWithoutTabAndLineBreak()

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

Copy a file without tab and line break.

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.

**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 31 of file fileMan.c.

#### 8.9.2.2 fgetFileExtension()

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

Definition at line 93 of file fileMan.c.

#### 8.9.2.3 fgetFileName()

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

Definition at line 130 of file fileMan.c.

#### 8.9.2.4 fgetFilePath()

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

Definition at line 178 of file fileMan.c.

## 8.10 fileMan.c

Go to the documentation of this file.

```
00001
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <errno.h>
00028 #include <string.h>
00029 #include "fileMan.h"
00030
00031 SHARED char *copyFileWithoutTabAndLineBreak(char *sourceFilePath, char **pathToCopy) //not finished
00032 {
00033
00034     errno = 0;
00035     getFileName(sourceFilePath, sourceFileName);
00036     getFileExtension(sourceFilePath, sourceFileExtension);
00037
00038
00039
00040     FILE *sourceFile = fopen(sourceFilePath, "r");
00041
00042     if (sourceFile == NULL)
00043     {
00044         fprintf(stderr, "Error :%s\n", strerror(errno));
00045         return NULL;
00046     }
00047     rewind(sourceFile);
00048     char *copiedName = NULL;
00049     if (pathToCopy == NULL)
00050     {
00051         copiedName = strcat(strcat(sourceFileName,"_copied"), sourceFileExtension); //modify here
00052     }
00053     else
00054     {
00055         copiedName = *pathToCopy;
00056     }
00057
00058     FILE *copiedFile = fopen(copiedName, "w");
00059     if (copiedFile == NULL)
00060     {
00061         fprintf(stderr, "Error :%s\n", strerror(errno));
00062         fclose(sourceFile);
00063         return NULL;
00064     }
00065     rewind(copiedFile);
00066
00067     while(fgetc(sourceFile) != EOF)
00068     {
00069         fseek(sourceFile, -1, SEEK_CUR);
00070         if (fgetc(sourceFile) != '\n')
00071         {
00072             fseek(sourceFile, -1, SEEK_CUR);
00073             if (fgetc(sourceFile) != '\t')
00074             {
00075                 fseek(sourceFile, -1, SEEK_CUR);
00076                 fputc(fgetc(sourceFile), copiedFile);
00077             }
00078         }
00079     }
00080     char *returnedName = NULL;
00081     int i = 0;
00082     while(sourceFileName[i] != '\0')
00083     {
00084         *(returnedName + i) = sourceFileName[i];
00085         i++;
00086     }
00087     *(returnedName + i) = '\0';
00088     fclose(copiedFile);
00089     fclose(sourceFile);
00090     return returnedName;
00091 }
00092
00093 SHARED void fgetFileExtension(char *sourceFilePath, char *extension)
00094 {
00095     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00096     {
00097         fprintf(stderr, "\nError :  Full path is too big\n");
00098         return;
00099     }
00100     int cpt = strlen(sourceFilePath);
00101     char pt = *(sourceFilePath + cpt);
00102
00103
00104     while((pt != '.')  && (cpt >= 0))
00105     {
```

```
00106         cpt--;
00107         pt = *(sourceFilePath + cpt);
00108     }
00109     if (cpt < 0)
00110     {
00111         fprintf(stderr, "\nError :  incorrect file path\n");
00112     }
00113     else
00114     {
00115         char res[strlen(sourceFilePath)-cpt+1];
00116         for (int i = cpt; i < strlen(sourceFilePath); ++i)
00117         {
00118             res[i - cpt] = *(sourceFilePath + i);
00119         }
00120         res[strlen(sourceFilePath)-cpt] = '\0';
00121         for (int i = 0; i < strlen(res); ++i)
00122         {
00123             *(extension + i) = res[i];
00124         }
00125         *(extension + strlen(res)) = '\0';
00126 
00127     }
00128 }
00129 
00130 SHARED void fgetFileName(char *sourceFilePath, char *fileName)
00131 {
00132     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00133     {
00134         fprintf(stderr, "\nError :  Full path is too big\n");
00135         return;
00136     }
00137     int cpt = strlen(sourceFilePath);
00138     char pt = *(sourceFilePath + cpt);
00139 
00140 
00141     while(cpt >= 0)
00142     {
00143         cpt--;
00144         pt = *(sourceFilePath + cpt);
00145         if (pt == '/' || pt == '\\')
00146         {
00147             break;
00148         }
00149     }
00150     cpt++;
00151     if (cpt < 0)
00152     {
00153         fprintf(stderr, "\nError :  incorrect file path\n");
00154     }
00155     else
00156     {
00157         char res[strlen(sourceFilePath)-cpt+1];
00158         for (int i = cpt; i < strlen(sourceFilePath); ++i)
00159         {
00160             res[i - cpt] = *(sourceFilePath + i);
00161         }
00162         res[strlen(sourceFilePath)-cpt] = '\0';
00163         for (int i = 0; i < strlen(res); ++i)
00164         {
00165             *(fileName + i) = res[i];
00166         }
00167         *(fileName + strlen(res)) = '\0';
00168         cpt = strlen(fileName) - 1;
00169         while(fileName[cpt] != '.')
00170         {
00171             fileName[cpt] = '\0';
00172             cpt--;
00173         }
00174         fileName[cpt] = '\0';
00175     }
00176 }
00177 
00178 SHARED void fgetFilePath(char *sourceFilePath, char *filePath)
00179 {
00180     if (strlen(sourceFilePath) > MAX_FEXT_SIZE + MAX_FPATH_SIZE + MAX_FNAME_SIZE)
00181     {
00182         fprintf(stderr, "\nError :  Full path is too big\n");
00183         return;
00184     }
00185     int cpt = strlen(sourceFilePath);
00186     char pt = *(sourceFilePath + cpt);
00187 
00188 
00189     while(cpt >= 0)
00190     {
00191         cpt--;
00192         pt = *(sourceFilePath + cpt);
```

```
00193         if (pt == '/' || pt == '\\')
00194         {
00195             break;
00196         }
00197     }
00198
00199
00200     if (cpt < 0)
00201     {
00202         return;
00203     }
00204
00205     else
00206     {
00207         char res[cpt+1];
00208         for (int i = 0; i < cpt; ++i)
00209         {
00210             res[i] = *(sourceFilePath + i);
00211         }
00212         res[cpt + 1] = '\0';
00213         for (int i = 0; i < strlen(res); ++i)
00214         {
00215             *(filePath + i) = res[i];
00216         }
00217         if (pt == '/')
00218         {
00219             *(filePath + strlen(res)-1) = '/';
00220         }
00221         else
00222         {
00223             *(filePath + strlen(res)-1) = '\\';
00224         }
00225         *(filePath + strlen(res)) = '\0';
00226
00227     }
00228 }
```

## 8.11 docs/src_documented/fileMan.h File Reference

This header contains macro definitions and function declarations that are written in this file. These functions are made to operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...

### Macros

- #define SHARED

  *Useful to choose how to use the lib on Windows systems.*
- #define MAX_FEXT_SIZE 50
- #define MAX_FNAME_SIZE 256
- #define MAX_FPATH_SIZE 512
- #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = ""; fgetFile↩ Extension(sourceFilePath, 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).*

## Functions

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

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

### 8.11.1 Detailed Description

This header contains macro definitions and function declarations that are written in this file. These functions are made to operate simple operation on files or file names, when there is no neee to analyze something like orccurrences, ...

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file fileMan.h.

### 8.11.2 Macro Definition Documentation

#### 8.11.2.1 getFileExtension

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

Gives you the file extension.

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 stock the extension. |

**Returns**

This doesn't really return anything, but displays an error message if no extension or invalid extension.

Definition at line 112 of file fileMan.h.

#### 8.11.2.2 getFileName

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

Gives you the file name.

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.

Definition at line 124 of file fileMan.h.

#### 8.11.2.3 getFilePath

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

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

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 first character. Won't display any error message if no path.

Definition at line 136 of file fileMan.h.

#### 8.11.2.4 MAX_FEXT_SIZE

```
#define MAX_FEXT_SIZE 50
```

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

Definition at line 84 of file fileMan.h.

#### 8.11.2.5 MAX_FNAME_SIZE

```
#define MAX_FNAME_SIZE 256
```

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

Definition at line 92 of file fileMan.h.

#### 8.11.2.6 MAX_FPATH_SIZE

```
#define MAX_FPATH_SIZE 512
```

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 100 of file fileMan.h.

#### 8.11.2.7 SHARED

```
#define SHARED
```

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

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)
#   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 at line 76 of file fileMan.h.

### 8.11.3 Function Documentation

#### 8.11.3.1 copyFileWithoutTabAndLineBreak()

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

Copy a file without tab and line break.

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.

**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 31 of file fileMan.c.

#### 8.11.3.2 fgetFileExtension()

```
SHARED void fgetFileExtension (
            char * sourceFileName,
            char * extension )
```

Definition at line 93 of file fileMan.c.

#### 8.11.3.3 fgetFileName()

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

Definition at line 130 of file fileMan.c.

#### 8.11.3.4 fgetFilePath()

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

Definition at line 178 of file fileMan.c.

## 8.12 fileMan.h

Go to the documentation of this file.

```
00001
00037 #ifndef FILEMAN_H
00038 #define FILEMAN_H
00039
00062 # if defined(_WIN32)
00063 /***************** "-D STATIC" ******************/
00064 #   if defined(STATIC)
00065 #     define SHARED
00066 /**************** "-D BUILD_DLL" ****************/
00067 #   else
00068 #     if defined(BUILD_DLL)
00069 #       define SHARED __declspec(dllexport)
00070 #     else
00071 #       define SHARED __declspec(dllimport)
00072 #     endif
00073 #   endif
00074 /****************** DEFAULT *******************/
00075 # else
00076 #   define SHARED
00077 # endif
00078
00083 #ifndef MAX_FEXT_SIZE
00084 #define MAX_FEXT_SIZE 50
00085 #endif
00086
00091 #ifndef MAX_FNAME_SIZE
00092 #define MAX_FNAME_SIZE 256
00093 #endif
00094
00099 #ifndef MAX_FPATH_SIZE
00100 #define MAX_FPATH_SIZE 512
00101 #endif
00102
00111 #ifndef getFileExtension
00112 #define getFileExtension(sourceFilePath, extension) char extension[MAX_FEXT_SIZE] = "";
      fgetFileExtension(sourceFilePath, extension)
00113 #endif
00114
00123 #ifndef getFileName
00124 #define getFileName(sourceFilePath, name) char name[MAX_FNAME_SIZE] = ""; fgetFileName(sourceFilePath,
      name)
00125 #endif
00126
00135 #ifndef getFilePath
00136 #define getFilePath(sourceFilePath, path) char path[MAX_FPATH_SIZE] = ""; fgetFilePath(sourceFilePath,
      path)
00137 #endif
00138
00151 SHARED char *copyFileWithoutTabAndLineBreak(char *sourceFilePath, char **pathToCopy); // copied file
      will be named like <sourceFile name>_copied
00152
00165 SHARED int copyFileWithoutStrings(const unsigned int argc, char *filePath, ...); // to do
00166 SHARED void fgetFileExtension(char *sourceFileName, char *extension);
00167 SHARED void fgetFileName(char *sourceFilePath, char *fileName);
00168 SHARED void fgetFilePath(char *sourceFilePath, char *filePath);
00169
00170
00171 #endif
00172
00173
```

## 8.13 docs/src_documented/fmanc.h File Reference

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

## Macros

- #define SHARED

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

### 8.13.1 Detailed Description

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

If you don't want to have troubles, just include this one instead of including the others one by one. If you want to use the functions defined in this or this source files, then write something like
```
#define USE_FCMX TRUE
```

(actually, just define USE_FCMX with a certain value, no matter what it is).

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file fmanc.h.

### 8.13.2 Macro Definition Documentation

#### 8.13.2.1 SHARED

```
#define SHARED
```

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

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)
#   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 at line 75 of file fmanc.h.

## 8.14 fmanc.h

Go to the documentation of this file.

```
00001
00036 #ifndef FMANC_H
00037 #define FMANC_H
00038
00061 # if defined(_WIN32)
00062 /***************** "-D STATIC" ******************/
00063 #   if defined(STATIC)
00064 #     define SHARED
00065 /**************** "-D BUILD_DLL" ****************/
00066 #   else
00067 #     if defined(BUILD_DLL)
00068 #       define SHARED __declspec(dllexport)
00069 #     else
00070 #       define SHARED __declspec(dllimport)
00071 #     endif
00072 #   endif
00073 /****************** DEFAULT ******************/
00074 # else
00075 #   define SHARED
00076 # endif
00077
00078
00079 #include "fileMan.h"
00080 #include "analyze.h"
00081
00082 #if defined(USE_FCMX)
00083 #include "fcmx.h"
00084 #include "./notByMe/lex_yy.h"
00085 #endif
00086
00087
00088
00089
00090
00091
00092
00093 #endif
```

## 8.15 docs/src_documented/notByMe/lex_yy.c File Reference

This is the file with the lexical scanner.

### Macros

- #define YY_INT_ALIGNED short int
- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 6
- #define YY_FLEX_SUBMINOR_VERSION 4
- #define FLEX_BETA
- #define FLEXINT_H
- #define INT8_MIN (-128)
- #define INT16_MIN (-32767-1)
- #define INT32_MIN (-2147483647-1)
- #define INT8_MAX (127)
- #define INT16_MAX (32767)
- #define INT32_MAX (2147483647)
- #define UINT8_MAX (255U)
- #define UINT16_MAX (65535U)
- #define UINT32_MAX (4294967295U)
- #define SIZE_MAX (~(size_t)0)
- #define yyconst const

- #define yynoreturn
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((YY_CHAR) (c))
- #define BEGIN (yy_start) = 1 + 2 ∗
- #define YY_START (((yy_start) - 1) / 2)
- #define YYSTATE YY_START
- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart( yyin )
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) ∗ sizeof(yy_state_type))
- #define YY_TYPEDEF_YY_BUFFER_STATE
- #define YY_TYPEDEF_YY_SIZE_T
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define YY_LESS_LINENO(n)
- #define YY_LINENO_REWIND_TO(ptr)
- #define yyless(n)
- #define unput(c) yyunput( c, (yytext_ptr) )
- #define YY_STRUCT_YY_BUFFER_STATE
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER
- #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
- #define YY_FLUSH_BUFFER yy_flush_buffer( YY_CURRENT_BUFFER )
- #define yy_new_buffer yy_create_buffer
- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define yywrap() (/∗CONSTCOND∗/1)
- #define YY_SKIP_YYWRAP
- #define yytext_ptr yytext
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 17
- #define YY_END_OF_BUFFER 18
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define String 1
- #define Char 2
- #define Comment 3
- #define CPPComment 4
- #define YY_EXTRA_TYPE void ∗
- #define YY_READ_BUF_SIZE 8192
- #define ECHO do { if (fwrite( yytext, (size_t) yyleng, 1, yyout )) {} } while (0)
- #define YY_INPUT(buf, result, max_size)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25
- #define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
- #define YY_DECL_IS_OURS 1
- #define YY_DECL int yylex (void)

- #define YY_USER_ACTION
- #define YY_BREAK /∗LINTED∗/break;
- #define YY_RULE_SETUP YY_USER_ACTION
- #define **YY_EXIT_FAILURE** 2
- #define yyless(n)
- #define **YYTABLES_NAME** "yytables"

## Typedefs

- typedef signed char flex_int8_t
- typedef short int flex_int16_t
- typedef int flex_int32_t
- typedef unsigned char flex_uint8_t
- typedef unsigned short int flex_uint16_t
- typedef unsigned int flex_uint32_t
- typedef struct yy_buffer_state ∗ YY_BUFFER_STATE
- typedef size_t yy_size_t
- typedef flex_uint8_t YY_CHAR
- typedef int yy_state_type

## Functions

- void **yyrestart** (FILE ∗input_file)
- void **yy_switch_to_buffer** (YY_BUFFER_STATE new_buffer)
- YY_BUFFER_STATE **yy_create_buffer** (FILE ∗file, int size)
- void **yy_delete_buffer** (YY_BUFFER_STATE b)
- void **yy_flush_buffer** (YY_BUFFER_STATE b)
- void **yypush_buffer_state** (YY_BUFFER_STATE new_buffer)
- void **yypop_buffer_state** (void)
- YY_BUFFER_STATE **yy_scan_buffer** (char ∗base, yy_size_t size)
- YY_BUFFER_STATE **yy_scan_string** (const char ∗yy_str)
- YY_BUFFER_STATE **yy_scan_bytes** (const char ∗bytes, int len)
- void ∗ **yyalloc** (yy_size_t)
- void ∗ **yyrealloc** (void ∗, yy_size_t)
- void **yyfree** (void ∗)
- int **fileno** (FILE ∗)
- int **yylex_destroy** (void)
- int **yyget_debug** (void)
- void **yyset_debug** (int debug_flag)
- YY_EXTRA_TYPE **yyget_extra** (void)
- void **yyset_extra** (YY_EXTRA_TYPE user_defined)
- FILE ∗ **yyget_in** (void)
- void **yyset_in** (FILE ∗_in_str)
- FILE ∗ **yyget_out** (void)
- void **yyset_out** (FILE ∗_out_str)
- int **yyget_leng** (void)
- char ∗ **yyget_text** (void)
- int **yyget_lineno** (void)
- void **yyset_lineno** (int _line_number)
- int yylex (void)
- if (!(yy_init))
- SHARED int deleteCStyleComments (char ∗filePath)

## Variables

- int yyleng
- FILE * yyin = NULL
- FILE * yyout = NULL
- int yylineno = 1
- char * yytext
- int yy_flex_debug = 0
- YY_DECL
- char * yy_cp
- char * yy_bp
- int yy_act
- int(* jj2_junk )(void) = input

### 8.15.1 Detailed Description

This is the file with the lexical scanner.

I let it here, but if you know how to use these functions (like me, lol), don't use them. Just Use the one defined in lex_yy.h. For more informations about my func, just go on the header's description.

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file lex_yy.c.

### 8.15.2 Macro Definition Documentation

#### 8.15.2.1 BEGIN

```
#define BEGIN (yy_start) = 1 + 2 *
```

Definition at line 148 of file lex_yy.c.

#### 8.15.2.2 Char

```
#define Char 2
```

Definition at line 479 of file lex_yy.c.

#### 8.15.2.3 Comment

```
#define Comment 3
```

Definition at line 480 of file lex_yy.c.

#### 8.15.2.4 CPPComment

```
#define CPPComment 4
```

Definition at line 481 of file lex_yy.c.

#### 8.15.2.5 ECHO

```
#define ECHO do { if (fwrite( yytext, (size_t) yyleng, 1, yyout )) {} } while (0)
```

Definition at line 576 of file lex_yy.c.

#### 8.15.2.6 EOB_ACT_CONTINUE_SCAN

```
#define EOB_ACT_CONTINUE_SCAN 0
```

Definition at line 192 of file lex_yy.c.

#### 8.15.2.7 EOB_ACT_END_OF_FILE

```
#define EOB_ACT_END_OF_FILE 1
```

Definition at line 193 of file lex_yy.c.

#### 8.15.2.8 EOB_ACT_LAST_MATCH

```
#define EOB_ACT_LAST_MATCH 2
```

Definition at line 194 of file lex_yy.c.

#### 8.15.2.9 FLEX_BETA

```
#define FLEX_BETA
```

Definition at line 44 of file lex_yy.c.

#### 8.15.2.10 FLEX_SCANNER

```
#define FLEX_SCANNER
```

Definition at line 39 of file lex_yy.c.

#### 8.15.2.11 FLEXINT_H

```
#define FLEXINT_H
```

Definition at line 60 of file lex_yy.c.

#### 8.15.2.12 INITIAL

```
#define INITIAL 0
```

Definition at line 477 of file lex_yy.c.

#### 8.15.2.13 INT16_MAX

```
#define INT16_MAX (32767)
```

Definition at line 102 of file lex_yy.c.

#### 8.15.2.14 INT16_MIN

```
#define INT16_MIN (-32767-1)
```

Definition at line 93 of file lex_yy.c.

#### 8.15.2.15 INT32_MAX

```
#define INT32_MAX (2147483647)
```

Definition at line 105 of file lex_yy.c.

#### 8.15.2.16 INT32_MIN

```
#define INT32_MIN (-2147483647-1)
```

Definition at line 96 of file lex_yy.c.

#### 8.15.2.17 INT8_MAX

```
#define INT8_MAX (127)
```

Definition at line 99 of file lex_yy.c.

#### 8.15.2.18 INT8_MIN

```
#define INT8_MIN (-128)
```

Definition at line 90 of file lex_yy.c.

#### 8.15.2.19 REJECT

```
#define REJECT reject_used_but_not_detected
```

Definition at line 457 of file lex_yy.c.

#### 8.15.2.20 SIZE_MAX

```
#define SIZE_MAX (~(size_t)0)
```

Definition at line 118 of file lex_yy.c.

#### 8.15.2.21 String

```
#define String 1
```

Definition at line 478 of file lex_yy.c.

#### 8.15.2.22 UINT16_MAX

```
#define UINT16_MAX (65535U)
```

Definition at line 111 of file lex_yy.c.

#### 8.15.2.23 UINT32_MAX

```
#define UINT32_MAX (4294967295U)
```

Definition at line 114 of file lex_yy.c.

#### 8.15.2.24 UINT8_MAX

```
#define UINT8_MAX (255U)
```

Definition at line 108 of file lex_yy.c.

#### 8.15.2.25 unput

```
#define unput(
              c ) yyunput( c, (yytext_ptr) )
```

Definition at line 212 of file lex_yy.c.

#### 8.15.2.26 YY_AT_BOL

```
#define YY_AT_BOL( ) (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
```

Definition at line 319 of file lex_yy.c.

#### 8.15.2.27 YY_BREAK

```
#define YY_BREAK /*LINTED*/break;
```

Definition at line 655 of file lex_yy.c.

#### 8.15.2.28 YY_BUF_SIZE

```
#define YY_BUF_SIZE 16384
```

Definition at line 170 of file lex_yy.c.

#### 8.15.2.29 YY_BUFFER_EOF_PENDING

```
#define YY_BUFFER_EOF_PENDING 2
```

Definition at line 247 of file lex_yy.c.

#### 8.15.2.30 YY_BUFFER_NEW

```
#define YY_BUFFER_NEW 0
```

Definition at line 244 of file lex_yy.c.

#### 8.15.2.31 YY_BUFFER_NORMAL

```
#define YY_BUFFER_NORMAL 1
```

Definition at line 245 of file lex_yy.c.

#### 8.15.2.32 YY_CURRENT_BUFFER

```
#define YY_CURRENT_BUFFER
```

**Value:**

```
( (yy_buffer_stack) \
?  (yy_buffer_stack)[(yy_buffer_stack_top)] \
:  NULL)
```

Definition at line 258 of file lex_yy.c.

#### 8.15.2.33 YY_CURRENT_BUFFER_LVALUE

```
#define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
```

Definition at line 262 of file lex_yy.c.

#### 8.15.2.34 YY_DECL

```
#define YY_DECL int yylex (void)
```

Definition at line 643 of file lex_yy.c.

#### 8.15.2.35 YY_DECL_IS_OURS

```
#define YY_DECL_IS_OURS 1
```

Definition at line 639 of file lex_yy.c.

#### 8.15.2.36 YY_DO_BEFORE_ACTION

```
#define YY_DO_BEFORE_ACTION
```

**Value:**

```
    (yytext_ptr) = yy_bp; \
    yyleng = (int) (yy_cp - yy_bp); \
    (yy_hold_char) = *yy_cp; \
    *yy_cp = '\0'; \
    (yy_c_buf_p) = yy_cp;
```

Definition at line 348 of file lex_yy.c.

#### 8.15.2.37 YY_END_OF_BUFFER

```
#define YY_END_OF_BUFFER 18
```

Definition at line 355 of file lex_yy.c.

#### 8.15.2.38 YY_END_OF_BUFFER_CHAR

```
#define YY_END_OF_BUFFER_CHAR 0
```

Definition at line 159 of file lex_yy.c.

#### 8.15.2.39 YY_EXTRA_TYPE

```
#define YY_EXTRA_TYPE void *
```

Definition at line 492 of file lex_yy.c.

#### 8.15.2.40 YY_FATAL_ERROR

```
#define YY_FATAL_ERROR(
              msg ) yy_fatal_error( msg )
```

Definition at line 630 of file lex_yy.c.

#### 8.15.2.41 YY_FLEX_MAJOR_VERSION

```
#define YY_FLEX_MAJOR_VERSION 2
```

Definition at line 40 of file lex_yy.c.

#### 8.15.2.42 YY_FLEX_MINOR_VERSION

```
#define YY_FLEX_MINOR_VERSION 6
```

Definition at line 41 of file lex_yy.c.

#### 8.15.2.43 YY_FLEX_SUBMINOR_VERSION

```
#define YY_FLEX_SUBMINOR_VERSION 4
```

Definition at line 42 of file lex_yy.c.

#### 8.15.2.44 YY_FLUSH_BUFFER

```
#define YY_FLUSH_BUFFER yy_flush_buffer( YY_CURRENT_BUFFER )
```

Definition at line 290 of file lex_yy.c.

#### 8.15.2.45 YY_INPUT

```
#define YY_INPUT(
                buf,
                result,
                max_size )
```

**Value:**

```
    if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
        { \
        int c = '*'; \
        int n; \
        for ( n = 0; n < max_size && \
                     (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
            buf[n] = (char) c; \
        if ( c == '\n' ) \
            buf[n++] = (char) c; \
        if ( c == EOF && ferror( yyin ) ) \
            YY_FATAL_ERROR( "input in flex scanner failed" ); \
        result = n; \
        } \
    else \
        { \
        errno=0; \
        while ( (result = (int) fread(buf, 1, (yy_size_t) max_size, yyin)) == 0 && ferror(yyin)) \
            { \
            if( errno != EINTR) \
                { \
                YY_FATAL_ERROR( "input in flex scanner failed" ); \
                break; \
                } \
            errno=0; \
            clearerr(yyin); \
            } \
        }\
\
```

Definition at line 583 of file lex_yy.c.

#### 8.15.2.46 YY_INT_ALIGNED

```
#define YY_INT_ALIGNED short int
```

Definition at line 35 of file lex_yy.c.

#### 8.15.2.47 YY_LESS_LINENO

```
#define YY_LESS_LINENO(
                n )
```

Definition at line 196 of file lex_yy.c.

#### 8.15.2.48 YY_LINENO_REWIND_TO

```
#define YY_LINENO_REWIND_TO(
                ptr )
```

Definition at line 197 of file lex_yy.c.

#### 8.15.2.49 YY_MORE_ADJ

```
#define YY_MORE_ADJ 0
```

Definition at line 459 of file lex_yy.c.

#### 8.15.2.50 yy_new_buffer

```
#define yy_new_buffer yy_create_buffer
```

Definition at line 300 of file lex_yy.c.

#### 8.15.2.51 YY_NEW_FILE

```
#define YY_NEW_FILE yyrestart( yyin )
```

Definition at line 158 of file lex_yy.c.

#### 8.15.2.52 YY_NULL

```
#define YY_NULL 0
```

Definition at line 137 of file lex_yy.c.

#### 8.15.2.53 YY_NUM_RULES

```
#define YY_NUM_RULES 17
```

Definition at line 354 of file lex_yy.c.

#### 8.15.2.54 YY_READ_BUF_SIZE

```
#define YY_READ_BUF_SIZE 8192
```

Definition at line 567 of file lex_yy.c.

#### 8.15.2.55 YY_RESTORE_YY_MORE_OFFSET

```
#define YY_RESTORE_YY_MORE_OFFSET
```

Definition at line 460 of file lex_yy.c.

#### 8.15.2.56 YY_RULE_SETUP

```
#define YY_RULE_SETUP  YY_USER_ACTION
```

Definition at line 658 of file lex_yy.c.

#### 8.15.2.57 YY_SC_TO_UI

```
#define YY_SC_TO_UI(
              c ) ((YY_CHAR) (c))
```

Definition at line 142 of file lex_yy.c.

#### 8.15.2.58 yy_set_bol

```
#define yy_set_bol(
              at_bol )
```

**Value:**

```
{ \
if ( !  YY_CURRENT_BUFFER ){\
    yyensure_buffer_stack (); \
    YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
} \
YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
}
```

Definition at line 310 of file lex_yy.c.

#### 8.15.2.59 yy_set_interactive

```
#define yy_set_interactive(
              is_interactive )
```

**Value:**

```
{ \
if ( !  YY_CURRENT_BUFFER ){ \
    yyensure_buffer_stack (); \
    YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
} \
YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
}
```

Definition at line 301 of file lex_yy.c.

#### 8.15.2.60 YY_SKIP_YYWRAP

```
#define YY_SKIP_YYWRAP
```

Definition at line 324 of file lex_yy.c.

#### 8.15.2.61 YY_START

```
#define YY_START (((yy_start) - 1) / 2)
```

Definition at line 153 of file lex_yy.c.

#### 8.15.2.62 YY_START_STACK_INCR

```
#define YY_START_STACK_INCR 25
```

Definition at line 625 of file lex_yy.c.

#### 8.15.2.63 YY_STATE_BUF_SIZE

```
#define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
```

Definition at line 176 of file lex_yy.c.

#### 8.15.2.64 YY_STATE_EOF

```
#define YY_STATE_EOF(
              state ) (YY_END_OF_BUFFER + state + 1)
```

Definition at line 156 of file lex_yy.c.

#### 8.15.2.65 YY_STRUCT_YY_BUFFER_STATE

```
#define YY_STRUCT_YY_BUFFER_STATE
```

Definition at line 215 of file lex_yy.c.

#### 8.15.2.66 YY_TYPEDEF_YY_BUFFER_STATE

```
#define YY_TYPEDEF_YY_BUFFER_STATE
```

Definition at line 179 of file lex_yy.c.

#### 8.15.2.67 YY_TYPEDEF_YY_SIZE_T

```
#define YY_TYPEDEF_YY_SIZE_T
```

Definition at line 184 of file lex_yy.c.

#### 8.15.2.68 YY_USER_ACTION

```
#define YY_USER_ACTION
```

Definition at line 650 of file lex_yy.c.

#### 8.15.2.69 yyconst

```
#define yyconst const
```

Definition at line 128 of file lex_yy.c.

#### 8.15.2.70 yyless [1/2]

```
#define yyless(
               n )
```

**Value:**

```
    do \
        { \
        /* Undo effects of setting up yytext.  */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
        *yy_cp = (yy_hold_char); \
        YY_RESTORE_YY_MORE_OFFSET \
        (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
        YY_DO_BEFORE_ACTION; /* set up yytext again */ \
        } \
    while ( 0 )
```

Definition at line 200 of file lex_yy.c.

#### 8.15.2.71 yyless [2/2]

```
#define yyless(
               n )
```

**Value:**

```
    do \
        { \
        /* Undo effects of setting up yytext.  */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
        yytext[yyleng] = (yy_hold_char); \
        (yy_c_buf_p) = yytext + yyless_macro_arg; \
        (yy_hold_char) = *(yy_c_buf_p); \
        *(yy_c_buf_p) = '\0'; \
        yyleng = yyless_macro_arg; \
        } \
    while ( 0 )
```

Definition at line 200 of file lex_yy.c.

#### 8.15.2.72 yymore

```
#define yymore( ) yymore_used_but_not_detected
```

Definition at line 458 of file lex_yy.c.

#### 8.15.2.73 yynoreturn

```
#define yynoreturn
```

Definition at line 133 of file lex_yy.c.

#### 8.15.2.74 YYSTATE

```
#define YYSTATE YY_START
```

Definition at line 154 of file lex_yy.c.

#### 8.15.2.75 yyterminate

```
#define yyterminate( ) return YY_NULL
```

Definition at line 620 of file lex_yy.c.

#### 8.15.2.76 yytext_ptr

```
#define yytext_ptr yytext
```

Definition at line 338 of file lex_yy.c.

#### 8.15.2.77 yywrap

```
#define yywrap( ) (/*CONSTCOND*/1)
```

Definition at line 323 of file lex_yy.c.

### 8.15.3 Typedef Documentation

#### 8.15.3.1 flex_int16_t

```
typedef short int flex_int16_t
```

Definition at line 82 of file lex_yy.c.

#### 8.15.3.2 flex_int32_t

```
typedef int flex_int32_t
```

Definition at line 83 of file lex_yy.c.

#### 8.15.3.3 flex_int8_t

```
typedef signed char flex_int8_t
```

Definition at line 81 of file lex_yy.c.

#### 8.15.3.4 flex_uint16_t

```
typedef unsigned short int flex_uint16_t
```

Definition at line 85 of file lex_yy.c.

#### 8.15.3.5 flex_uint32_t

```
typedef unsigned int flex_uint32_t
```

Definition at line 86 of file lex_yy.c.

#### 8.15.3.6 flex_uint8_t

```
typedef unsigned char flex_uint8_t
```

Definition at line 84 of file lex_yy.c.

#### 8.15.3.7 YY_BUFFER_STATE

```
typedef struct yy_buffer_state* YY_BUFFER_STATE
```

Definition at line 180 of file lex_yy.c.

#### 8.15.3.8 YY_CHAR

```
typedef flex_uint8_t YY_CHAR
```

Definition at line 325 of file lex_yy.c.

#### 8.15.3.9 yy_size_t

```
typedef size_t yy_size_t
```

Definition at line 185 of file lex_yy.c.

#### 8.15.3.10 yy_state_type

```
typedef int yy_state_type
```

Definition at line 329 of file lex_yy.c.

### 8.15.4 Function Documentation

#### 8.15.4.1 deleteCStyleComments()

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

Definition at line 1865 of file lex_yy.c.

#### 8.15.4.2 if()

```
if (
            !  yy_init )
```

Definition at line 669 of file lex_yy.c.

#### 8.15.4.3 yylex()

```
int yylex (
            void  )
```

Definition at line 36 of file noComments.l.

### 8.15.5 Variable Documentation

#### 8.15.5.1 jj2_junk

```
int(* jj2_junk) (void) (
            void  ) = input
```

Definition at line 1863 of file lex_yy.c.

#### 8.15.5.2 yy_act

```
int yy_act
```

Definition at line 667 of file lex_yy.c.

#### 8.15.5.3 yy_bp

```
char * yy_bp
```

Definition at line 666 of file lex_yy.c.

#### 8.15.5.4 yy_cp

```
char* yy_cp
```

Definition at line 666 of file lex_yy.c.

#### 8.15.5.5 YY_DECL

```
YY_DECL
```

**Initial value:**

```
{
    yy_state_type yy_current_state
```

Definition at line 663 of file lex_yy.c.

#### 8.15.5.6 yy_flex_debug

```
int yy_flex_debug = 0
```

Definition at line 452 of file lex_yy.c.

#### 8.15.5.7 yyin

```
FILE * yyin = NULL
```

Definition at line 327 of file lex_yy.c.

#### 8.15.5.8 yyleng

```
int yyleng
```

Definition at line 267 of file lex_yy.c.

#### 8.15.5.9 yylineno

```
int yylineno = 1
```

Definition at line 332 of file lex_yy.c.

#### 8.15.5.10 yyout

```
FILE * yyout = NULL
```

Definition at line 190 of file lex_yy.c.

#### 8.15.5.11 yytext

```
char * yytext
```

Definition at line 461 of file lex_yy.c.

## 8.16 lex_yy.c

Go to the documentation of this file.

```
00001
00033 #line 2 "lex.yy.c"
00034
00035 #define  YY_INT_ALIGNED short int
00036
00037 /* A lexical scanner generated by flex */
00038
00039 #define FLEX_SCANNER
00040 #define YY_FLEX_MAJOR_VERSION 2
00041 #define YY_FLEX_MINOR_VERSION 6
00042 #define YY_FLEX_SUBMINOR_VERSION 4
00043 #if YY_FLEX_SUBMINOR_VERSION > 0
00044 #define FLEX_BETA
00045 #endif
00046
00047 /* First, we deal with  platform-specific or compiler-specific issues.  */
00048
00049 /* begin standard C headers.  */
00050 #include <stdio.h>
00051 #include <string.h>
00052 #include <errno.h>
00053 #include <stdlib.h>
00054
00055 /* end standard C headers.  */
00056
00057 /* flex integer type definitions */
00058
00059 #ifndef FLEXINT_H
00060 #define FLEXINT_H
00061
00062 /* C99 systems have <inttypes.h>.  Non-C99 systems may or may not.  */
00063
00064 #if defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
00065
00066 /* C99 says to define __STDC_LIMIT_MACROS before including stdint.h,
00067 * if you want the limit (max/min) macros for int types.
00068 */
00069 #ifndef __STDC_LIMIT_MACROS
00070 #define __STDC_LIMIT_MACROS 1
00071 #endif
```

```
00072
00073 #include <inttypes.h>
00074 typedef int8_t flex_int8_t;
00075 typedef uint8_t flex_uint8_t;
00076 typedef int16_t flex_int16_t;
00077 typedef uint16_t flex_uint16_t;
00078 typedef int32_t flex_int32_t;
00079 typedef uint32_t flex_uint32_t;
00080 #else
00081 typedef signed char flex_int8_t;
00082 typedef short int flex_int16_t;
00083 typedef int flex_int32_t;
00084 typedef unsigned char flex_uint8_t;
00085 typedef unsigned short int flex_uint16_t;
00086 typedef unsigned int flex_uint32_t;
00087
00088 /* Limits of integral types.  */
00089 #ifndef INT8_MIN
00090 #define INT8_MIN               (-128)
00091 #endif
00092 #ifndef INT16_MIN
00093 #define INT16_MIN              (-32767-1)
00094 #endif
00095 #ifndef INT32_MIN
00096 #define INT32_MIN              (-2147483647-1)
00097 #endif
00098 #ifndef INT8_MAX
00099 #define INT8_MAX               (127)
00100 #endif
00101 #ifndef INT16_MAX
00102 #define INT16_MAX              (32767)
00103 #endif
00104 #ifndef INT32_MAX
00105 #define INT32_MAX              (2147483647)
00106 #endif
00107 #ifndef UINT8_MAX
00108 #define UINT8_MAX              (255U)
00109 #endif
00110 #ifndef UINT16_MAX
00111 #define UINT16_MAX             (65535U)
00112 #endif
00113 #ifndef UINT32_MAX
00114 #define UINT32_MAX             (4294967295U)
00115 #endif
00116
00117 #ifndef SIZE_MAX
00118 #define SIZE_MAX               (~(size_t)0)
00119 #endif
00120
00121 #endif /* !  C99 */
00122
00123 #endif /* !  FLEXINT_H */
00124
00125 /* begin standard C++ headers.  */
00126
00127 /* TODO: this is always defined, so inline it */
00128 #define yyconst const
00129
00130 #if defined(__GNUC__) && __GNUC__ >= 3
00131 #define yynoreturn __attribute__((__noreturn__))
00132 #else
00133 #define yynoreturn
00134 #endif
00135
00136 /* Returned upon end-of-file.  */
00137 #define YY_NULL 0
00138
00139 /* Promotes a possibly negative, possibly signed char to an
00140 *   integer in range [0..255] for use as an array index.
00141 */
00142 #define YY_SC_TO_UI(c) ((YY_CHAR) (c))
00143
00144 /* Enter a start condition.    This macro really ought to take a parameter,
00145 * but we do it the disgusting crufty way forced on us by the ()-less
00146 * definition of BEGIN.
00147 */
00148 #define BEGIN (yy_start) = 1 + 2 *
00149 /* Translate the current start state into a value that can be later handed
00150 * to BEGIN to return to the state.    The YYSTATE alias is for lex
00151 * compatibility.
00152 */
00153 #define YY_START (((yy_start) - 1) / 2)
00154 #define YYSTATE YY_START
00155 /* Action number for EOF rule of a given start state.  */
00156 #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
00157 /* Special action meaning "start processing a new file".  */
00158 #define YY_NEW_FILE yyrestart( yyin  )
```

```
00159 #define YY_END_OF_BUFFER_CHAR 0
00160
00161 /* Size of default input buffer.  */
00162 #ifndef YY_BUF_SIZE
00163 #ifdef __ia64__
00164 /* On IA-64, the buffer size is 16k, not 8k.
00165 * Moreover, YY_BUF_SIZE is 2*YY_READ_BUF_SIZE in the general case.
00166 * Ditto for the __ia64__ case accordingly.
00167 */
00168 #define YY_BUF_SIZE 32768
00169 #else
00170 #define YY_BUF_SIZE 16384
00171 #endif /* __ia64__ */
00172 #endif
00173
00174 /* The state buf must be large enough to hold one state per character in the main buffer.
00175 */
00176 #define YY_STATE_BUF_SIZE   ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
00177
00178 #ifndef YY_TYPEDEF_YY_BUFFER_STATE
00179 #define YY_TYPEDEF_YY_BUFFER_STATE
00180 typedef struct yy_buffer_state *YY_BUFFER_STATE;
00181 #endif
00182
00183 #ifndef YY_TYPEDEF_YY_SIZE_T
00184 #define YY_TYPEDEF_YY_SIZE_T
00185 typedef size_t yy_size_t;
00186 #endif
00187
00188 extern int yyleng;
00189
00190 extern FILE *yyin, *yyout;
00191
00192 #define EOB_ACT_CONTINUE_SCAN 0
00193 #define EOB_ACT_END_OF_FILE 1
00194 #define EOB_ACT_LAST_MATCH 2
00195
00196 #define YY_LESS_LINENO(n)
00197 #define YY_LINENO_REWIND_TO(ptr)
00198
00199 /* Return all but the first "n" matched characters back to the input stream.  */
00200 #define yyless(n) \
00201 do \
00202 { \
00203 /* Undo effects of setting up yytext.  */\
00204 int yyless_macro_arg = (n); \
00205 YY_LESS_LINENO(yyless_macro_arg);\
00206 *yy_cp = (yy_hold_char); \
00207 YY_RESTORE_YY_MORE_OFFSET \
00208 (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
00209 YY_DO_BEFORE_ACTION; /* set up yytext again */\
00210 } \
00211 while ( 0 )
00212 #define unput(c) yyunput( c, (yytext_ptr)  )
00213
00214 #ifndef YY_STRUCT_YY_BUFFER_STATE
00215 #define YY_STRUCT_YY_BUFFER_STATE
00216 struct yy_buffer_state
00217     {
00218     FILE *yy_input_file;
00219
00220     char *yy_ch_buf;
00221
00222     int yy_buf_size;
00223
00224
00225     int yy_n_chars;
00226
00227
00228     int yy_is_our_buffer;
00229
00230
00231     int yy_is_interactive;
00232
00233
00234     int yy_at_bol;
00235
00236     int yy_bs_lineno;
00237     int yy_bs_column;
00238
00239
00240     int yy_fill_buffer;
00241
00242     int yy_buffer_status;
00243
00244 #define YY_BUFFER_NEW 0
00245 #define YY_BUFFER_NORMAL 1
```

```
00246
00247 #define YY_BUFFER_EOF_PENDING 2
00248
00249     };
00250 #endif /* !YY_STRUCT_YY_BUFFER_STATE */
00251
00252 /* Stack of input buffers.  */
00253 static size_t yy_buffer_stack_top = 0;
00254 static size_t yy_buffer_stack_max = 0;
00255 static YY_BUFFER_STATE * yy_buffer_stack = NULL;
00256
00257
00258 #define YY_CURRENT_BUFFER ( (yy_buffer_stack) \
00259 ?  (yy_buffer_stack)[(yy_buffer_stack_top)] \
00260 :  NULL)
00261
00262 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
00263
00264 /* yy_hold_char holds the character lost when yytext is formed.  */
00265 static char yy_hold_char;
00266 static int yy_n_chars;      /* number of characters read into yy_ch_buf */
00267 int yyleng;
00268
00269 /* Points to current character in buffer.  */
00270 static char *yy_c_buf_p = NULL;
00271 static int yy_init = 0;     /* whether we need to initialize */
00272 static int yy_start = 0;    /* start state number */
00273
00274 /* Flag which is used to allow yywrap()'s to do buffer switches
00275 * instead of setting up a fresh yyin.    A bit of a hack ...
00276 */
00277 static int yy_did_buffer_switch_on_eof;
00278
00279 void yyrestart ( FILE *input_file  );
00280 void yy_switch_to_buffer ( YY_BUFFER_STATE new_buffer  );
00281 YY_BUFFER_STATE yy_create_buffer ( FILE *file, int size  );
00282 void yy_delete_buffer ( YY_BUFFER_STATE b  );
00283 void yy_flush_buffer ( YY_BUFFER_STATE b  );
00284 void yypush_buffer_state ( YY_BUFFER_STATE new_buffer  );
00285 void yypop_buffer_state ( void );
00286
00287 static void yyensure_buffer_stack ( void );
00288 static void yy_load_buffer_state ( void );
00289 static void yy_init_buffer ( YY_BUFFER_STATE b, FILE *file  );
00290 #define YY_FLUSH_BUFFER yy_flush_buffer( YY_CURRENT_BUFFER )
00291
00292 YY_BUFFER_STATE yy_scan_buffer ( char *base, yy_size_t size  );
00293 YY_BUFFER_STATE yy_scan_string ( const char *yy_str  );
00294 YY_BUFFER_STATE yy_scan_bytes ( const char *bytes, int len  );
00295
00296 void *yyalloc ( yy_size_t  );
00297 void *yyrealloc ( void *, yy_size_t  );
00298 void yyfree ( void *  );
00299
00300 #define yy_new_buffer yy_create_buffer
00301 #define yy_set_interactive(is_interactive) \
00302 { \
00303 if ( !  YY_CURRENT_BUFFER ){ \
00304 yyensure_buffer_stack (); \
00305 YY_CURRENT_BUFFER_LVALUE =    \
00306 yy_create_buffer( yyin, YY_BUF_SIZE ); \
00307 } \
00308 YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
00309 }
00310 #define yy_set_bol(at_bol) \
00311 { \
00312 if ( !  YY_CURRENT_BUFFER ){\
00313 yyensure_buffer_stack (); \
00314 YY_CURRENT_BUFFER_LVALUE =    \
00315 yy_create_buffer( yyin, YY_BUF_SIZE ); \
00316 } \
00317 YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
00318 }
00319 #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
00320
00321 /* Begin user sect3 */
00322
00323 #define yywrap() (/*CONSTCOND*/1)
00324 #define YY_SKIP_YYWRAP
00325 typedef flex_uint8_t YY_CHAR;
00326
00327 FILE *yyin = NULL, *yyout = NULL;
00328
00329 typedef int yy_state_type;
00330
00331 extern int yylineno;
00332 int yylineno = 1;
```

```
00333
00334 extern char *yytext;
00335 #ifdef yytext_ptr
00336 #undef yytext_ptr
00337 #endif
00338 #define yytext_ptr yytext
00339
00340 static yy_state_type yy_get_previous_state ( void );
00341 static yy_state_type yy_try_NUL_trans ( yy_state_type current_state  );
00342 static int yy_get_next_buffer ( void );
00343 static void yynoreturn yy_fatal_error ( const char* msg  );
00344
00345 /* Done after the current pattern has been matched and before the
00346 * corresponding action - sets up yytext.
00347 */
00348 #define YY_DO_BEFORE_ACTION \
00349 (yytext_ptr) = yy_bp; \
00350 yyleng = (int) (yy_cp - yy_bp); \
00351 (yy_hold_char) = *yy_cp; \
00352 *yy_cp = '\0'; \
00353 (yy_c_buf_p) = yy_cp;
00354 #define YY_NUM_RULES 17
00355 #define YY_END_OF_BUFFER 18
00356 /* This struct is not used in this scanner,
00357 but its presence is necessary.  */
00358 struct yy_trans_info
00359     {
00360     flex_int32_t yy_verify;
00361     flex_int32_t yy_nxt;
00362     };
00363 static const flex_int16_t yy_accept[36] =
00364     {   0,
00365         0,    0,    0,    0,    0,    0,   12,   12,   15,   15,
00366        18,    5,    1,    2,    5,    7,    8,    7,   10,   11,
00367        10,   12,   17,   15,   16,    3,    4,    6,    9,   12,
00368        13,   13,   14,   15,    0
00369     } ;
00370
00371 static const YY_CHAR yy_ec[256] =
00372     {   0,
00373         1,    1,    1,    1,    1,    1,    1,    1,    1,    2,
00374         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00375         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00376         1,    1,    1,    3,    1,    1,    1,    1,    4,    1,
00377         1,    5,    1,    1,    1,    1,    6,    1,    1,    1,
00378         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00379         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00380         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00381         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00382         1,    7,    1,    1,    1,    1,    1,    1,    1,    1,
00383
00384         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00385         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00386         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00387         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00388         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00389         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00390         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00391         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00392         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00393         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00394
00395         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00396         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00397         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00398         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00399         1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
00400         1,    1,    1,    1,    1
00401     } ;
00402
00403 static const YY_CHAR yy_meta[8] =
00404     {   0,
00405         1,    2,    1,    1,    3,    1,    1
00406     } ;
00407
00408 static const flex_int16_t yy_base[45] =
00409     {   0,
00410         0,    0,    5,   10,   14,   18,   10,    9,    9,    8,
00411         9,   58,   58,   58,   21,   58,   58,    0,   58,   58,
00412         0,    0,   23,    0,   58,   58,   58,   58,   58,    0,
00413        58,   25,   58,    0,   58,   31,   34,   37,   40,   43,
00414        46,   49,   51,   54
00415     } ;
00416
00417 static const flex_int16_t yy_def[45] =
00418     {   0,
00419        35,    1,   36,   36,   37,   37,   38,   38,   39,   39,
```

```
00420        35,   35,   35,   35,   35,   35,   35,   40,   35,   35,
00421        41,   42,   43,   44,   35,   35,   35,   35,   35,   42,
00422        35,   43,   35,   44,    0,   35,   35,   35,   35,   35,
00423        35,   35,   35,   35
00424     } ;
00425
00426 static const flex_int16_t yy_nxt[66] =
00427     {   0,
00428        12,   12,   13,   14,   12,   15,   12,   17,   35,   25,
00429        25,   18,   17,   23,   23,   35,   18,   20,   35,   35,
00430        21,   20,   35,   35,   21,   26,   27,   32,   33,   32,
00431        33,   16,   16,   16,   19,   19,   19,   22,   22,   22,
00432        24,   24,   24,   28,   35,   28,   29,   35,   29,   30,
00433        30,   31,   31,   31,   34,   35,   34,   11,   35,   35,
00434        35,   35,   35,   35,   35
00435     } ;
00436
00437 static const flex_int16_t yy_chk[66] =
00438     {   0,
00439         1,    1,    1,    1,    1,    1,    1,    3,   11,   10,
00440         9,    3,    4,    8,    7,    0,    4,    5,    0,    0,
00441         5,    6,    0,    0,    6,   15,   15,   23,   23,   32,
00442        32,   36,   36,   36,   37,   37,   37,   38,   38,   38,
00443        39,   39,   39,   40,    0,   40,   41,    0,   41,   42,
00444        42,   43,   43,   43,   44,    0,   44,   35,   35,   35,
00445        35,   35,   35,   35,   35
00446     } ;
00447
00448 static yy_state_type yy_last_accepting_state;
00449 static char *yy_last_accepting_cpos;
00450
00451 extern int yy_flex_debug;
00452 int yy_flex_debug = 0;
00453
00454 /* The intent behind this definition is that it'll catch
00455  * any uses of REJECT which flex missed.
00456  */
00457 #define REJECT reject_used_but_not_detected
00458 #define yymore() yymore_used_but_not_detected
00459 #define YY_MORE_ADJ 0
00460 #define YY_RESTORE_YY_MORE_OFFSET
00461 char *yytext;
00462 #line 1 "abc2.l"
00463 #line 4 "abc2.l"
00464 #include <string.h>
00465 #include <errno.h>
00466 #include "../fileMan.h"
00467 #include "lex_yy.h"
00468 /* remove C comments */
00469
00470 /* replace each comment with a single blank */
00471
00472 extern int fileno(FILE *);      /* avoid gcc warning */
00473 #line 476 "lex.yy.c"
00474
00475 #line 478 "lex.yy.c"
00476
00477 #define INITIAL 0
00478 #define String 1
00479 #define Char 2
00480 #define Comment 3
00481 #define CPPComment 4
00482
00483 #ifndef YY_NO_UNISTD_H
00484 /* Special case for "unistd.h", since it is non-ANSI. We include it way
00485  * down here because we want the user's section 1 to have been scanned first.
00486  * The user has a chance to override it with an option.
00487  */
00488 #include <unistd.h>
00489 #endif
00490
00491 #ifndef YY_EXTRA_TYPE
00492 #define YY_EXTRA_TYPE void *
00493 #endif
00494
00495 static int yy_init_globals ( void );
00496
00497 /* Accessor methods to globals.
00498 These are made visible to non-reentrant scanners for convenience.  */
00499
00500 int yylex_destroy ( void );
00501
00502 int yyget_debug ( void );
00503
00504 void yyset_debug ( int debug_flag  );
00505
00506 YY_EXTRA_TYPE yyget_extra ( void );
```

```
00507
00508 void yyset_extra ( YY_EXTRA_TYPE user_defined  );
00509
00510 FILE *yyget_in ( void );
00511
00512 void yyset_in  ( FILE * _in_str  );
00513
00514 FILE *yyget_out ( void );
00515
00516 void yyset_out  ( FILE * _out_str  );
00517
00518             int yyget_leng ( void );
00519
00520 char *yyget_text ( void );
00521
00522 int yyget_lineno ( void );
00523
00524 void yyset_lineno ( int _line_number  );
00525
00526 /* Macros after this point can all be overridden by user definitions in
00527 * section 1.
00528 */
00529
00530 #ifndef YY_SKIP_YYWRAP
00531 #ifdef __cplusplus
00532 extern "C" int yywrap ( void );
00533 #else
00534 extern int yywrap ( void );
00535 #endif
00536 #endif
00537
00538 #ifndef YY_NO_UNPUT
00539
00540     static void yyunput ( int c, char *buf_ptr  );
00541
00542 #endif
00543
00544 #ifndef yytext_ptr
00545 static void yy_flex_strncpy ( char *, const char *, int );
00546 #endif
00547
00548 #ifdef YY_NEED_STRLEN
00549 static int yy_flex_strlen ( const char * );
00550 #endif
00551
00552 #ifndef YY_NO_INPUT
00553 #ifdef __cplusplus
00554 static int yyinput ( void );
00555 #else
00556 static int input ( void );
00557 #endif
00558
00559 #endif
00560
00561 /* Amount of stuff to slurp up with each read.  */
00562 #ifndef YY_READ_BUF_SIZE
00563 #ifdef __ia64__
00564 /* On IA-64, the buffer size is 16k, not 8k */
00565 #define YY_READ_BUF_SIZE 16384
00566 #else
00567 #define YY_READ_BUF_SIZE 8192
00568 #endif /* __ia64__ */
00569 #endif
00570
00571 /* Copy whatever the last rule matched to the standard output.  */
00572 #ifndef ECHO
00573 /* This used to be an fputs(), but since the string might contain NUL's,
00574 * we now use fwrite().
00575 */
00576 #define ECHO do { if (fwrite( yytext, (size_t) yyleng, 1, yyout )) {} } while (0)
00577 #endif
00578
00579 /* Gets input and stuffs it into "buf".    number of characters read, or YY_NULL,
00580 * is returned in "result".
00581 */
00582 #ifndef YY_INPUT
00583 #define YY_INPUT(buf,result,max_size) \
00584 if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
00585 { \
00586 int c = '*'; \
00587 int n; \
00588 for ( n = 0; n < max_size && \
00589 (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
00590 buf[n] = (char) c; \
00591 if ( c == '\n' ) \
00592 buf[n++] = (char) c; \
00593 if ( c == EOF && ferror( yyin ) ) \
```

```
00594 YY_FATAL_ERROR( "input in flex scanner failed" ); \
00595         result = n; \
00596         } \
00597     else \
00598         { \
00599         errno=0; \
00600         while ( (result = (int) fread(buf, 1, (yy_size_t) max_size, yyin)) == 0 && ferror(yyin)) \
00601             { \
00602             if( errno != EINTR) \
00603                 { \
00604                 YY_FATAL_ERROR( "input in flex scanner failed" ); \
00605                 break; \
00606                 } \
00607             errno=0; \
00608             clearerr(yyin); \
00609             } \
00610         }\
00611 \
00612
00613 #endif
00614
00615 /* No semi-colon after return; correct usage is to write "yyterminate();" -
00616 * we don't want an extra ';' after the "return" because that will cause
00617 * some compilers to complain about unreachable statements.
00618 */
00619 #ifndef yyterminate
00620 #define yyterminate() return YY_NULL
00621 #endif
00622
00623 /* Number of entries by which start-condition stack grows.  */
00624 #ifndef YY_START_STACK_INCR
00625 #define YY_START_STACK_INCR 25
00626 #endif
00627
00628 /* Report a fatal error.  */
00629 #ifndef YY_FATAL_ERROR
00630 #define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
00631 #endif
00632
00633 /* end tables serialization structures and prototypes */
00634
00635 /* Default declaration of generated scanner - a define so the user can
00636 * easily add parameters.
00637 */
00638 #ifndef YY_DECL
00639 #define YY_DECL_IS_OURS 1
00640
00641 extern int yylex (void);
00642
00643 #define YY_DECL int yylex (void)
00644 #endif /* !YY_DECL */
00645
00646 /* Code executed at the beginning of each rule, after yytext and yyleng
00647 * have been set up.
00648 */
00649 #ifndef YY_USER_ACTION
00650 #define YY_USER_ACTION
00651 #endif
00652
00653 /* Code executed at the end of each rule.  */
00654 #ifndef YY_BREAK
00655 #define YY_BREAK /*LINTED*/break;
00656 #endif
00657
00658 #define YY_RULE_SETUP \
00659 YY_USER_ACTION
00660
00661 /* The main scanner function which does all the work.
00662 */
00663 YY_DECL
00664 {
00665     yy_state_type yy_current_state;
00666     char *yy_cp, *yy_bp;
00667     int yy_act;
00668
00669     if ( !(yy_init) )
00670         {
00671         (yy_init) = 1;
00672
00673 #ifdef YY_USER_INIT
00674         YY_USER_INIT;
00675 #endif
00676
00677         if ( !  (yy_start) )
00678             (yy_start) = 1; /* first start state */
00679
00680         if ( !  yyin )
```

```
00681             yyin = stdin;
00682
00683         if ( !  yyout )
00684             yyout = stdout;
00685
00686         if ( !  YY_CURRENT_BUFFER ) {
00687             yyensure_buffer_stack ();
00688             YY_CURRENT_BUFFER_LVALUE =
00689                 yy_create_buffer( yyin, YY_BUF_SIZE );
00690         }
00691
00692         yy_load_buffer_state(  );
00693         }
00694
00695     {
00696 #line 17 "abc2.l"
00697
00698
00699 #line 702 "lex.yy.c"
00700
00701     while ( /*CONSTCOND*/1 )        /* loops until end-of-file is reached */
00702         {
00703         yy_cp = (yy_c_buf_p);
00704
00705         /* Support of yytext.  */
00706         *yy_cp = (yy_hold_char);
00707
00708         /* yy_bp points to the position in yy_ch_buf of the start of
00709 * the current run.
00710 */
00711         yy_bp = yy_cp;
00712
00713         yy_current_state = (yy_start);
00714 yy_match:
00715         do
00716             {
00717             YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)] ;
00718             if ( yy_accept[yy_current_state] )
00719                 {
00720                 (yy_last_accepting_state) = yy_current_state;
00721                 (yy_last_accepting_cpos) = yy_cp;
00722                 }
00723             while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
00724                 {
00725                 yy_current_state = (int) yy_def[yy_current_state];
00726                 if ( yy_current_state >= 36 )
00727                     yy_c = yy_meta[yy_c];
00728                 }
00729             yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
00730             ++yy_cp;
00731             }
00732         while ( yy_base[yy_current_state] != 58 );
00733
00734 yy_find_action:
00735         yy_act = yy_accept[yy_current_state];
00736         if ( yy_act == 0 )
00737             { /* have to back up */
00738             yy_cp = (yy_last_accepting_cpos);
00739             yy_current_state = (yy_last_accepting_state);
00740             yy_act = yy_accept[yy_current_state];
00741             }
00742
00743         YY_DO_BEFORE_ACTION;
00744
00745 do_action:    /* This label is used only to access EOF actions.  */
00746
00747         switch ( yy_act )
00748     { /* beginning of action switch */
00749             case 0:  /* must back up */
00750             /* undo the effects of YY_DO_BEFORE_ACTION */
00751             *yy_cp = (yy_hold_char);
00752             yy_cp = (yy_last_accepting_cpos);
00753             yy_current_state = (yy_last_accepting_state);
00754             goto yy_find_action;
00755
00756 case 1:
00757 YY_RULE_SETUP
00758 #line 19 "abc2.l"
00759 /* "String" */ ECHO; BEGIN( String);
00760     YY_BREAK
00761 case 2:
00762 YY_RULE_SETUP
00763 #line 21 "abc2.l"
00764 /* 'Char' */ ECHO; BEGIN( Char);
00765     YY_BREAK
00766 case 3:
00767 YY_RULE_SETUP
```

```
00768 #line 23 "abc2.l"
00769 /* comment */ BEGIN( Comment);
00770     YY_BREAK
00771 case 4:
00772 YY_RULE_SETUP
00773 #line 25 "abc2.l"
00774 /* C++ comment */ BEGIN( CPPComment);
00775     YY_BREAK
00776 case 5:
00777 /* rule 5 can match eol */
00778 YY_RULE_SETUP
00779 #line 27 "abc2.l"
00780 /* everything else */ ECHO;
00781     YY_BREAK
00782
00783 case 6:
00784 YY_RULE_SETUP
00785 #line 30 "abc2.l"
00786 /* escape sequence */ ECHO;
00787     YY_BREAK
00788 case 7:
00789 /* rule 7 can match eol */
00790 YY_RULE_SETUP
00791 #line 31 "abc2.l"
00792 /* not "quote" */ ECHO;
00793     YY_BREAK
00794 case 8:
00795 YY_RULE_SETUP
00796 #line 32 "abc2.l"
00797 /* end of "String" */ ECHO; BEGIN( INITIAL);
00798     YY_BREAK
00799
00800
00801 case 9:
00802 YY_RULE_SETUP
00803 #line 36 "abc2.l"
00804 /* escape sequence */ ECHO;
00805     YY_BREAK
00806 case 10:
00807 /* rule 10 can match eol */
00808 YY_RULE_SETUP
00809 #line 37 "abc2.l"
00810 /* not 'quote' */ ECHO;
00811     YY_BREAK
00812 case 11:
00813 YY_RULE_SETUP
00814 #line 38 "abc2.l"
00815 /* end of 'Char' */ ECHO; BEGIN( INITIAL);
00816     YY_BREAK
00817
00818
00819 case 12:
00820 /* rule 12 can match eol */
00821 YY_RULE_SETUP
00822 #line 42 "abc2.l"
00823 /* not a '*' */
00824     YY_BREAK
00825 case 13:
00826 /* rule 13 can match eol */
00827 YY_RULE_SETUP
00828 #line 43 "abc2.l"
00829 /* '*'s not followed by '/' */
00830     YY_BREAK
00831 case 14:
00832 YY_RULE_SETUP
00833 #line 44 "abc2.l"
00834 /* end of Comment */ putchar( ' '); BEGIN(INITIAL);
00835     YY_BREAK
00836
00837
00838 case 15:
00839 YY_RULE_SETUP
00840 #line 48 "abc2.l"
00841 /* to end of line */
00842     YY_BREAK
00843 case 16:
00844 /* rule 16 can match eol */
00845 YY_RULE_SETUP
00846 #line 49 "abc2.l"
00847 ECHO; BEGIN( INITIAL);
00848     YY_BREAK
00849
00850 case 17:
00851 YY_RULE_SETUP
00852 #line 52 "abc2.l"
00853 ECHO;
00854     YY_BREAK
```

```
00855 #line 858 "lex.yy.c"
00856 case YY_STATE_EOF(INITIAL):
00857 case YY_STATE_EOF(String):
00858 case YY_STATE_EOF(Char):
00859 case YY_STATE_EOF(Comment):
00860 case YY_STATE_EOF(CPPComment):
00861     yyterminate();
00862
00863     case YY_END_OF_BUFFER:
00864         {
00865         /* Amount of text matched not including the EOB char.  */
00866         int yy_amount_of_matched_text = (int) (yy_cp - (yytext_ptr)) - 1;
00867
00868         /* Undo the effects of YY_DO_BEFORE_ACTION. */
00869         *yy_cp = (yy_hold_char);
00870         YY_RESTORE_YY_MORE_OFFSET
00871
00872         if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_NEW )
00873             {
00874             /* We're scanning a new file or input source.    It's
00875 * possible that this happened because the user
00876 * just pointed yyin at a new source and called
00877 * yylex().    If so, then we have to assure
00878 * consistency between YY_CURRENT_BUFFER and our
00879 * globals.    Here is the right place to do so, because
00880 * this is the first action (other than possibly a
00881 * back-up) that will match for the new input source.
00882 */
00883             (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
00884             YY_CURRENT_BUFFER_LVALUE->yy_input_file = yyin;
00885             YY_CURRENT_BUFFER_LVALUE->yy_buffer_status = YY_BUFFER_NORMAL;
00886             }
00887
00888         /* Note that here we test for yy_c_buf_p "<=" to the position
00889 * of the first EOB in the buffer, since yy_c_buf_p will
00890 * already have been incremented past the NUL character
00891 * (since all states make transitions on EOB to the
00892 * end-of-buffer state).    Contrast this with the test
00893 * in input().
00894 */
00895         if ( (yy_c_buf_p) <= &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
00896             { /* This was really a NUL. */
00897             yy_state_type yy_next_state;
00898
00899             (yy_c_buf_p) = (yytext_ptr) + yy_amount_of_matched_text;
00900
00901             yy_current_state = yy_get_previous_state(  );
00902
00903             /* Okay, we're now positioned to make the NUL
00904 * transition.    We couldn't have
00905 * yy_get_previous_state() go ahead and do it
00906 * for us because it doesn't know how to deal
00907 * with the possibility of jamming (and we don't
00908 * want to build jamming into it because then it
00909 * will run more slowly).
00910 */
00911
00912             yy_next_state = yy_try_NUL_trans( yy_current_state );
00913
00914             yy_bp = (yytext_ptr) + YY_MORE_ADJ;
00915
00916             if ( yy_next_state )
00917                 {
00918                 /* Consume the NUL. */
00919                 yy_cp = ++(yy_c_buf_p);
00920                 yy_current_state = yy_next_state;
00921                 goto yy_match;
00922                 }
00923
00924             else
00925                 {
00926                 yy_cp = (yy_c_buf_p);
00927                 goto yy_find_action;
00928                 }
00929             }
00930
00931         else switch ( yy_get_next_buffer(  ) )
00932             {
00933             case EOB_ACT_END_OF_FILE:
00934                 {
00935                 (yy_did_buffer_switch_on_eof) = 0;
00936
00937                 if ( yywrap(  ) )
00938                     {
00939                     /* Note:  because we've taken care in
00940 * yy_get_next_buffer() to have set up
00941 * yytext, we can now set up
```

```
00942 * yy_c_buf_p so that if some total
00943 * hoser (like flex itself) wants to
00944 * call the scanner after we return the
00945 * YY_NULL, it'll still work - another
00946 * YY_NULL will get returned.
00947 */
00948                     (yy_c_buf_p) = (yytext_ptr) + YY_MORE_ADJ;
00949
00950                     yy_act = YY_STATE_EOF(YY_START);
00951                     goto do_action;
00952                     }
00953
00954                 else
00955                     {
00956                     if ( !  (yy_did_buffer_switch_on_eof) )
00957                         YY_NEW_FILE;
00958                     }
00959                 break;
00960                 }
00961
00962             case EOB_ACT_CONTINUE_SCAN:
00963                 (yy_c_buf_p) =
00964                     (yytext_ptr) + yy_amount_of_matched_text;
00965
00966                 yy_current_state = yy_get_previous_state(  );
00967
00968                 yy_cp = (yy_c_buf_p);
00969                 yy_bp = (yytext_ptr) + YY_MORE_ADJ;
00970                 goto yy_match;
00971
00972             case EOB_ACT_LAST_MATCH:
00973                 (yy_c_buf_p) =
00974                 &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)];
00975
00976                 yy_current_state = yy_get_previous_state(  );
00977
00978                 yy_cp = (yy_c_buf_p);
00979                 yy_bp = (yytext_ptr) + YY_MORE_ADJ;
00980                 goto yy_find_action;
00981             }
00982         break;
00983         }
00984
00985     default:
00986         YY_FATAL_ERROR(
00987             "fatal flex scanner internal error--no action found" );
00988     } /* end of action switch */
00989         } /* end of scanning one token */
00990     } /* end of user's declarations */
00991 } /* end of yylex */
00992
00993 /* yy_get_next_buffer - try to read in a new buffer
00994 *
00995 * Returns a code representing an action:
00996 *  EOB_ACT_LAST_MATCH -
00997 *  EOB_ACT_CONTINUE_SCAN - continue scanning from current position
00998 *  EOB_ACT_END_OF_FILE - end of file
00999 */
01000 static int yy_get_next_buffer (void)
01001 {
01002         char *dest = YY_CURRENT_BUFFER_LVALUE->yy_ch_buf;
01003     char *source = (yytext_ptr);
01004     int number_to_move, i;
01005     int ret_val;
01006
01007     if ( (yy_c_buf_p) > &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] )
01008         YY_FATAL_ERROR(
01009         "fatal flex scanner internal error--end of buffer missed" );
01010
01011     if ( YY_CURRENT_BUFFER_LVALUE->yy_fill_buffer == 0 )
01012         { /* Don't try to fill the buffer, so this is an EOF. */
01013         if ( (yy_c_buf_p) - (yytext_ptr) - YY_MORE_ADJ == 1 )
01014             {
01015             /* We matched a single character, the EOB, so
01016 * treat this as a final EOF.
01017 */
01018             return EOB_ACT_END_OF_FILE;
01019             }
01020
01021         else
01022             {
01023             /* We matched some text prior to the EOB, first
01024 * process it.
01025 */
01026             return EOB_ACT_LAST_MATCH;
01027             }
01028         }
```

```
01029
01030     /* Try to read more data.  */
01031
01032     /* First move last chars to start of buffer.  */
01033     number_to_move = (int) ((yy_c_buf_p) - (yytext_ptr) - 1);
01034
01035     for ( i = 0; i < number_to_move; ++i )
01036         *(dest++) = *(source++);
01037
01038     if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_EOF_PENDING )
01039         /* don't do the read, it's not guaranteed to return an EOF,
01040 * just force an EOF
01041 */
01042         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars) = 0;
01043
01044     else
01045         {
01046             int num_to_read =
01047             YY_CURRENT_BUFFER_LVALUE->yy_buf_size - number_to_move - 1;
01048
01049         while ( num_to_read <= 0 )
01050             { /* Not enough room in the buffer - grow it.  */
01051
01052             /* just a shorter name for the current buffer */
01053             YY_BUFFER_STATE b = YY_CURRENT_BUFFER_LVALUE;
01054
01055             int yy_c_buf_p_offset =
01056                 (int) ((yy_c_buf_p) - b->yy_ch_buf);
01057
01058             if ( b->yy_is_our_buffer )
01059                 {
01060                 int new_size = b->yy_buf_size * 2;
01061
01062                 if ( new_size <= 0 )
01063                     b->yy_buf_size += b->yy_buf_size / 8;
01064                 else
01065                     b->yy_buf_size *= 2;
01066
01067                 b->yy_ch_buf = (char *)
01068                     /* Include room in for 2 EOB chars.  */
01069                     yyrealloc( (void *) b->yy_ch_buf,
01070                               (yy_size_t) (b->yy_buf_size + 2)  );
01071                 }
01072             else
01073                 /* Can't grow it, we don't own it.  */
01074                 b->yy_ch_buf = NULL;
01075
01076             if ( !  b->yy_ch_buf )
01077                 YY_FATAL_ERROR(
01078                 "fatal error - scanner input buffer overflow" );
01079
01080             (yy_c_buf_p) = &b->yy_ch_buf[yy_c_buf_p_offset];
01081
01082             num_to_read = YY_CURRENT_BUFFER_LVALUE->yy_buf_size -
01083                         number_to_move - 1;
01084
01085             }
01086
01087         if ( num_to_read > YY_READ_BUF_SIZE )
01088             num_to_read = YY_READ_BUF_SIZE;
01089
01090         /* Read in more data.  */
01091         YY_INPUT( (&YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[number_to_move]),
01092             (yy_n_chars), num_to_read );
01093
01094         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
01095         }
01096
01097     if ( (yy_n_chars) == 0 )
01098         {
01099         if ( number_to_move == YY_MORE_ADJ )
01100             {
01101             ret_val = EOB_ACT_END_OF_FILE;
01102             yyrestart( yyin  );
01103             }
01104
01105         else
01106             {
01107             ret_val = EOB_ACT_LAST_MATCH;
01108             YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
01109                 YY_BUFFER_EOF_PENDING;
01110             }
01111         }
01112
01113     else
01114         ret_val = EOB_ACT_CONTINUE_SCAN;
01115
```

```
01116     if (((yy_n_chars) + number_to_move) > YY_CURRENT_BUFFER_LVALUE->yy_buf_size) {
01117         /* Extend the array by 50%, plus the number we really need.  */
01118         int new_size = (yy_n_chars) + number_to_move + ((yy_n_chars) >> 1);
01119         YY_CURRENT_BUFFER_LVALUE->yy_ch_buf = (char *) yyrealloc(
01120             (void *) YY_CURRENT_BUFFER_LVALUE->yy_ch_buf, (yy_size_t) new_size  );
01121         if ( !  YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
01122             YY_FATAL_ERROR( "out of dynamic memory in yy_get_next_buffer()" );
01123         /* "- 2" to take care of EOB's */
01124         YY_CURRENT_BUFFER_LVALUE->yy_buf_size = (int) (new_size - 2);
01125     }
01126
01127     (yy_n_chars) += number_to_move;
01128     YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] = YY_END_OF_BUFFER_CHAR;
01129     YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] = YY_END_OF_BUFFER_CHAR;
01130
01131     (yytext_ptr) = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[0];
01132
01133     return ret_val;
01134 }
01135
01136 /* yy_get_previous_state - get the state just before the EOB char was reached */
01137
01138     static yy_state_type yy_get_previous_state (void)
01139 {
01140     yy_state_type yy_current_state;
01141     char *yy_cp;
01142
01143     yy_current_state = (yy_start);
01144
01145     for ( yy_cp = (yytext_ptr) + YY_MORE_ADJ; yy_cp < (yy_c_buf_p); ++yy_cp )
01146         {
01147         YY_CHAR yy_c = (*yy_cp ?  yy_ec[YY_SC_TO_UI(*yy_cp)] :  1);
01148         if ( yy_accept[yy_current_state] )
01149             {
01150             (yy_last_accepting_state) = yy_current_state;
01151             (yy_last_accepting_cpos) = yy_cp;
01152             }
01153         while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
01154             {
01155             yy_current_state = (int) yy_def[yy_current_state];
01156             if ( yy_current_state >= 36 )
01157                 yy_c = yy_meta[yy_c];
01158             }
01159         yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
01160         }
01161
01162     return yy_current_state;
01163 }
01164
01165 /* yy_try_NUL_trans - try to make a transition on the NUL character
01166 *
01167 * synopsis
01168 *  next_state = yy_try_NUL_trans( current_state );
01169 */
01170     static yy_state_type yy_try_NUL_trans  (yy_state_type yy_current_state )
01171 {
01172     int yy_is_jam;
01173         char *yy_cp = (yy_c_buf_p);
01174
01175     YY_CHAR yy_c = 1;
01176     if ( yy_accept[yy_current_state] )
01177         {
01178         (yy_last_accepting_state) = yy_current_state;
01179         (yy_last_accepting_cpos) = yy_cp;
01180         }
01181     while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
01182         {
01183         yy_current_state = (int) yy_def[yy_current_state];
01184         if ( yy_current_state >= 36 )
01185             yy_c = yy_meta[yy_c];
01186         }
01187     yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
01188     yy_is_jam = (yy_current_state == 35);
01189
01190         return yy_is_jam ?  0 :  yy_current_state;
01191 }
01192
01193 #ifndef YY_NO_UNPUT
01194
01195     static void yyunput (int c, char * yy_bp )
01196 {
01197     char *yy_cp;
01198
01199     yy_cp = (yy_c_buf_p);
01200
01201     /* undo effects of setting up yytext */
01202     *yy_cp = (yy_hold_char);
```

```
01203
01204     if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
01205         { /* need to shift things up to make room */
01206         /* +2 for EOB chars.  */
01207         int number_to_move = (yy_n_chars) + 2;
01208         char *dest = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[
01209                         YY_CURRENT_BUFFER_LVALUE->yy_buf_size + 2];
01210         char *source =
01211                 &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[number_to_move];
01212
01213         while ( source > YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
01214             *--dest = *--source;
01215
01216         yy_cp += (int) (dest - source);
01217         yy_bp += (int) (dest - source);
01218         YY_CURRENT_BUFFER_LVALUE->yy_n_chars =
01219             (yy_n_chars) = (int) YY_CURRENT_BUFFER_LVALUE->yy_buf_size;
01220
01221         if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
01222             YY_FATAL_ERROR( "flex scanner push-back overflow" );
01223         }
01224
01225     *--yy_cp = (char) c;
01226
01227     (yytext_ptr) = yy_bp;
01228     (yy_hold_char) = *yy_cp;
01229     (yy_c_buf_p) = yy_cp;
01230 }
01231
01232 #endif
01233
01234 #ifndef YY_NO_INPUT
01235 #ifdef __cplusplus
01236     static int yyinput (void)
01237 #else
01238     static int input  (void)
01239 #endif
01240
01241 {
01242     int c;
01243
01244     *(yy_c_buf_p) = (yy_hold_char);
01245
01246     if ( *(yy_c_buf_p) == YY_END_OF_BUFFER_CHAR )
01247         {
01248         /* yy_c_buf_p now points to the character we want to return.
01249 * If this occurs *before* the EOB characters, then it's a
01250 * valid NUL; if not, then we've hit the end of the buffer.
01251 */
01252         if ( (yy_c_buf_p) < &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
01253             /* This was really a NUL. */
01254             *(yy_c_buf_p) = '\0';
01255
01256         else
01257             { /* need more input */
01258             int offset = (int) ((yy_c_buf_p) - (yytext_ptr));
01259             ++(yy_c_buf_p);
01260
01261             switch ( yy_get_next_buffer(  ) )
01262                 {
01263                 case EOB_ACT_LAST_MATCH:
01264                     /* This happens because yy_g_n_b()
01265 * sees that we've accumulated a
01266 * token and flags that we need to
01267 * try matching the token before
01268 * proceeding.    But for input(),
01269 * there's no matching to consider.
01270 * So convert the EOB_ACT_LAST_MATCH
01271 * to EOB_ACT_END_OF_FILE.
01272 */
01273
01274                     /* Reset buffer status.  */
01275                     yyrestart( yyin );
01276
01277                     /*FALLTHROUGH*/
01278
01279                 case EOB_ACT_END_OF_FILE:
01280                     {
01281                     if ( yywrap(  ) )
01282                         return 0;
01283
01284                     if ( !  (yy_did_buffer_switch_on_eof) )
01285                         YY_NEW_FILE;
01286 #ifdef __cplusplus
01287                     return yyinput();
01288 #else
01289                     return input();
```

```
01290 #endif
01291                     }
01292
01293                 case EOB_ACT_CONTINUE_SCAN:
01294                     (yy_c_buf_p) = (yytext_ptr) + offset;
01295                     break;
01296                 }
01297             }
01298         }
01299
01300     c = *(unsigned char *) (yy_c_buf_p);    /* cast for 8-bit char's */
01301     *(yy_c_buf_p) = '\0';   /* preserve yytext */
01302     (yy_hold_char) = *++(yy_c_buf_p);
01303
01304     return c;
01305 }
01306 #endif  /* ifndef YY_NO_INPUT */
01307
01308 /* Immediately switch to a different input stream.
01309 * @param input_file A readable stream.
01310 *
01311 * @note This function does not reset the start condition to @c INITIAL .
01312 */
01313     void yyrestart  (FILE * input_file )
01314 {
01315
01316     if ( !  YY_CURRENT_BUFFER ){
01317         yyensure_buffer_stack ();
01318         YY_CURRENT_BUFFER_LVALUE =
01319             yy_create_buffer( yyin, YY_BUF_SIZE );
01320     }
01321
01322     yy_init_buffer( YY_CURRENT_BUFFER, input_file );
01323     yy_load_buffer_state(  );
01324 }
01325
01326 /* Switch to a different input buffer.
01327 * @param new_buffer The new input buffer.
01328 *
01329 */
01330     void yy_switch_to_buffer  (YY_BUFFER_STATE  new_buffer )
01331 {
01332
01333     /* TODO. We should be able to replace this entire function body
01334 * with
01335 *      yypop_buffer_state();
01336 *      yypush_buffer_state(new_buffer);
01337 */
01338     yyensure_buffer_stack ();
01339     if ( YY_CURRENT_BUFFER == new_buffer )
01340         return;
01341
01342     if ( YY_CURRENT_BUFFER )
01343         {
01344         /* Flush out information for old buffer.  */
01345         *(yy_c_buf_p) = (yy_hold_char);
01346         YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
01347         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
01348         }
01349
01350     YY_CURRENT_BUFFER_LVALUE = new_buffer;
01351     yy_load_buffer_state(  );
01352
01353     /* We don't actually know whether we did this switch during
01354 * EOF (yywrap()) processing, but the only time this flag
01355 * is looked at is after yywrap() is called, so it's safe
01356 * to go ahead and always set it.
01357 */
01358     (yy_did_buffer_switch_on_eof) = 1;
01359 }
01360
01361 static void yy_load_buffer_state  (void)
01362 {
01363         (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
01364     (yytext_ptr) = (yy_c_buf_p) = YY_CURRENT_BUFFER_LVALUE->yy_buf_pos;
01365     yyin = YY_CURRENT_BUFFER_LVALUE->yy_input_file;
01366     (yy_hold_char) = *(yy_c_buf_p);
01367 }
01368
01369
01370     YY_BUFFER_STATE yy_create_buffer  (FILE * file, int  size )
01371 {
01372     YY_BUFFER_STATE b;
01373
01374     b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
01375     if ( !  b )
01376         YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );
```

```
01377
01378     b->yy_buf_size = size;
01379
01380     /* yy_ch_buf has to be 2 characters longer than the size given because
01381  * we need to put in 2 end-of-buffer characters.
01382  */
01383     b->yy_ch_buf = (char *) yyalloc( (yy_size_t) (b->yy_buf_size + 2)  );
01384     if ( !  b->yy_ch_buf )
01385         YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );
01386
01387     b->yy_is_our_buffer = 1;
01388
01389     yy_init_buffer( b, file );
01390
01391     return b;
01392 }
01393
01394 /* Destroy the buffer.
01395  * @param b a buffer created with yy_create_buffer()
01396  *
01397  */
01398     void yy_delete_buffer (YY_BUFFER_STATE  b )
01399 {
01400
01401     if ( !  b )
01402         return;
01403
01404     if ( b == YY_CURRENT_BUFFER ) /* Not sure if we should pop here.  */
01405         YY_CURRENT_BUFFER_LVALUE = (YY_BUFFER_STATE) 0;
01406
01407     if ( b->yy_is_our_buffer )
01408         yyfree( (void *) b->yy_ch_buf  );
01409
01410     yyfree( (void *) b  );
01411 }
01412
01413 /* Initializes or reinitializes a buffer.
01414  * This function is sometimes called more than once on the same buffer,
01415  * such as during a yyrestart() or at EOF.
01416  */
01417     static void yy_init_buffer  (YY_BUFFER_STATE  b, FILE * file )
01418
01419 {
01420     int oerrno = errno;
01421
01422     yy_flush_buffer( b );
01423
01424     b->yy_input_file = file;
01425     b->yy_fill_buffer = 1;
01426
01427     /* If b is the current buffer, then yy_init_buffer was _probably_
01428  * called from yyrestart() or through yy_get_next_buffer.
01429  * In that case, we don't want to reset the lineno or column.
01430  */
01431     if (b != YY_CURRENT_BUFFER){
01432         b->yy_bs_lineno = 1;
01433         b->yy_bs_column = 0;
01434     }
01435
01436         b->yy_is_interactive = file ?  (isatty( fileno(file) ) > 0) :  0;
01437
01438     errno = oerrno;
01439 }
01440
01441 /* Discard all buffered characters.  On the next scan, YY_INPUT will be called.
01442  * @param b the buffer state to be flushed, usually @c YY_CURRENT_BUFFER.
01443  *
01444  */
01445     void yy_flush_buffer (YY_BUFFER_STATE  b )
01446 {
01447         if ( !  b )
01448         return;
01449
01450     b->yy_n_chars = 0;
01451
01452     /* We always need two end-of-buffer characters.    The first causes
01453  * a transition to the end-of-buffer state.    The second causes
01454  * a jam in that state.
01455  */
01456     b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
01457     b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;
01458
01459     b->yy_buf_pos = &b->yy_ch_buf[0];
01460
01461     b->yy_at_bol = 1;
01462     b->yy_buffer_status = YY_BUFFER_NEW;
01463
```

```
01464     if ( b == YY_CURRENT_BUFFER )
01465         yy_load_buffer_state(  );
01466 }
01467 
01468 /* Pushes the new state onto the stack.  The new state becomes
01469 *  the current state.  This function will allocate the stack
01470 *  if necessary.
01471 *  @param new_buffer The new state.
01472 *
01473 */
01474 void yypush_buffer_state (YY_BUFFER_STATE new_buffer )
01475 {
01476         if (new_buffer == NULL)
01477         return;
01478 
01479     yyensure_buffer_stack();
01480 
01481     /* This block is copied from yy_switch_to_buffer.  */
01482     if ( YY_CURRENT_BUFFER )
01483         {
01484         /* Flush out information for old buffer.  */
01485         *(yy_c_buf_p) = (yy_hold_char);
01486         YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
01487         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
01488         }
01489 
01490     /* Only push if top exists.  Otherwise, replace top.  */
01491     if (YY_CURRENT_BUFFER)
01492         (yy_buffer_stack_top)++;
01493     YY_CURRENT_BUFFER_LVALUE = new_buffer;
01494 
01495     /* copied from yy_switch_to_buffer.  */
01496     yy_load_buffer_state(  );
01497     (yy_did_buffer_switch_on_eof) = 1;
01498 }
01499 
01500 /* Removes and deletes the top of the stack, if present.
01501 *  The next element becomes the new top.
01502 *
01503 */
01504 void yypop_buffer_state (void)
01505 {
01506         if (!YY_CURRENT_BUFFER)
01507         return;
01508 
01509     yy_delete_buffer(YY_CURRENT_BUFFER );
01510     YY_CURRENT_BUFFER_LVALUE = NULL;
01511     if ((yy_buffer_stack_top) > 0)
01512         --(yy_buffer_stack_top);
01513 
01514     if (YY_CURRENT_BUFFER) {
01515         yy_load_buffer_state(  );
01516         (yy_did_buffer_switch_on_eof) = 1;
01517     }
01518 }
01519 
01520 /* Allocates the stack if it does not exist.
01521 *  Guarantees space for at least one push.
01522 */
01523 static void yyensure_buffer_stack (void)
01524 {
01525     yy_size_t num_to_alloc;
01526 
01527     if (!(yy_buffer_stack)) {
01528 
01529         /* First allocation is just for 2 elements, since we don't know if this
01530 * scanner will even need a stack.  We use 2 instead of 1 to avoid an
01531 * immediate realloc on the next call.
01532 */
01533       num_to_alloc = 1; /* After all that talk, this was set to 1 anyways...  */
01534         (yy_buffer_stack) = (struct yy_buffer_state**)yyalloc
01535                                 (num_to_alloc * sizeof(struct yy_buffer_state*)
01536                                 );
01537         if ( !  (yy_buffer_stack) )
01538             YY_FATAL_ERROR( "out of dynamic memory in yyensure_buffer_stack()" );
01539 
01540         memset((yy_buffer_stack), 0, num_to_alloc * sizeof(struct yy_buffer_state*));
01541 
01542         (yy_buffer_stack_max) = num_to_alloc;
01543         (yy_buffer_stack_top) = 0;
01544         return;
01545     }
01546 
01547     if ((yy_buffer_stack_top) >= ((yy_buffer_stack_max)) - 1){
01548 
01549         /* Increase the buffer to prepare for a possible push.  */
01550         yy_size_t grow_size = 8 /* arbitrary grow size */;
```

```
01551
01552         num_to_alloc = (yy_buffer_stack_max) + grow_size;
01553         (yy_buffer_stack) = (struct yy_buffer_state**)yyrealloc
01554                                 ((yy_buffer_stack),
01555                                 num_to_alloc * sizeof(struct yy_buffer_state*)
01556                                 );
01557         if ( !  (yy_buffer_stack) )
01558             YY_FATAL_ERROR( "out of dynamic memory in yyensure_buffer_stack()" );
01559
01560         /* zero only the new slots.*/
01561         memset((yy_buffer_stack) + (yy_buffer_stack_max), 0, grow_size * sizeof(struct
      yy_buffer_state*));
01562         (yy_buffer_stack_max) = num_to_alloc;
01563     }
01564 }
01565
01566 /* Setup the input buffer state to scan directly from a user-specified character buffer.
01567  * @param base the character buffer
01568  * @param size the size in bytes of the character buffer
01569  *
01570  * @return the newly allocated buffer state object.
01571  */
01572 YY_BUFFER_STATE yy_scan_buffer  (char * base, yy_size_t  size )
01573 {
01574     YY_BUFFER_STATE b;
01575
01576     if ( size < 2 ||
01577          base[size-2] != YY_END_OF_BUFFER_CHAR ||
01578          base[size-1] != YY_END_OF_BUFFER_CHAR )
01579         /* They forgot to leave room for the EOB's.  */
01580         return NULL;
01581
01582     b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
01583     if ( !  b )
01584         YY_FATAL_ERROR( "out of dynamic memory in yy_scan_buffer()" );
01585
01586     b->yy_buf_size = (int) (size - 2);  /* "- 2" to take care of EOB's */
01587     b->yy_buf_pos = b->yy_ch_buf = base;
01588     b->yy_is_our_buffer = 0;
01589     b->yy_input_file = NULL;
01590     b->yy_n_chars = b->yy_buf_size;
01591     b->yy_is_interactive = 0;
01592     b->yy_at_bol = 1;
01593     b->yy_fill_buffer = 0;
01594     b->yy_buffer_status = YY_BUFFER_NEW;
01595
01596     yy_switch_to_buffer( b  );
01597
01598     return b;
01599 }
01600
01601 /* Setup the input buffer state to scan a string.  The next call to yylex() will
01602  * scan from a @e copy of @a str.
01603  * @param yystr a NUL-terminated string to scan
01604  *
01605  * @return the newly allocated buffer state object.
01606  * @note If you want to scan bytes that may contain NUL values, then use
01607  *       yy_scan_bytes() instead.
01608  */
01609 YY_BUFFER_STATE yy_scan_string (const char * yystr )
01610 {
01611
01612     return yy_scan_bytes( yystr, (int) strlen(yystr) );
01613 }
01614
01615 /* Setup the input buffer state to scan the given bytes.  The next call to yylex() will
01616  * scan from a @e copy of @a bytes.
01617  * @param yybytes the byte buffer to scan
01618  * @param _yybytes_len the number of bytes in the buffer pointed to by @a bytes.
01619  *
01620  * @return the newly allocated buffer state object.
01621  */
01622 YY_BUFFER_STATE yy_scan_bytes  (const char * yybytes, int  _yybytes_len )
01623 {
01624     YY_BUFFER_STATE b;
01625     char *buf;
01626     yy_size_t n;
01627     int i;
01628
01629     /* Get memory for full buffer, including space for trailing EOB's.  */
01630     n = (yy_size_t) (_yybytes_len + 2);
01631     buf = (char *) yyalloc( n  );
01632     if ( !  buf )
01633         YY_FATAL_ERROR( "out of dynamic memory in yy_scan_bytes()" );
01634
01635     for ( i = 0; i < _yybytes_len; ++i )
01636         buf[i] = yybytes[i];
```

```
01637
01638     buf[_yybytes_len] = buf[_yybytes_len+1] = YY_END_OF_BUFFER_CHAR;
01639
01640     b = yy_scan_buffer( buf, n );
01641     if ( !  b )
01642         YY_FATAL_ERROR( "bad buffer in yy_scan_bytes()" );
01643
01644     /* It's okay to grow etc.  this buffer, and we should throw it
01645 * away when we're done.
01646 */
01647     b->yy_is_our_buffer = 1;
01648
01649     return b;
01650 }
01651
01652 #ifndef YY_EXIT_FAILURE
01653 #define YY_EXIT_FAILURE 2
01654 #endif
01655
01656 static void yynoreturn yy_fatal_error (const char* msg )
01657 {
01658             fprintf( stderr, "%s\n", msg );
01659     exit( YY_EXIT_FAILURE );
01660 }
01661
01662 /* Redefine yyless() so it works in section 3 code.  */
01663
01664 #undef yyless
01665 #define yyless(n) \
01666 do \
01667 { \
01668 /* Undo effects of setting up yytext.  */\
01669 int yyless_macro_arg = (n); \
01670 YY_LESS_LINENO(yyless_macro_arg);\
01671 yytext[yyleng] = (yy_hold_char); \
01672 (yy_c_buf_p) = yytext + yyless_macro_arg; \
01673 (yy_hold_char) = *(yy_c_buf_p); \
01674 *(yy_c_buf_p) = '\0'; \
01675 yyleng = yyless_macro_arg; \
01676 } \
01677 while ( 0 )
01678
01679 /* Accessor  methods (get/set functions) to struct members.  */
01680
01681 /* Get the current line number.
01682 *
01683 */
01684 int yyget_lineno  (void)
01685 {
01686
01687     return yylineno;
01688 }
01689
01690 /* Get the input stream.
01691 *
01692 */
01693 FILE *yyget_in  (void)
01694 {
01695         return yyin;
01696 }
01697
01698 /* Get the output stream.
01699 *
01700 */
01701 FILE *yyget_out  (void)
01702 {
01703         return yyout;
01704 }
01705
01706 /* Get the length of the current token.
01707 *
01708 */
01709 int yyget_leng  (void)
01710 {
01711         return yyleng;
01712 }
01713
01714 /* Get the current token.
01715 *
01716 */
01717
01718 char *yyget_text  (void)
01719 {
01720         return yytext;
01721 }
01722
01723 /* Set the current line number.
```

```
01724 * @param _line_number line number
01725 *
01726 */
01727 void yyset_lineno (int  _line_number )
01728 {
01729
01730     yylineno = _line_number;
01731 }
01732
01733 /* Set the input stream.  This does not discard the current
01734 * input buffer.
01735 * @param _in_str A readable stream.
01736 *
01737 * @see yy_switch_to_buffer
01738 */
01739 void yyset_in (FILE *  _in_str )
01740 {
01741         yyin = _in_str ;
01742 }
01743
01744 void yyset_out (FILE *  _out_str )
01745 {
01746         yyout = _out_str ;
01747 }
01748
01749 int yyget_debug  (void)
01750 {
01751         return yy_flex_debug;
01752 }
01753
01754 void yyset_debug (int  _bdebug )
01755 {
01756         yy_flex_debug = _bdebug ;
01757 }
01758
01759 static int yy_init_globals (void)
01760 {
01761         /* Initialization is the same as for the non-reentrant scanner.
01762 * This function is called from yylex_destroy(), so don't allocate here.
01763 */
01764
01765     (yy_buffer_stack) = NULL;
01766     (yy_buffer_stack_top) = 0;
01767     (yy_buffer_stack_max) = 0;
01768     (yy_c_buf_p) = NULL;
01769     (yy_init) = 0;
01770     (yy_start) = 0;
01771
01772 /* Defined in main.c */
01773 #ifdef YY_STDINIT
01774     yyin = stdin;
01775     yyout = stdout;
01776 #else
01777     yyin = NULL;
01778     yyout = NULL;
01779 #endif
01780
01781     /* For future reference:  Set errno on error, since we are called by
01782 * yylex_init()
01783 */
01784     return 0;
01785 }
01786
01787 /* yylex_destroy is for both reentrant and non-reentrant scanners.  */
01788 int yylex_destroy  (void)
01789 {
01790
01791     /* Pop the buffer stack, destroying each element.  */
01792     while(YY_CURRENT_BUFFER){
01793         yy_delete_buffer( YY_CURRENT_BUFFER  );
01794         YY_CURRENT_BUFFER_LVALUE = NULL;
01795         yypop_buffer_state();
01796     }
01797
01798     /* Destroy the stack itself.  */
01799     yyfree((yy_buffer_stack) );
01800     (yy_buffer_stack) = NULL;
01801
01802     /* Reset the globals.  This is important in a non-reentrant scanner so the next time
01803 * yylex() is called, initialization will occur.  */
01804     yy_init_globals( );
01805
01806     return 0;
01807 }
01808
01809 /*
01810 * Internal utility routines.
```

```
01811 */
01812
01813 #ifndef yytext_ptr
01814 static void yy_flex_strncpy (char* s1, const char * s2, int n )
01815 {
01816
01817     int i;
01818     for ( i = 0; i < n; ++i )
01819         s1[i] = s2[i];
01820 }
01821 #endif
01822
01823 #ifdef YY_NEED_STRLEN
01824 static int yy_flex_strlen (const char * s )
01825 {
01826     int n;
01827     for ( n = 0; s[n]; ++n )
01828         ;
01829
01830     return n;
01831 }
01832 #endif
01833
01834 void *yyalloc (yy_size_t  size )
01835 {
01836             return malloc(size);
01837 }
01838
01839 void *yyrealloc  (void * ptr, yy_size_t  size )
01840 {
01841
01842     /* The cast to (char *) in the following accommodates both
01843 * implementations that use char* generic pointers, and those
01844 * that use void* generic pointers.    It works with the latter
01845 * because both ANSI C and C++ allow castless assignment from
01846 * any pointer type to void*, and deal with argument conversions
01847 * as though doing an assignment.
01848 */
01849     return realloc(ptr, size);
01850 }
01851
01852 void yyfree (void * ptr )
01853 {
01854             free( (char *) ptr );   /* see yyrealloc() for (char *) cast */
01855 }
01856
01857 #define YYTABLES_NAME "yytables"
01858
01859 #line 52 "abc2.l"
01860
01861
01862 void (*jj_junk)(int,char *) = yyunput; /* avoid gcc warnings */
01863 int (*jj2_junk)(void) = input;
01864
01865 SHARED int deleteCStyleComments(char *filePath)
01866 {
01867     errno = 0;
01868     yyin=fopen(filePath,"r");
01869
01870     getFileName(filePath, sName);
01871     getFileExtension(filePath, sExtension);
01872     getFilePath(filePath, sPath);
01873     char *filToR = strcat(sPath, strcat(strcat(sName,"_"),sExtension));
01874     yyout=fopen(filToR,"w");
01875     yylex();
01876     fclose(yyin);
01877     fclose(yyout);
01878     if (remove(filePath) != 0)
01879     {
01880         fprintf(stderr, "%s\n", strerror(errno));
01881         return -2;
01882     }
01883     if (rename(filToR,filePath) != 0)
01884     {
01885         fprintf(stderr, "%s\n", strerror(errno));
01886         return -1;
01887     }
01888     return 0;
01889 }
```

## 8.17 docs/src_documented/notByMe/lex_yy.h File Reference

This header contains a function written by flex to delete C-style comments in a file.

## Macros

- #define SHARED

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

## Functions

- SHARED int deleteCStyleComments (char ∗filePath)

### 8.17.1 Detailed Description

This header contains a function written by flex to delete C-style comments in a file.

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file lex_yy.h.

### 8.17.2 Macro Definition Documentation

#### 8.17.2.1 SHARED

```
#define SHARED
```

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

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)
#   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 at line 73 of file lex_yy.h.

### 8.17.3 Function Documentation

#### 8.17.3.1 deleteCStyleComments()

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

Definition at line 1865 of file lex_yy.c.

## 8.18 lex_yy.h

Go to the documentation of this file.

```
00001
00033 #ifndef LEX_YY_H
00034 #define LEX_YY_H
00035
00059 # if defined(_WIN32)
00060 /***************** "-D STATIC" ******************/
00061 #   if defined(STATIC)
00062 #     define SHARED
00063 /**************** "-D BUILD_DLL" ****************/
00064 #   else
00065 #     if defined(BUILD_DLL)
00066 #       define SHARED __declspec(dllexport)
00067 #     else
00068 #       define SHARED __declspec(dllimport)
00069 #     endif
00070 #   endif
00071 /****************** DEFAULT ******************/
00072 # else
00073 #   define SHARED
00074 # endif
00075
00076
00077
00078 SHARED int deleteCStyleComments(char *filePath);
00079
00080
00081 #endif
```

## 8.19 docs/src_documented/notByMe/noComments.l File Reference

### Functions

- int **fileno** (FILE *)
- int yylex (void)
- int deleteCStyleComments (char *filePath)

### Variables

- void(* jj_junk )(int, char *) = yyunput
- int(* jj2_junk )(void) = input

### 8.19.1 Detailed Description

**Author**

Axel PASCON (a.k.a. brvtalcake)

**Date**

2022

Definition in file noComments.l.

### 8.19.2 Function Documentation

#### 8.19.2.1 deleteCStyleComments()

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

Definition at line 76 of file noComments.l.

#### 8.19.2.2 yylex()

```
int yylex (
            void  )
```

Definition at line 36 of file noComments.l.

### 8.19.3 Variable Documentation

#### 8.19.3.1 jj2_junk

```
int(* jj2_junk) (void) (
            void  ) = input
```

Definition at line 74 of file noComments.l.

#### 8.19.3.2 jj_junk

```
void(* jj_junk) (int, char *) (
             int ,
             char *  ) = yyunput
```

Definition at line 73 of file noComments.l.

## 8.20 noComments.l

Go to the documentation of this file.

```
00001
00007
00008
00015
00016
00019
00020 %option noyywrap
00021
00022 %{
00023 #include <string.h>
00024 #include <errno.h>
00025 #include "../fileMan.h"
00026 #include "lex_yy.h"
00027 /* remove C comments */
00028
00029 /* replace each comment with a single blank */
00030
00031 extern int fileno(FILE *);      /* avoid gcc warning */
00032 %}
00033
00034 %x String Char Comment CPPComment
00035
00036 %%
00037
00038 \"      /* "String" */ ECHO; BEGIN( String);
00039
00040 \'      /* 'Char' */ ECHO; BEGIN( Char);
00041
00042 "/*"    /* comment */ BEGIN( Comment);
00043
00044 "//"    /* C++ comment */ BEGIN( CPPComment);
00045
00046 .|\n    /* everything else */ ECHO;
00047
00048 <String>{
00049   \\.                   /* escape sequence */ ECHO;
00050   [^"]          /* not "quote" */ ECHO;
00051   \"            /* end of "String" */ ECHO; BEGIN( INITIAL);
00052 }
00053
00054 <Char>{
00055   \\.                   /* escape sequence */ ECHO;
00056   [^']          /* not 'quote' */ ECHO;
00057   \'            /* end of 'Char' */ ECHO; BEGIN( INITIAL);
00058 }
00059
00060 <Comment>{
00061   [^*]*         /* not a '*' */
00062   "*"+[^/]      /* '*'s not followed by '/' */
00063   "*"+"/"       /* end of Comment */ putchar( ' '); BEGIN(INITIAL);
00064 }
00065
00066 <CPPComment>{
00067   .*            /* to end of line */
00068   \n            ECHO; BEGIN( INITIAL);
00069 }
00070
00071 %%
00072
00073 void (*jj_junk)(int,char *) = yyunput; /* avoid gcc warnings */
00074 int (*jj2_junk)(void) = input;
00075
00076 int deleteCStyleComments(char *filePath)
00077 {
00078     errno = 0;
00079   yyin=fopen(filePath,"r");
```

```
00080
00081     getFileName(filePath, sName);
00082     getFileExtension(filePath, sExtension);
00083     getFilePath(filePath, sPath);
00084     char *filToR = strcat(sPath, strcat(strcat(sName,"_"),sExtension));
00085   yyout=fopen(filToR,"w");
00086   yylex();
00087     fclose(yyin);
00088     fclose(yyout);
00089     if (remove(filePath) != 0)
00090     {
00091         fprintf(stderr, "%s\n", strerror(errno));
00092         return -2;
00093     }
00094     if (rename(filToR,filePath) != 0)
00095     {
00096         fprintf(stderr, "%s\n", strerror(errno));
00097         return -1;
00098     }
00099     return 0;
00100
```

# Index