

# SmartFolder

Maxime Lovino

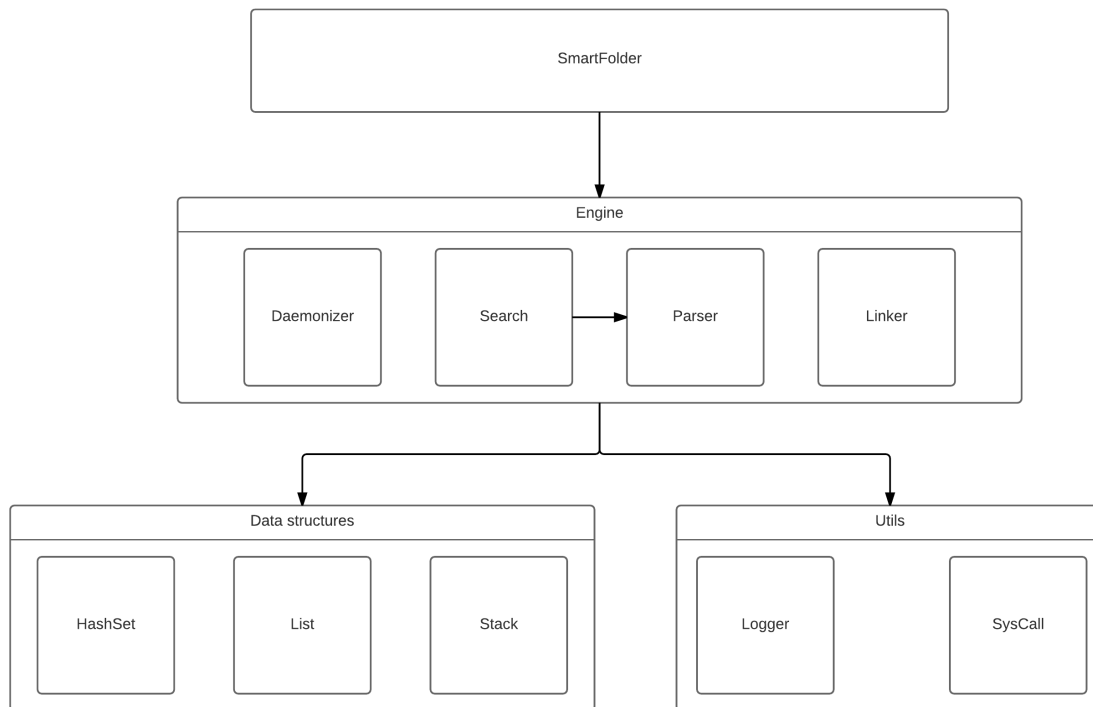
Thomas Ibanez

22 décembre 2016

# 1 Introduction

Hello world

## 2 Architecture



### 2.1 Data Structures

#### 2.1.1 List

Nous avons une structure de liste qui sert à stocker la liste des résultats d'une recherche (liste de fichiers). Nous définissons des fonctions permettant d'agréer plusieurs listes au travers d'une union, intersection, etc... correspondant aux opérations booléennes classiques.

```
#ifndef _LIST_H_
#define _LIST_H_

typedef struct listElement_st {
    char* data;
    struct listElement_st* next;
} ListElement;

typedef struct list_st {
    ListElement* head;
    int size;
} List;

List* initList();
void insert(List* l, char* element);
void removeIndex(List* l, int idx);
void removeObject(List* l, char* element);
int searchInList(List* l, char* element);
char* get(List* l, int idx);
List* listUnion(List* l1, List* l2);
```

```
List* listIntersect(List* l1, List* l2);
List* listXOR(List* l1, List* l2);
List* listComplement(List* l1, List* l2);
void deleteList(List* l);

#endif /* end of include guard: _LIST_H_ */
```

### 2.1.2 HashSet

Nous avons une structure de table de hachage permettant de stocker les matches actuels présent dans le dossier, cela nous permet lors du prochain run de la recherche de tester de façon rapide (recherche en  $O(1)$  en général et maximum en  $\Theta(n)$ ).

```
#ifndef _HASH_SET_
#define _HASH_SET_
#include "Logger.h"

typedef struct HashSet {
    char** table;
    int size;
    int filled;
} HashSet;

void initSet(HashSet* table, int size);
void expand(HashSet** table);
void put(HashSet** table, char* filePath);
void removeFromSet(HashSet* table, char* filePath);
int contains(HashSet* table, char* filePath);
int searchInSet(HashSet* table, char* filePath);
int hash(char* text);
void deleteSet(HashSet** table);
void dumpSet(HashSet* table);

#endif /* end of include guard: _HASH_SET_ */
```

### 2.1.3 Stack

Nous utilisons une structure de Pile pour stocker les résultats d'une recherche particulière (on stocke un pointeur de la liste des résultats) pour ensuite les rassembler lorsque nous trouvons un opérateur booléen. (principe de l'évaluation d'une expression polonaise inverse)

```
#ifndef _STACK_H_
#define _STACK_H_

typedef struct stackElement_st {
    struct stackElement_st* next;
    void* value;
} stackElement;

typedef struct stack_st {
    stackElement* top;
    int size;
} Stack;

Stack* initStack();
void push(Stack* s, void* element);
void pop(Stack* s);
```

```
int isEmpty(Stack* s);
void deleteStack(Stack* s);

#endif /* end of include guard: _STACK_H_ */
```

## 2.2 Utils

### 2.2.1 Logger

Nous avons réalisé un Logger pour pouvoir afficher sur la console des messages de notre programme, pour ceci nous définissons 3 niveaux de severité pour un message :

- 0 : INFO, un message d'information, par exemple un dump de hashtable ou des informations sur les fichiers trouvés
- 1 : WARNING, un message de warning, par exemple une erreur dans notre programme durant son execution mais qui n'empêche pas le programme de continuer
- 2 : FATAL, un message d'erreur, lorsqu'une erreur se produit qui fait crasher le programme entièrement

Dans le fichier `Logger.h` on peut changer la valeur du `#define LOG_LEVEL X` par le niveau minimal que l'on veut. C'est-à-dire que si nous définissons 1 ici, le logger affichera uniquement les messages de niveau 1 et 2.

Dans notre code nous pouvons utiliser la fonction `void logMessage(int level, const char* format, ...)`; pour logger un message en spécifiant son niveau et en utilisant une syntaxe similaire à `printf(...)` ensuite.

```
#ifndef _LOGGER_H_
#define _LOGGER_H_
#ifndef LOG_LEVEL
#define LOG_LEVEL 0
#endif

#include <stdio.h>
#include <stdarg.h>

void logMessage(int level, const char* format, ...);

#endif /* end of include guard: _LOGGER_H_ */
```

### 2.2.2 SysCall

Nous avons un fichier regroupant des fonctions wrappers pour les appels systèmes que nous utilisons dans les différentes parties de notre programme. De ce fait, si nous voulons porter notre programme sur un autre système, nous avons juste à modifier le contenu de ces fonctions pour adapter ces appels systèmes.

## 2.3 Engine

### 2.3.1 Search

### 2.3.2 Parser

### 2.3.3 Linker

### 2.3.4 Daemonizer

## 2.4 SmartFolder

# 3 Utilisation du programme

## 3.1 Syntaxe de recherche