I'M IN LOVE WITH THE XML

RAPPORT DE PROJET

Audrey DI VITO | Noé LARRIEU-LACOSTE | Isaac OUSLIMANE

Langage C Avancé

26 novembre 2020

TABLE DES MATIERES

I. INTRODUCTION

1. RAPPEL DU SUJET

Pour recontextualiser, le lancement du projet a eu lieu à l'issue de la semaine de C, le vendredi 16 octobre. Pour ce projet, nous devons réaliser une application en C permettant de valider des fichiers XML à l’aide d’une DTD.

Nous devons construire un ensemble de fonction permettant dans un premier temps de tester si les éléments d’un fichier xml sont conformes à la DTD, ainsi que de vérifier si le fichier XML est correct.

Nous devons ensuite prendre en compte la validation des attributs afin qu’ils soient conformes à la DTD.

Il faut également vérifier si le fichier XML est correct, sans limite de profondeur.

Enfin, il faut réaliser un éditeur graphique de DTD afin de valider directement un fichier XML depuis une interface développé avec **GTK**. Cette interface devra également être capable de proposer des suggestions lors de l’écriture du fichier XML.

2. ETUDIANTS AYANT PARTICIPES

Audrey DI VITO

Je m’appelle Audrey DI VITO et je viens d’un BTS SNIR. J’ai déjà fait du C en première année de BTS mais je n’avais jamais fait du C jusqu’à ce niveau. La piscine de C s’est donc avérée assez compliquée concernant certains exercices. Pour le projet, c’était similaire. Déjà à l’annonce du projet, cela me semblait impossible pour moi de le faire jusqu’à l’interface graphique.

Noé LARRIEU-LACOSTE

Je m’appelle Noé LARRIEU-LACOSTE, actuellement en 3ème année à l’ESGI, j’ai également effectué mes deux premières années dans cette école. Cela m’a permis de me familiariser avec le C à travers plusieurs projet, nécessitant parfois une interface graphique. La piscine s’est donc mieux déroulée car je connaissais déjà certaines subtilités du langage.

Le projet demandé ne me parait impossible mais va nécessiter un investissement je pense supérieur aux 60h estimés dans le sujet. Je ne connais pas très bien le XML et pas du tout la DTD donc cela va demander un apprentissage et de la réflexion sur comment appréhender le sujet. Enfin, la dernière partie concernant l’autocomplétions me laisse pour le moment perplexe.

Isaac OUSLIMANE

Je m'appelle Isaac, j'ai 21 ans, j'ai eu un bac scientifique et un BTS Systèmes numériques option informatique. C'est ma première année à l'ESGI. J'appréhendais la semaine de C et ce projet, car je n'ai jamais eu l'occasion de travailler le C durant mon cursus scolaire, mais j'ai vu ça comme une opportunité afin de développer mes compétences en C ainsi que mon travail d'équipe.

Concernant le projet en lui-même il apparaît comme une suite logique a la semaine de C. Il nous permet de remettre en pratique tout ce que l'on a pu étudier durant cette semaine et développer notre autonomie.

II. FOCUS SUR L’APPLICATION

1. FONCTIONNALITES

Parser[1] un fichier XML

Avant d’être une application permettant de valider un fichier XML, elle doit le comprendre, l’analyser. C’est pour cela qu’une des principales fonctionnalités de l’application consiste à parser un fichier XML et le stocker à l’aide de structures de données.

Cela nous apporte plusieurs avantages :

Avoir un document structuré dans notre code afin de mieux naviguer dedans

La limite de profondeur n’est plus un problème

Nous pouvons au moment où nous parsons un document détecter les erreurs de syntaxes

Parser un fichier DTD

De la même manière que nous parsons un fichier XML, nous parsons un fichier DTD. En effet le parsing d'un fichier DTD est plus ou moins semblable au parsing d'un fichier XML, le principe reste le même.

Au moyen de structure de données conçues spécialement pour un fichier DTD dont nous détaillerons l'utilisation plus tard dans le document, nous stockons toutes les données issues du fichier.

C'est pourquoi certains avantages constatés pour le parsing d'un fichier XML sont également applicables dans le cas d'un fichier DTD.

[1] **Parser \paʁ.se** v. , Parcourir le contenu d'un texte ou d'un fichier en l'analysant pour vérifier sa syntaxe ou en extraire des éléments.

Vérification syntaxique du fichier XML

Lorsque nous parsons notre fichier XML pour le stocker, il faut que le fichier XML soit correct pour ne pas créer d’erreur.

Nous avons donc dû implémenter une série de règles afin de vérifier que notre document est correct.

Voici une liste non exhaustive de certaines de ces règles :

- Une seule balise à la racine du document
- La balise **<?xml**, si elle existe, doit être au tout début du document
- Un texte doit forcément être à l'intérieur d'une balise et non à la racine du document
- Toute balise ouverte doit être fermée
- Les balises fermante doivent porter le même nom que les balises ouvrantes du même niveau
- Les commentaires doivent bien respecter la syntaxe, à savoir **< !--** *Commentaire* **-->**
- Tout attribut doit posséder une clé et une valeur
- La valeur d'un attribut peut être en simple quote ou double quote
- Le premier caractère d'une balise doit forcément être un caractère virgule et les suivants doivent être soit des caractères , soit des chiffres, soit certains caractères spéciaux très spécifiques (-, _)

Si une seule de ces règles n’est pas respecté, cela coupe le processus de parsing, indique dans l’application ainsi que dans un fichier log.txt la ligne, la colonne ainsi que la nature de l’erreur.

Un code erreur est alors renvoyé.

Vérification syntaxique du fichier DTD

Bien que ce ne soit pas indiqué explicitement dans le sujet que nous devons vérifier la syntaxe du fichier DTD, nous avons choisi de le faire.

Ainsi notre application avant de vérifier si le fichier XML est correct comme demandé dans l'énoncé du projet, vérifie d'abord la conformité du fichier DTD.

Une analyse syntaxique est réalisée afin de desceller de potentielles erreurs.

Par conséquent notre application gère à la fois les erreurs au niveau du fichier XML, mais également les potentielles erreurs du fichier DTD.

Voici une liste non exhaustive de certaines de ces règles :

- La règle DOCTYPE ne doit contenir le nom que d'une seule balise
- Une règle concernant un élément doit avoir sa règle entre parenthèses
- Les éléments présents dans la parenthèse d’une règle ELEMENT doivent être séparées par des caractères spécifique : **|** ou **,**
- Il doit forcément il y avoir un élément après | ou ,
- Chaque balise ouverte doit être fermée par >
- Les parenthèses ne peuvent pas être vides
- Les éléments contenus dans la parenthèse et séparés ne peuvent pas être composés de plusieurs mots
- Sur une règle concernant un attribut (ATTLIST) il faut forcément qu'il y ait une option commençant par # tel que #REQUIRED ou #IMPLIED

Validation de la conformité d’un fichier XML depuis une DTD externe

C'est donc à partir d'une DTD externe que nous vérifions si le fichier XML est correct ou non.

Nous allons donc charger dans des structures de données notre document XML et DTD.

A partir de cela, nous allons itérer sur les règles DTD pour valider notre document XML, à savoir :

Vérifier la syntaxe du fichier XML et DTD au moment du parsing

Valider le DOCTYPE (première balise) du document XML en accord avec la règle DTD si celle-ci est présente

Valider les règles DTD concernant les éléments du fichier XML ainsi que leur contenu

Valider les règles DTD concernant les attributs d’un élément XML ainsi que leur contenu

2. CHOIX D’IMPLEMENTATIONS

DTD externe

Pour une meilleure répartition des tâches et afin de prendre une décision dès le début du projet, il a été décidé que notre application serait conçue pour valider un document XML depuis une DTD externe.

Cela signifie que la DTD doit être dans un fichier à part (avec l’extension **dtd**) et non au début du fichier XML.

Parser le fichier XML et DTD

Nous ne savons pas s’il y a d’autres approches possibles pour valider un document XML, mais nous avons fait le choix de parser nos documents afin de pouvoir vérifier leur conformités, mais aussi car cela nous semblait plus simple aussi pour itérer sur chaque règle DTD et naviguer dans le document XML, peu importe la profondeur.

Cross plateforme

Parce qu’un programme qui fonctionne sur un système d’exploitation c’est bien, un programme qui fonctionne sur plusieurs systèmes d’exploitation c’est mieux !

Nous avons souhaité faire en sorte que tout notre programme soit compilable aussi bien sur le système *Windows* que *Linux*.

Cela impliquait d’être vigilant par rapport aux dépendances, qui peuvent varier d’un système à l’autre, surtout GTK qui fait bien transpirer sur Windows…

Projet CMAKE

CMake (**C**ross platform **M**ake) est un outil open source permettant de gérer la compilation d'un projet C/C++ sur différente plateformes.

C’est beaucoup plus simple que de devoir compiler à la main avec GCC, et permet de créer des scripts avancer, qui vont beaucoup aider pour lier des librairies à notre projet et aussi faire une application multi plateforme.

L’outil est bien pris en charge dans l’IDE Clion, éditeur que nous utilisons pour développer notre application.

3. STRUCTURES DE DONNEES

Pour que notre application fonctionne correctement, notamment au niveau du parsing, il a fallu créer différentes structures de données afin d'ordonner notre code.

DTD Parser

```
typedef struct dtd_document_s dtd_document;

struct dtd_document_s {
    char *source;
    char *root_node;
    element_node *first_element_node;
    attribute_node *first_attribute_node;
};
```

Cette structure est notre point d'entrée vers notre document DTD.

Les champs suivants représentent :

Char *source : contient sous forme de chaine de caractère l'intégralité de notre fichier DTD.

Char *root_node : correspond au nom que doit avoir la première balise du document XML (!DOCTYPE). Si ça n'est pas indiqué, la variable vaut **NULL**.

Element_node *first_element_node : premier élément de notre liste chainée contenant les règles sur les éléments

Attribute node *first_attribute_node : premier élément de notre liste chainée contenant les règles sur les attributs

```c
typedef struct element_node_s element_node;

struct element_node_s {
    char *tag_name;
    char *rule_type;
    dtd_rule *rule;
    struct element_node_s *next;
};
```

Cette structure fonctionne comme une liste chainée contenant toute les règles sur les éléments d'un fichier DTD.

Les champs suivants représentent :

Char *tag_name : correspond au nom de la/les balise(s) concernée(s) par la règle

Char *rule_type : type de règle, en l'occurrence « !ELEMENT »

Dtd_rule *rule : premier élément de notre liste chainée contenant le détail des règles pour l'élément concerné

Element_node *next : prochaine règle concernant un élément. Si c'est la dernière règle, ***next** vaudra alors **NULL**

```c
typedef struct dtd_rule_s dtd_rule;

struct dtd_rule_s {
    char *rule_name;
    char rule_spec;
    char rule_sep;
    struct dtd_rule_s *next;
};
```

Dans le cas d'une règle pour les éléments, il peut y avoir différents éléments dans la parenthèse, séparé par différents caractères.

Les champs suivants représentent :

Char *rule_name : le nom de l'élément enfant, ou bien **#PCDATA**

Char rule_spec : la répétition de l'élément (+, ?,*)

Char rule_sep : la séparation avec la règle suivante (, |)

Dtd_rule *next : prochaine règle. Si c'est la dernière règle, ***next** vaudra alors **NULL**

```c
typedef struct attribute_node_s attribute_node;

struct attribute_node_s {
    char *rule_type;
    char *element_name;
    char *attribute_name;
    char *attribute_type;
    char *attribute_option;
    struct attribute_node_s *next;
};
```

Cette structure fonctionne comme une liste chainée contenant toute les règles sur les attributs d'un fichier DTD.

Les champs suivants représentent :

Char *rule_type : type de règle, en l'occurrence « !ATTLIST»

Char *element_name : nom de la balise concernée dans le document XML

Char *attribute_name : nom de l'attribut concerné.

Char *attribute_type : contient la règle sur le contenu de l'attribut comme (H | F) ou encore **CDATA**

Char *attribute_option : contient l'option de la règle, comme par exemple **#REQUIRED** ou encore **#IMPLIED**

Attribute_node *next : prochaine règle concernant un attribut. Si c'est la dernière règle, ***next** vaudra alors **NULL**

XML Parser

```c
typedef struct xml_document_s xml_document;

struct xml_document_s {
    char *source;
    char *version;
    char *encoding;
    xml_node *root_node;
};
```

Cette structure est notre point d'entrée vers notre document XML.

Les champs suivants représentent :

Char *source : contient sous forme de chaine de caractère l'intégralité de notre fichier XML.

Char *version : contient la version du document si elle a été renseignée, sinon vaut **NULL**

Char *encoding : contient l'encodage du document si ça a été renseignée, sinon vaut **NULL**

Xml_node *root_node : correspond à la première balise de notre document XML.

```c
typedef struct xml_node_s xml_node;

struct xml_node_s {
    char *tag;
    char *inner_text;
    xml_node *parent;
    xml_attribute_list attribute_list;
    xml_node_list children;
};
```

Cette structure représente une balise XML ainsi que son contenu.

Les champs suivants représentent :

Char *tag : le nom de la balise XML

Char *inner_text : le texte à l'intérieur de la balise s'il y en a, sinon vaut **NULL**

Xml_node *parent : la balise parente, si c'est la première balise du document, vaut **NULL**

Xml_attribute_list attribute_list : contient les informations sur les attributs de la balise.

Xml_node_list children : contient les informations sur les balises enfantes de celle-ci.

```c
typedef struct xml_attribute_list_s xml_attribute_list;

struct xml_attribute_list_s {
    int capacity;
    int size;
    xml_attribute *data;
};
```

Contient les informations sur les attributs d’une balise.

Les champs suivants représentent :

Int capacity : capacité de notre tableau d’attributs

Int size : nombre d’attributs

Xml_attribute *data : tableau d’attributs

```c
typedef struct xml_attribute_s xml_attribute;

struct xml_attribute_s {
    char *key;
    char *value;
};
```

Contient les information d’un attribut.

Les champs suivants représentent :

Char *key : la clé de l’attribut

Char *value : le contenu de l’attribut

```c
typedef struct xml_node_list_s xml_node_list;

struct xml_node_list_s {
    int capacity;
    int size;
    xml_node **data;
};
```

Contient les informations sur les enfants d’une balise.

Les champs suivants représentent :

Int capacity : capacité de notre tableau d’enfants

Int size : nombre d’enfants

Xml_node **data : tableau de balises XML

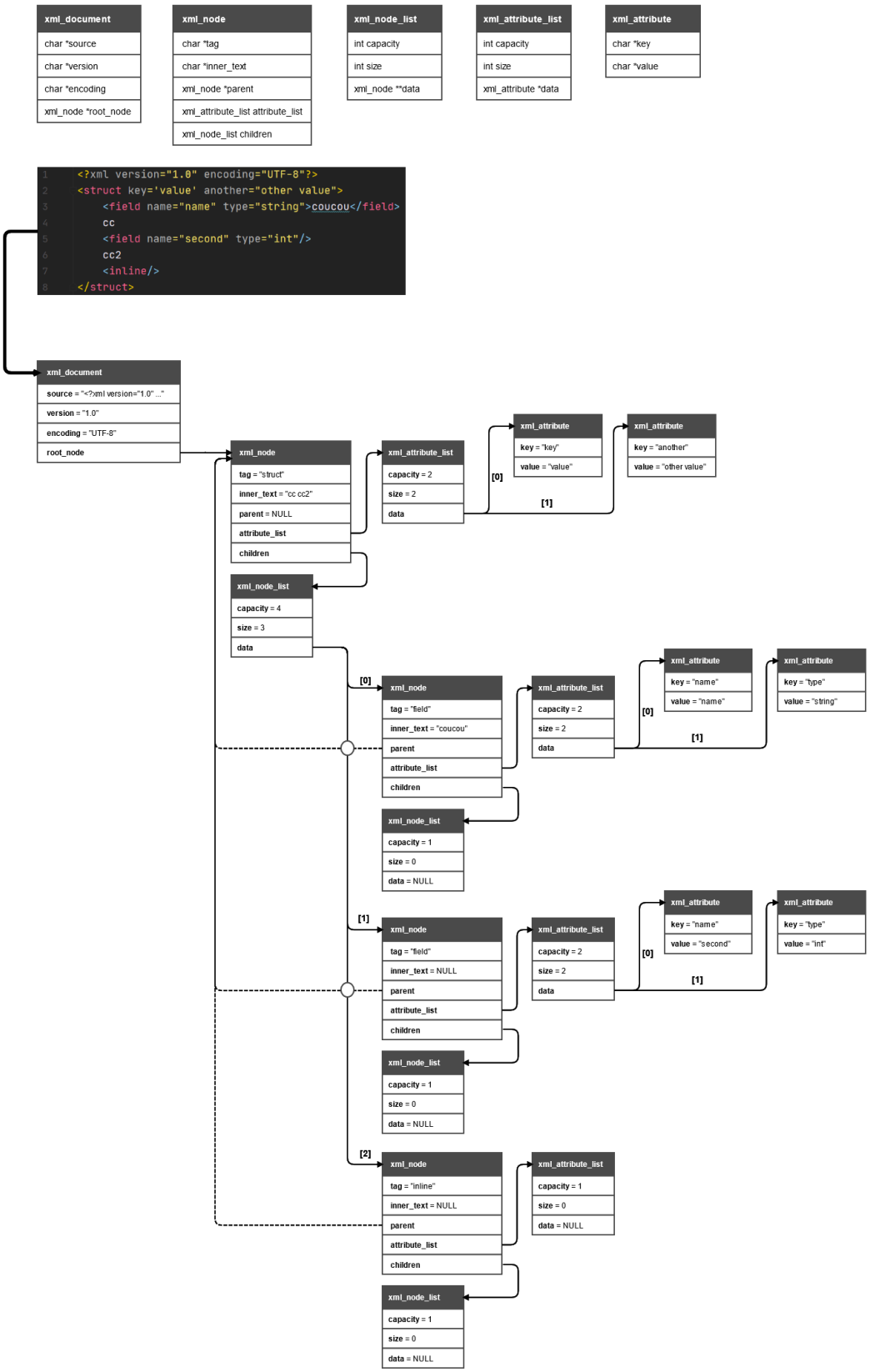
GTK

Pour GTK, il est utile de créer une structure de données regroupant tout les widget de notre interface. De cette manière, à l’aide d’une variable statique que nous initialisons au début du programme pour connecter chacun des widgets au pointeur associé, nous pouvons interagir avec n’importe quel widget depuis n’importe où dans notre code.

```c
typedef struct {
    GtkWidget *window;
    GtkButton *validateButton;
    GtkButton *flushButton;
    GtkFileChooserButton *xmlFileChooserButton;
    GtkFileChooserButton *dtdFileChooserButton;
    GtkLabel *statusLabel;
    GtkTextBuffer *consoleTextBuffer;
    GtkTextView *consoleTextView;
    GtkScrolledWindow *scrollableWindow;
} App_widgets;


static App_widgets *widgets;
```

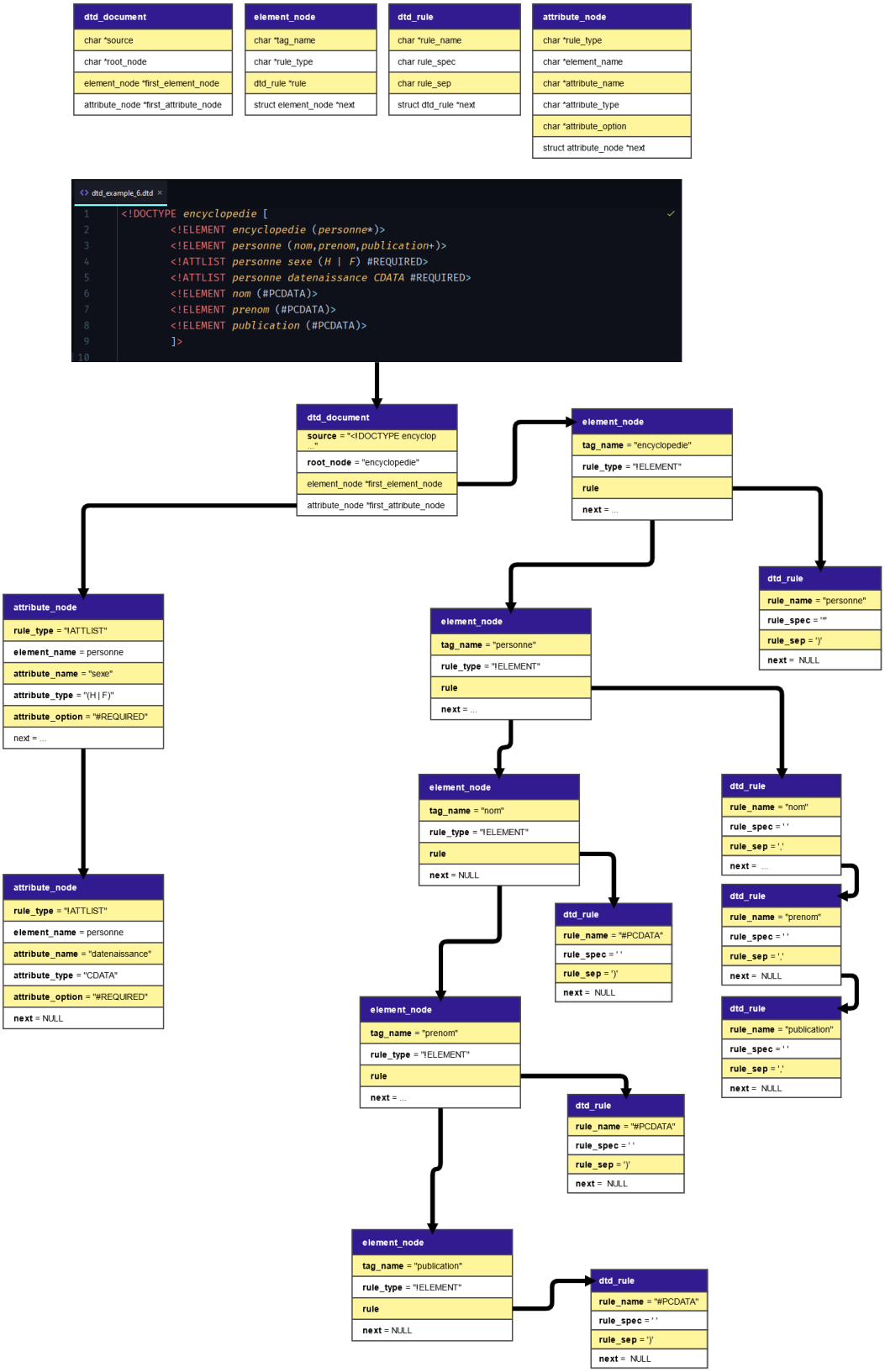
4. SCHEMA XML

Ceci est la représentation graphique de la manière dont la parseur XML traite le document et créé un ensemble de structure de données représentant notre document XML.



5. SCHEMA DTD

Ceci est la représentation graphique de la manière dont la parseur DTD traite le document et créé un ensemble de structure de données représentant notre document DTD.



6. FONCTIONS PRINCIPALES

Menu en ligne de commande

L’application comporte un **menu_cli** qui est simple. Le menu se décompose en 2 parties. L’utilisateur doit choisir d’un côté le fichier XML et de l’autre le fichier DTD.

Ensuite, l’application vérifie si les fichiers existent dans les chemins que l’utilisateur fourni ainsi que l’extension des fichiers.

Si l’un des deux fichiers n’est pas correct, l’application affiche une erreur et demande à saisir de nouveau le chemin d’accès.

Il envoie ensuite les informations à la méthode de validation.

Parse DTD

Comme son nom l'indique la fonction "**parse_dtd**" nous permet de parser le document DTD. C'est une fonction primordiale dans notre application car elle nous permet d'une part de parcourir le fichier DTD mais également de vérifier qu'il ne comporte pas d'erreurs de syntaxe.

Ci-joint la définition de cette méthode :

```
int parse_dtd(dtd_document *document);
```

Comme on peut le voir cette fonction prend en paramètre une variable de type « **dtd_document** » ce qui correspond à notre structure « **dtd_document_s** » et renvoie un entier.

Pour comprendre pourquoi, ci-joint la déclaration de cette méthode :

```
int parse_dtd(dtd_document *document) {

    size_t size = strlen(document->source);

    if (is_doctype(document, size)) {
        return doctype_process(&document, size);
    } else {
        return no_doctype_process(&document, size);
    }

    return 0;
}
```

Étant donné qu'un fichier DTD peut comporter ou non la déclaration "**DOCTYPE!**", la fonction "**parse_dtd**" gèrent les deux possibilités.

C'est la méthode "**is_doctype(document, size)**" qui le détermine en recherchant la présence de la chaine "**DOCTYPE!**" dans le fichier.

Les méthodes « **doctype_process** » et « **no_doctype_process** » sont très semblables, mais quoi qu'il en soit à l'issue du parsing du fichier DTD les méthodes renvoient un entier, 1 s'il n'y a eu aucune erreur durant la compilation et 0 le cas échéant.

Parse xml

Le nom de cette fonction est également explicite.

Tout comme la méthode « **parse_dtd** » cette méthode renvoie l'entier '**1**' si l'exécution n'a rencontré aucun problème, **0** sinon et prend en paramètre une variable de type « **xml_document** » ce qui correspond à notre structure « **xml_document** » et la taille du fichier XML.

Ci-joint la définition de cette méthode :

```
int parse_xml_file(xml_document *document, size_t size);
```

Validate dtd

C’est la fonction « **validate_dtd**» qui détermine si le fichier XML est conforme au fichier DTD, elle prend en paramètres le chemin vers le fichier XML ainsi que le chemin vers le fichier DTD.

Ci-joint la définition de la méthode :

```
int validate_dtd(const char *xml_path, const char *dtd_path);
```

Toute comme les méthodes de parsing, cette méthode renvoie l'entier '**1**' si l'exécution n'a rencontré aucun problème, **0** sinon.

Voilà comment la méthode peut être est appelée :

```
if (validate_dtd("xml_files/xml_example_6.xml",
"dtd_files/dtd_example_6.dtd"))
{
    printf("DTD Test 1 valided\n");
}
```

7. DETAILS TECHNIQUES

Fichier log

Notre application comprend un fichier log qui nous permet d'obtenir et/ou d'afficher différentes informations relatives à l'utilisation de notre application.

Ce fichier s'avère très utile lorsqu'on souhaite comprendre l'origine d'une erreur. Dans notre cas tout cela est géré à l'aide d'un ensemble de méthodes.

Ci-joint les définitions des fonctions du fichier **log.h**

```
int setLogFileName(char *filename);

void logIt(char *message, int error);

void console_writeline(const char *text);
```

Voici un exemple de log, on y retrouve la date et l’heure. Lorsque une erreur est détecté, on remarque que la ligne et la colonne de l’erreur est renseigné.

```
[21-11-2020 14:30:57] File exist !

[21-11-2020 14:31:02] Testing 'D:\Projets\CXML\dtd_files\dtd_example_6.dtd' file extension, should be 'dtd'
[21-11-2020 14:31:02] Extension is valid !

[21-11-2020 14:31:02] Testing if 'D:\Projets\CXML\dtd_files\dtd_example_6.dtd' exists...

[21-11-2020 14:31:02] File exist !

[21-11-2020 14:31:02] ERROR line 22 column 15 - Closing tag don't match with opening tag. Expected :'nom', got :'personne'
[21-11-2020 14:31:24] ERROR line 22 column 15 - Closing tag don't match with opening tag. Expected :'nom', got :'personne'
[21-11-2020 15:05:31] Testing 'D:\Projets\CXML\dtd_files\dtd_example_6.dtd' file extension, should be 'xml'
[21-11-2020 15:05:31] Wrong file extension, expected 'xml' but founded 'dtd' !
```

CMake

CMake (**C**ross platform **M**ake) est un outil open source permettant de gérer la compilation d'un projet C/C++ sur différente plateformes.

C’est beaucoup plus simple que de devoir compiler à la main avec GCC, et permet de créer des scripts avancer, qui vont beaucoup aider pour lier des librairies à notre projet et aussi faire une application multi plateforme.

L’outil est bien pris en charge dans l’IDE Clion, éditeur que nous utilisons pour développer notre application.

III. DOSSIER D'INSTALLATION

1. PARTIE 1 A 3 (MENU LIGNE DE COMMANDES)

Pour installer l'application de la partie 1 à 3, rien de plus simple !

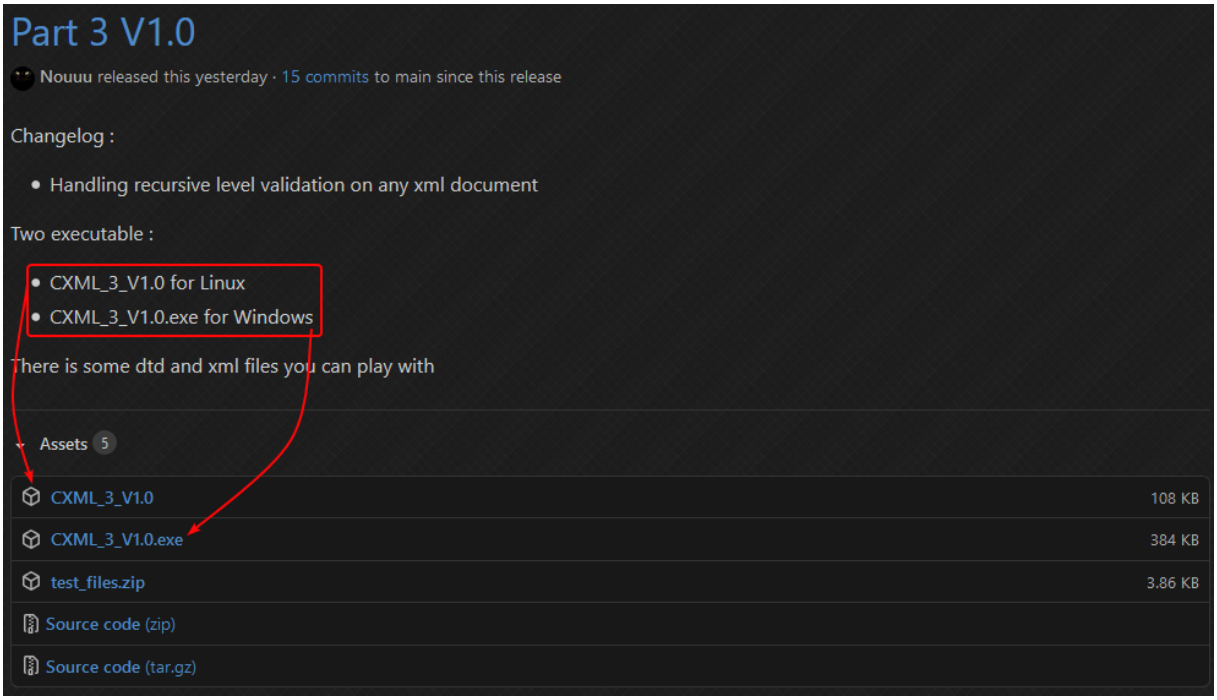
Il suffit simplement de télécharger avec le lien suivant, selon la version que vous souhaitez :

Partie 1 : https://github.com/Nouuu/CXML/releases/tag/Part_1_v2.5

Partie 2 : https://github.com/Nouuu/CXML/releases/tag/Part_2_v1.5

Partie 3 : https://github.com/Nouuu/CXML/releases

Choisir l'exécutable correspondant au système, cela est détaillé à chaque fois :



Il suffira alors de lancer l'exécutable (Cf : Dossier d'utilisation).

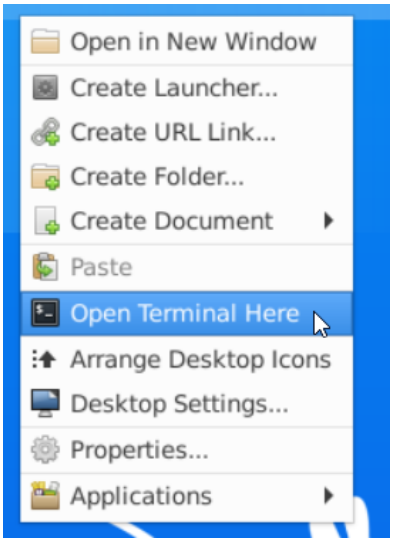
2. PARTIE 4 (APPLICATION INTERFACE GRAPHIQUE GTK)

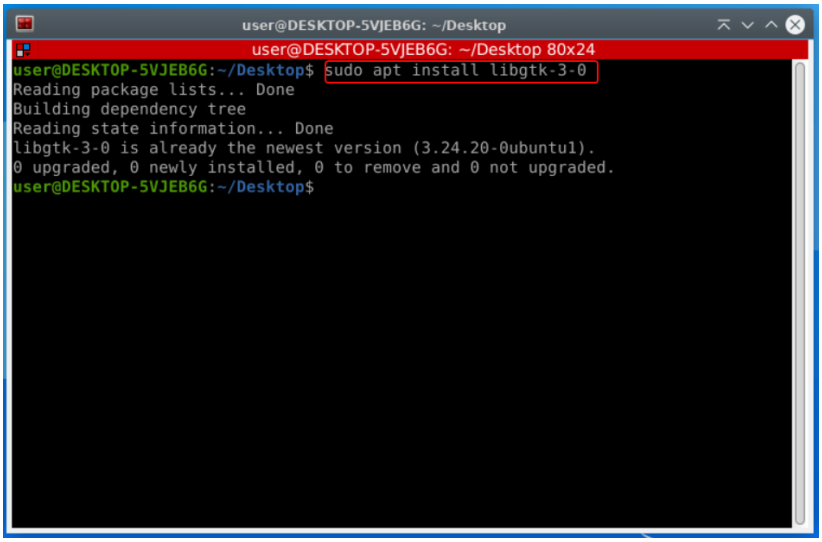
La partie 4 contient des fichiers pour faire fonctionner correctement GTK, et s’installe donc différemment.

Linux

Avant tout chose, vous devez installer sur votre poste la librairie GTK.

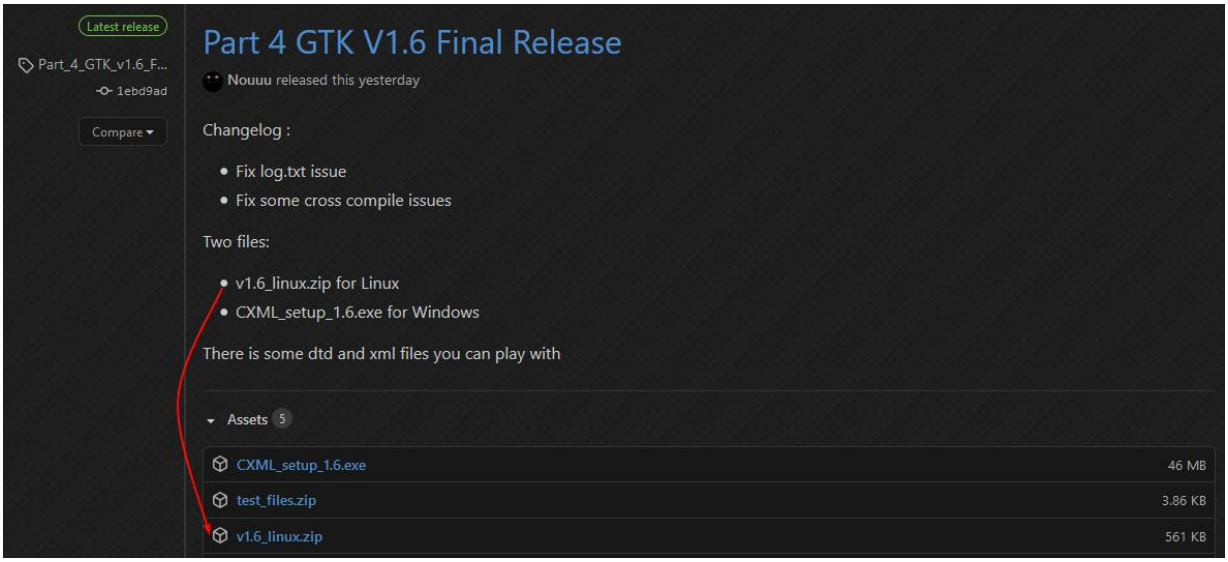
Pour cela, vous devez ouvrir une invite de commande avec les droits administrateurs et installer le paquet suivant :



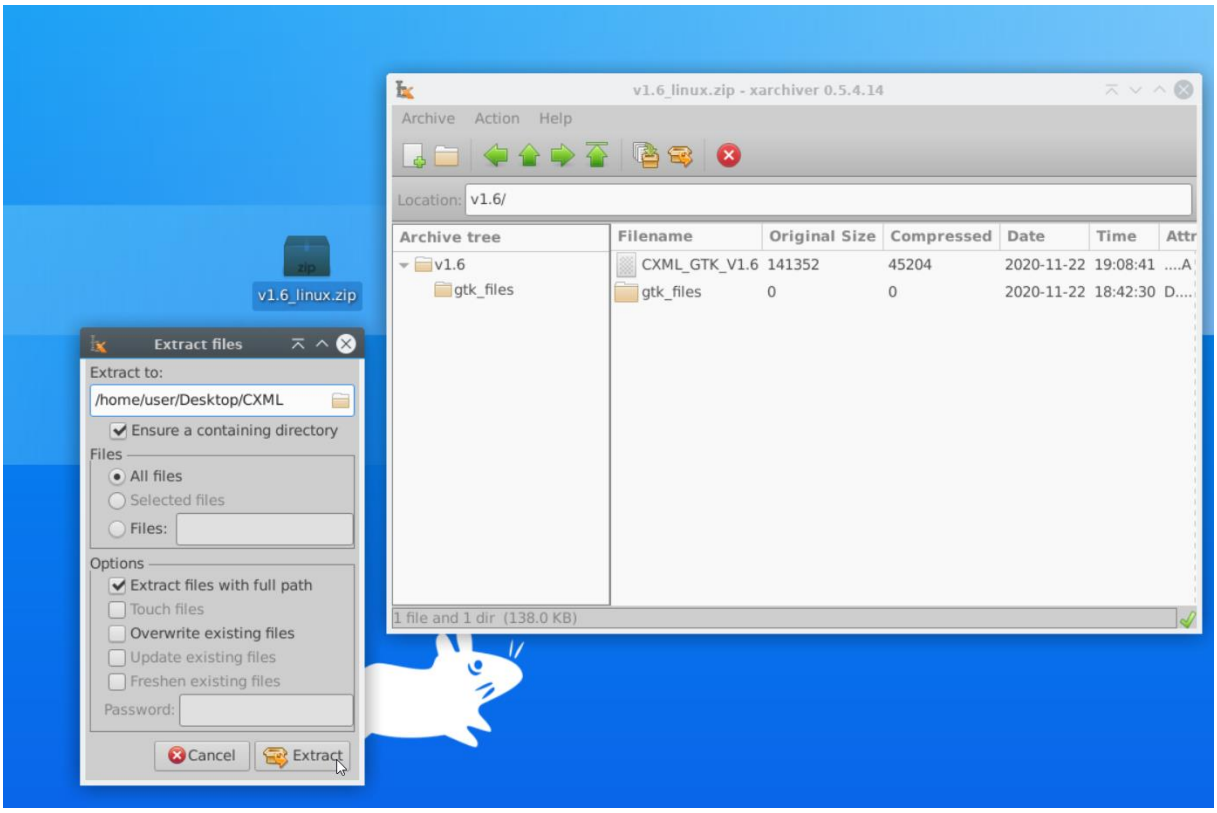


Vous pourrez télécharger ensuite l’archive de l’application depuis ce lien, en faisant attention à bien prendre la version pour Linux.

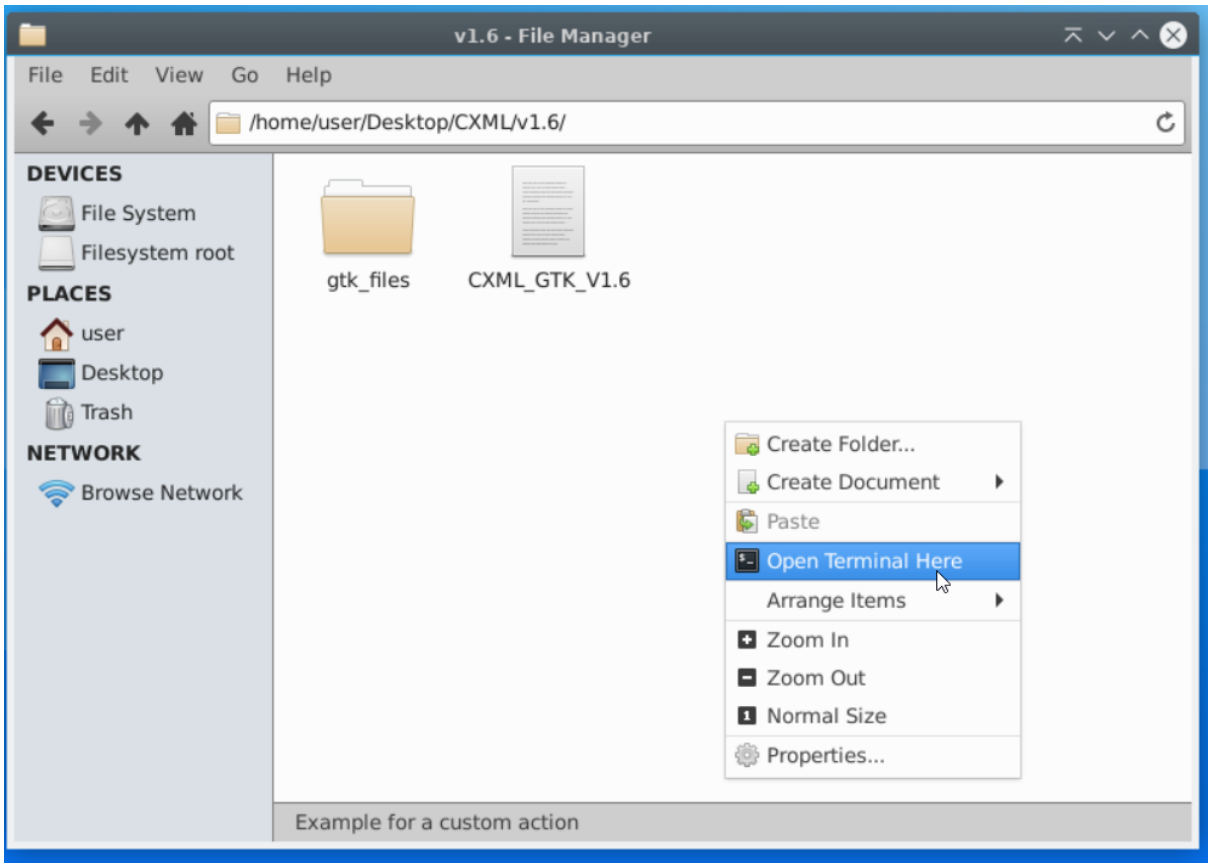
https://github.com/Nouuu/CXML/releases



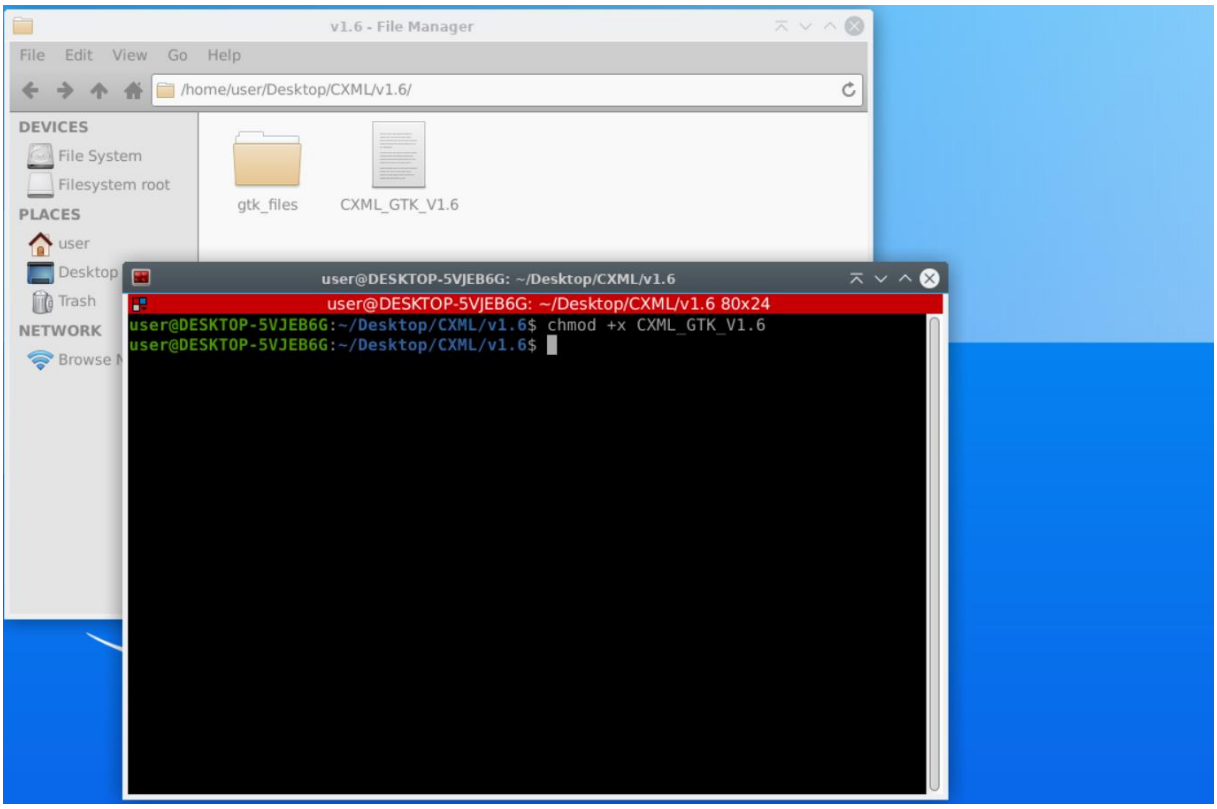
Une fois téléchargé, décompressez l’archive :



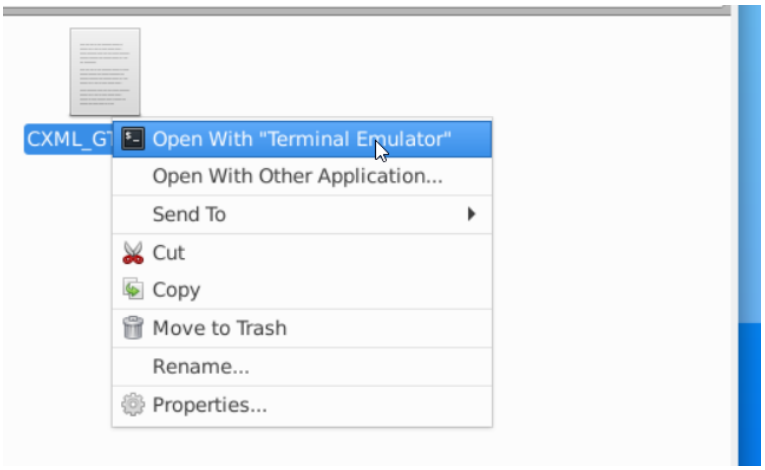
Rendez-vous ensuite à l’intérieur du dossier et ouvrez-y un terminal :



Enfin, exécutez la commande suivante pour rendre le lanceur d’application exécutable :



Vous pourrez alors faire un clic droit dessus et l’ouvrir avec un terminal :



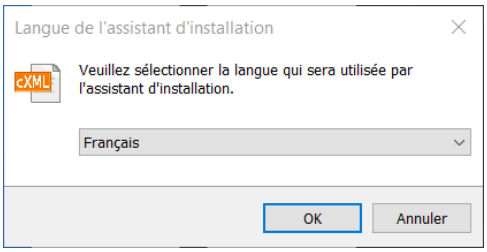
Windows

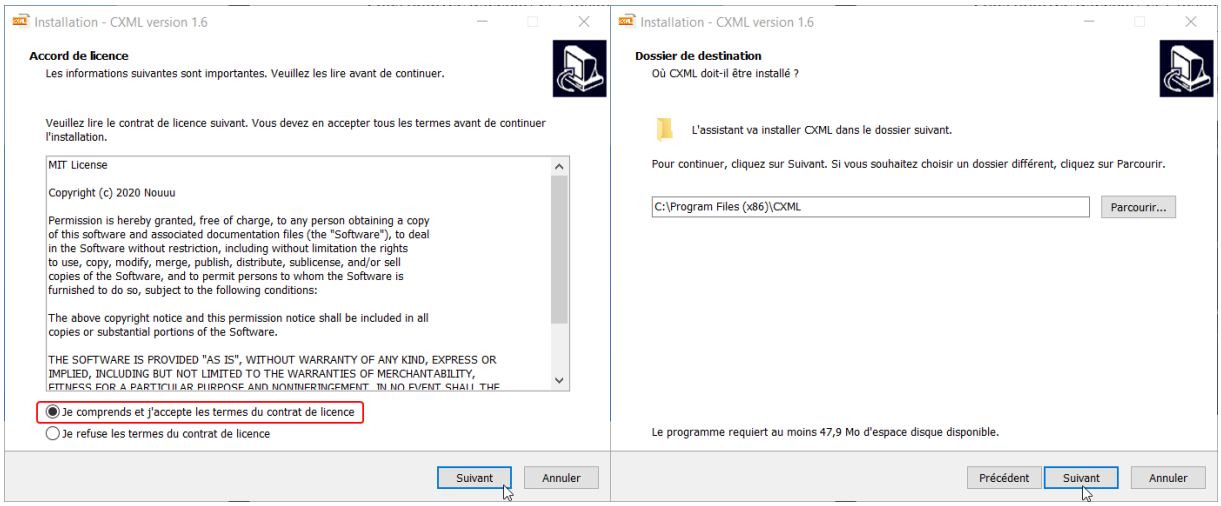
Pour Windows (le meilleur OS), c’est beaucoup plus simple !

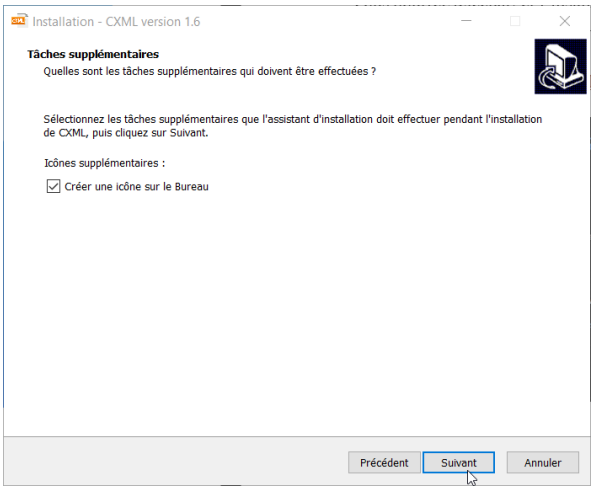
Vous pourrez télécharger l’installateur de l’application depuis ce lien, en faisant attention à bien prendre la version pour Windows.

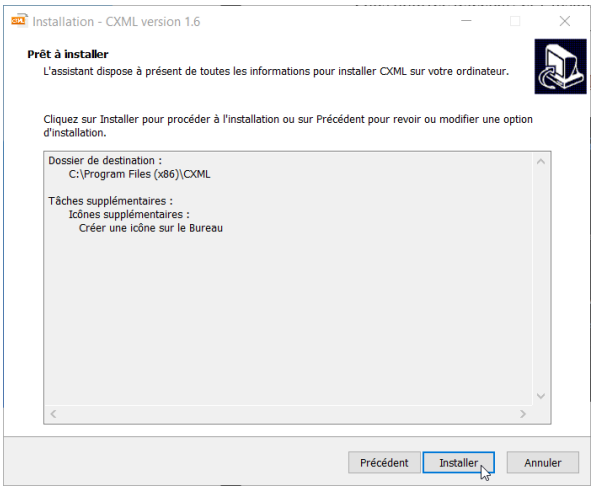
https://github.com/Nouuu/CXML/releases

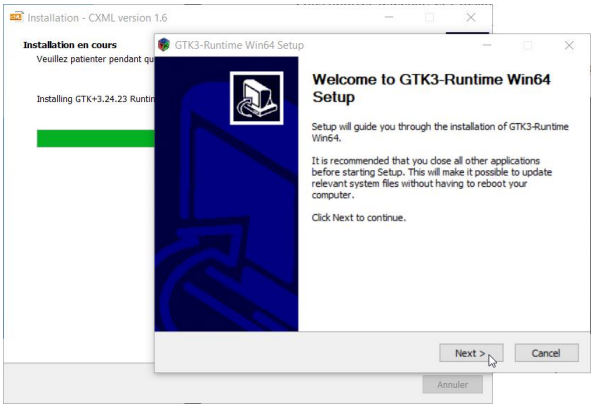
Lancez l’installateur et suivez toutes les étapes :

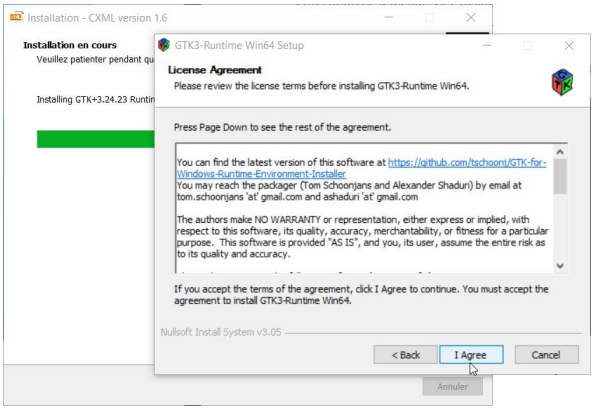


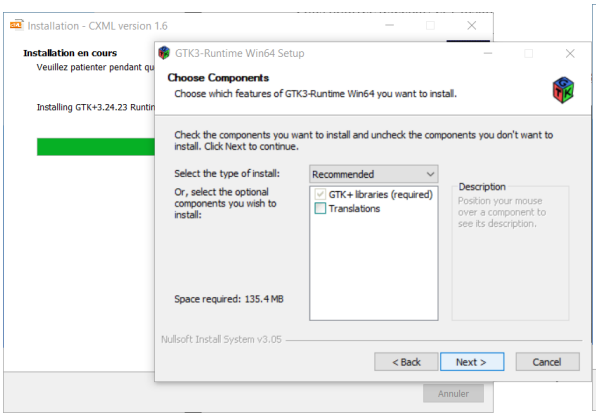


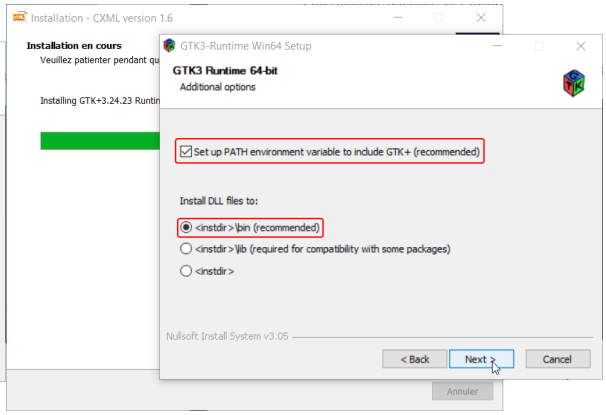

Installation - CXML version 1.6
Tâches supplémentaires
Quelles sont les tâches supplémentaires qui doivent être effectuées ?
Sélectionnez les tâches supplémentaires que l'assistant d'installation doit effectuer pendant l'installation de CXML, puis cliquez sur Suivant.
Icônes supplémentaires :
Créer une icône sur le Bureau
Précédent
Suivant
Annuler

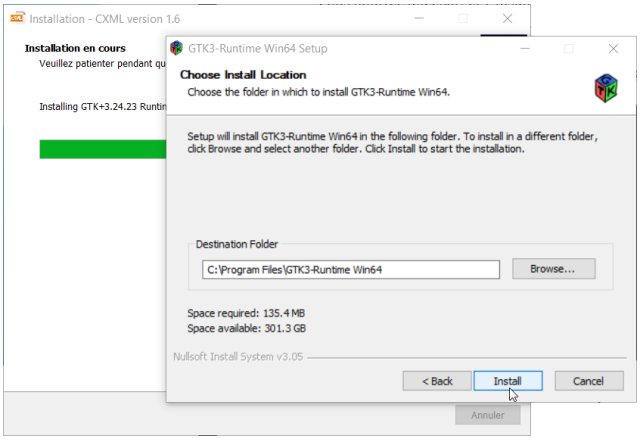

Installation - CXML version 1.6
Prêt à installer
L'assistant dispose à présent de toutes les informations pour installer CXML sur votre ordinateur.
Cliquez sur Installer pour procéder à l'installation ou sur Précédent pour revoir ou modifier une option d'installation.
Dossier de destination :
C:\Program Files (x86)\CXML
Tâches supplémentaires :
Icônes supplémentaires :
Créer une icône sur le Bureau
Précédent
Installer
Annuler

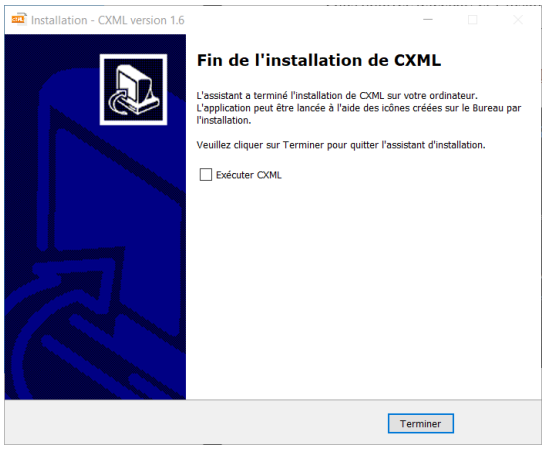

Installation - CXML version 1.6
Installation en cours
GTK3-Runtime Win64 Setup
Welcome to GTK3-Runtime Win64 Setup
Setup will guide you through the installation of GTK3-Runtime Win64.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel


Installation - CXML version 1.6
Installation en cours
GTK3-Runtime Win64 Setup
License Agreement
Please review the license terms before installing GTK3-Runtime Win64.
Press Page Down to see the rest of the agreement.
You can find the latest version of this software at https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer
You may reach the packager (Tom Schoonjans and Alexander Shaduri) by email at tom.schoonjans 'at' gmail.com and ashaduri 'at' gmail.com
The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install GTK3-Runtime Win64.
Nullsoft Install System v3.05
< Back
I Agree
Cancel


Installation - CXML version 1.6
Installation en cours
GTK3-Runtime Win64 Setup
Choose Components
Choose which features of GTK3-Runtime Win64 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select the type of install:
Recommended
Or, select the optional components you wish to install:
GTK+ libraries (required)
Translations
Description
Position your mouse over a component to see its description.
Space required: 135.4 MB
Nullsoft Install System v3.05
< Back
Next >
Cancel


Installation - CXML version 1.6
Installation en cours
GTK3-Runtime Win64 Setup
GTK3 Runtime 64-bit
Additional options
Set up PATH environment variable to include GTK+ (recommended)
Install DLL files to:
<instdir>\bin (recommended)
<instdir>\lib (required for compatibility with some packages)
<instdir>
Nullsoft Install System v3.05
< Back
Next >
Cancel


Installation - CXML version 1.6
Installation en cours
GTK3-Runtime Win64 Setup
Choose Install Location
Choose the folder in which to install GTK3-Runtime Win64.
Setup will install GTK3-Runtime Win64 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Program Files\GTK3-Runtime Win64
Browse...
Space required: 135.4 MB
Space available: 301.3 GB
Nullsoft Install System v3.05
< Back
Install
Cancel


Installation - CXML version 1.6
Fin de l'installation de CXML
L'assistant a terminé l'installation de CXML sur votre ordinateur.
L'application peut être lancée à l'aide des icônes créées sur le Bureau par l'installation.
Veuillez cliquer sur Terminer pour quitter l'assistant d'installation.
Exécuter CXML
Terminer

IV. DOSSIER D’UTILISATION

1. PARTIE 1 A 3 (MENU LIGNE DE COMMANDES)

L’application en ligne de commande peut s’exécuter de deux manières différentes.

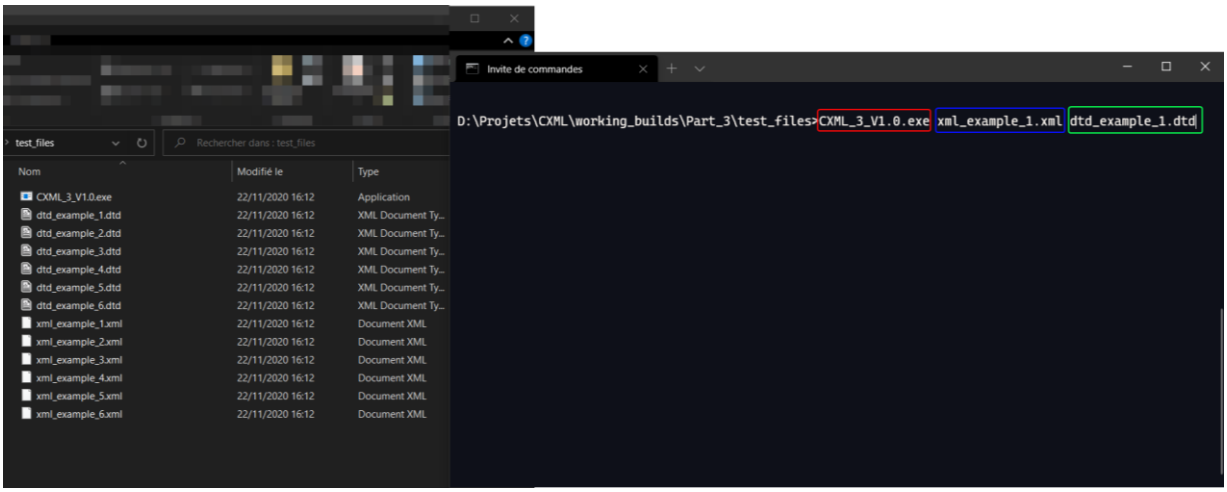
Exécution avec arguments

La première façon d’exécuter le programme est de le lancer en ligne de commande en mettant en argument :

Le chemin du fichier XML (relatif ou absolue)

Le chemin du fichier DTD (relatif ou absolue)

Si les arguments fournis à l’entrée du programme ne sont pas valides (mauvaise extension ou fichier inexistant, pas accessible en lecture), le programme va basculer automatiquement sur le menu pour renseigner des chemins valides.



Exécution avec le menu

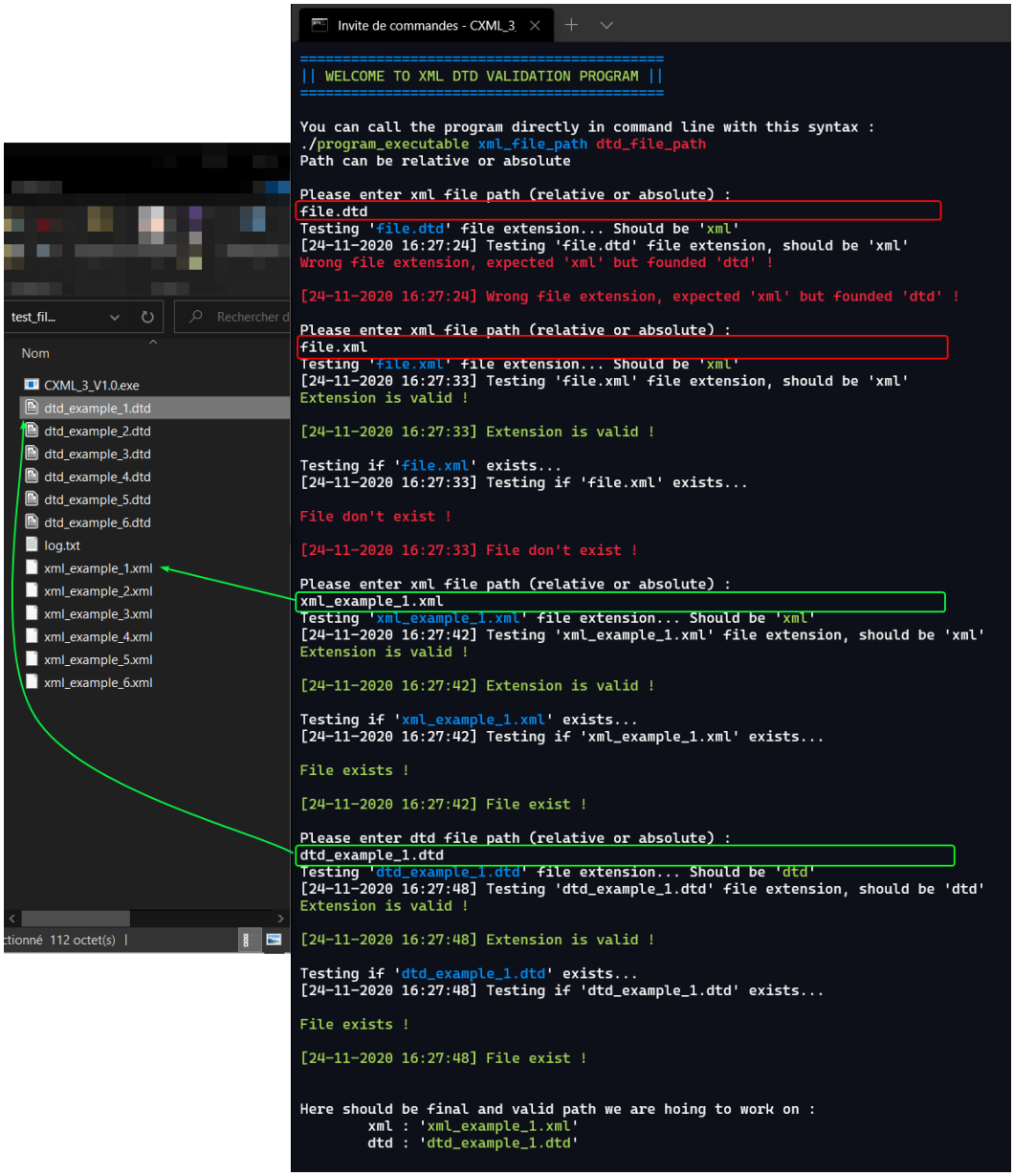
Il est également possible de lancer le programme en double cliquant dessus, ou bien en ligne de commande mais sans donner d’arguments.

Dans ce cas-là, le programme vous demandera de renseigner un chemin valide vers un fichier XML, puis un chemin valide vers un fichier DTD.

Le programme continuera de demander à l’utilisateur un chemin valide tant que celui-ci ne fournira pas un fichier :

Avec la bonne extension (**.xml** ou **.dtd**)

Qui existe bien sur le disque et accessible en lecture



Lancement de la validation

Une fois que les paramètres du programmes sont validés, celui-ci va automatiquement lancer le processus de validation.

Si tout se passe bien, on devrait avoir ce résultat :

```
Here should be final and valid path we are hoing to work on :
        xml : 'xml_example_1.xml'
        dtd : 'dtd_example_1.dtd'


Your xml document is conform to given dtd !

Press any key to leave
```

Dans le cas où le fichier XML n’est pas conforme à la DTD, un message d’erreur indiquant quelle règle n’est pas respecté apparait :

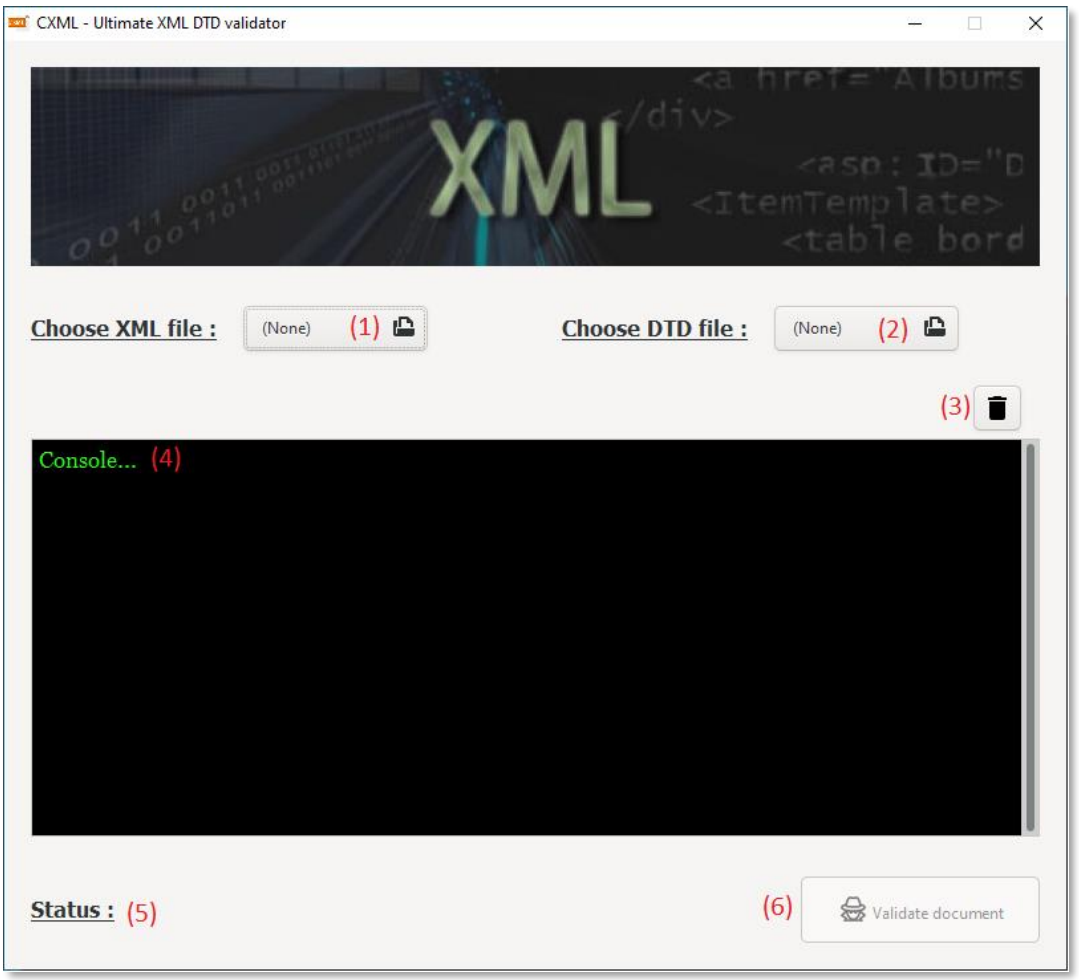
```
Here should be final and valid path we are hoing to work on :
        xml : 'xml_example_1.xml'
        dtd : 'dtd_example_2.dtd'

[24-11-2020 17:08:00] DTD Rule error - 'classrooms' element, 'classroom' child at position 2 not supposed to be here

Your xml document is not conform to given dtd !

Press any key to leave
```

Si le fichier XML n’est pas valide (syntaxiquement parlant) la validation DTD n’a pas lieu de se lancer et un message indiquant la ligne, la colonne et le type d’erreur est affiché :

```
Here should be final and valid path we are hoing to work on :
        xml : 'test_6.xml'
        dtd : 'dtd_example_1.dtd'

[24-11-2020 17:10:03] ERROR line 6 column 28 - |classroomAL</classroom| is not a valid tag name

Your xml document is not conform to given dtd !

Press any key to leave
```

2. PARTIE 4 (APPLICATION INTERFACE GRAPHIQUE GTK)

Voilà comment se présente l'application graphique. C'est ici que nous allons importer nos différents fichiers et vérifier leur exactitude.



(1) : Importer un fichier XML

(2) : Importer un fichier DTD

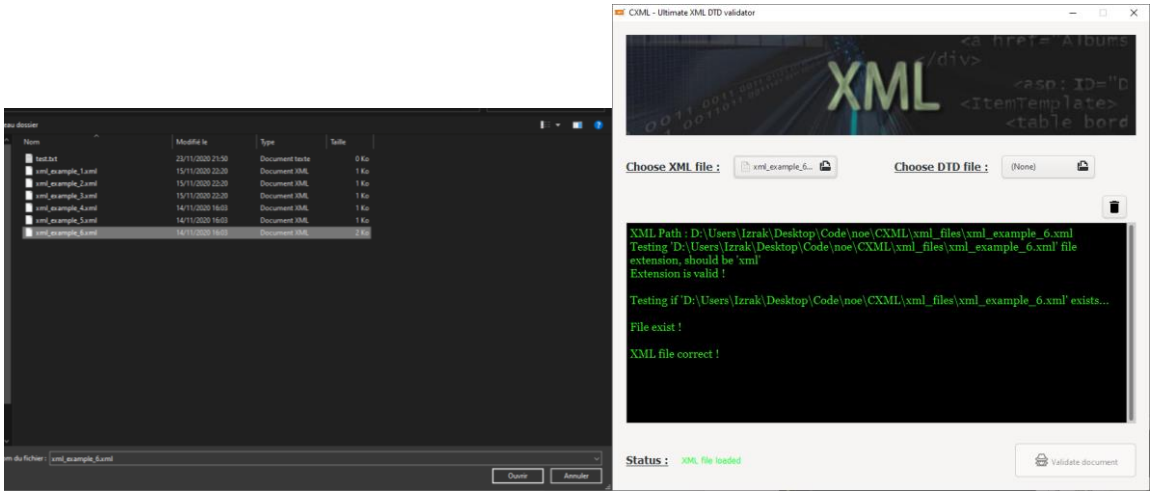
(3) : Vide la console

(4) : La console

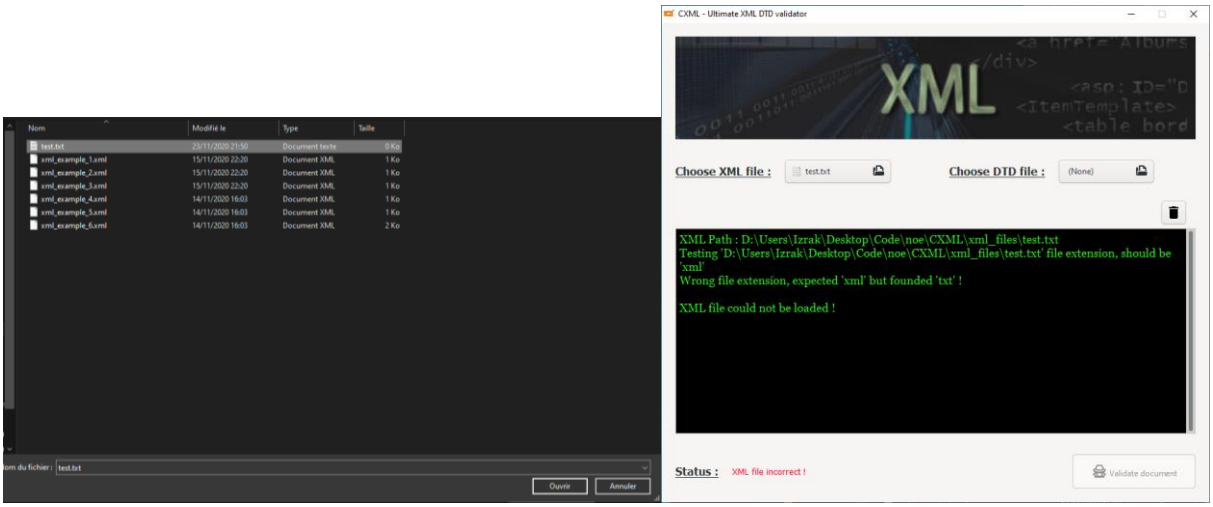
(5) : Affiche le statut de l’application

(6) : Bouton pour lancer la validation

Importation du fichier XML

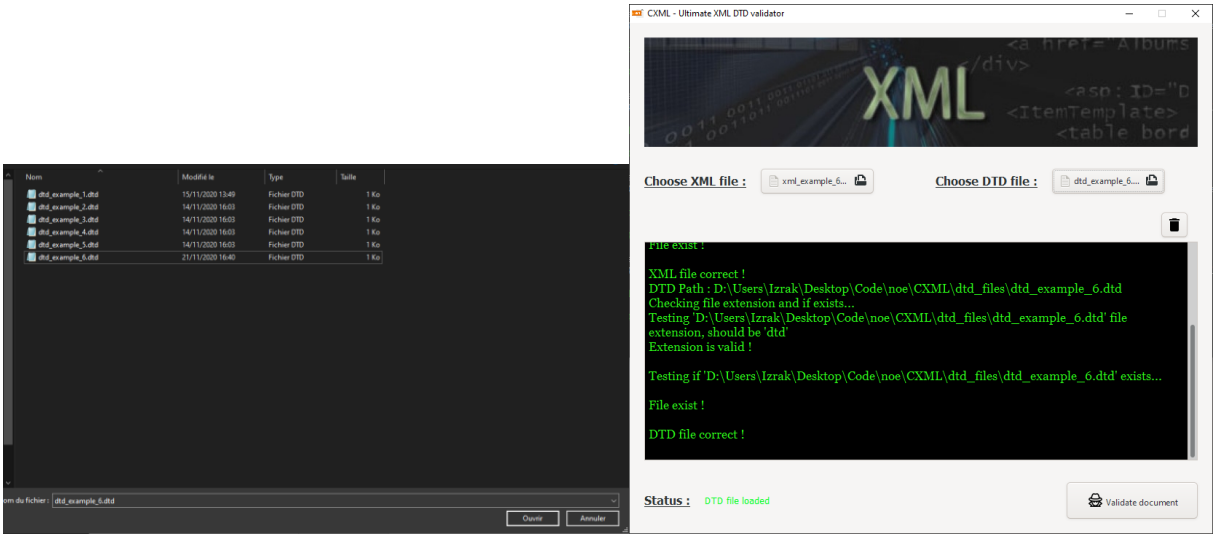


Importation d’un fichier autre que XML (Gestion d’erreur)

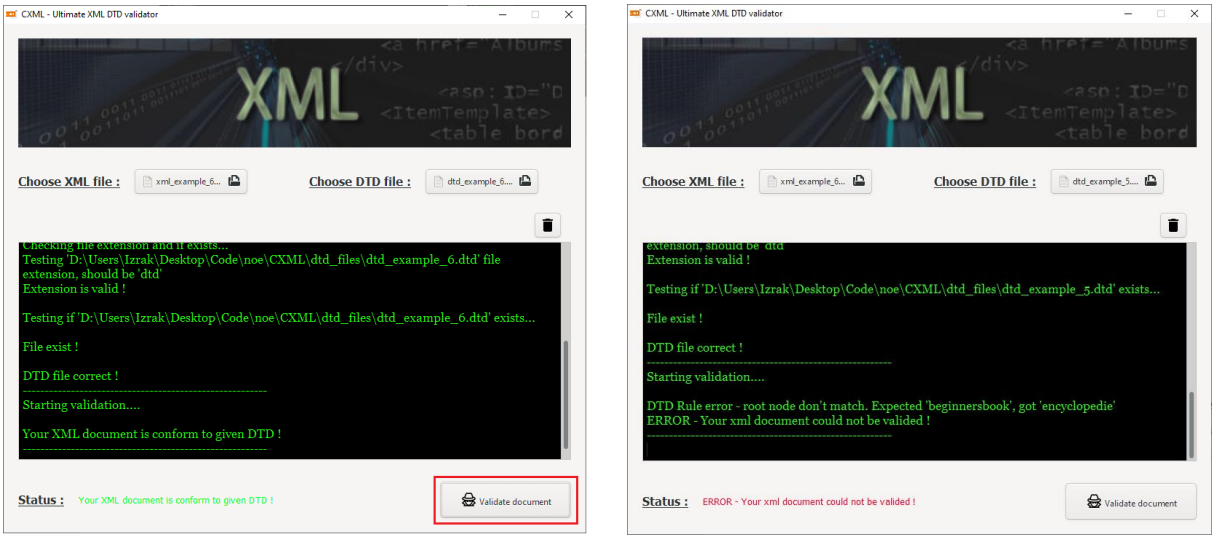


Si j’ouvre un fichier avec une autre extension que "**.xml**", une erreur s'affiche.

Importation du fichier DTD



Validation du document



Une fois tous les documents importés et validés par l'application, il ne vous reste plus qu'à presser le bouton "**Validate document**" afin de vérifier la conformité des documents.

On peut voir les traitements effectués par l'application dans la console. Une fois cela fini, il est clairement indiqué si oui ou non le fichier XML est conforme à la DTD.

Pour le champ « statuts » le code couleur est relativement simple, la couleur verte signifie que le traitement lancé s’est effectué avec succès et la couleur rouge signifie qu'il y a une erreur et donc que le fichier XML n'est pas conforme à la DTD.

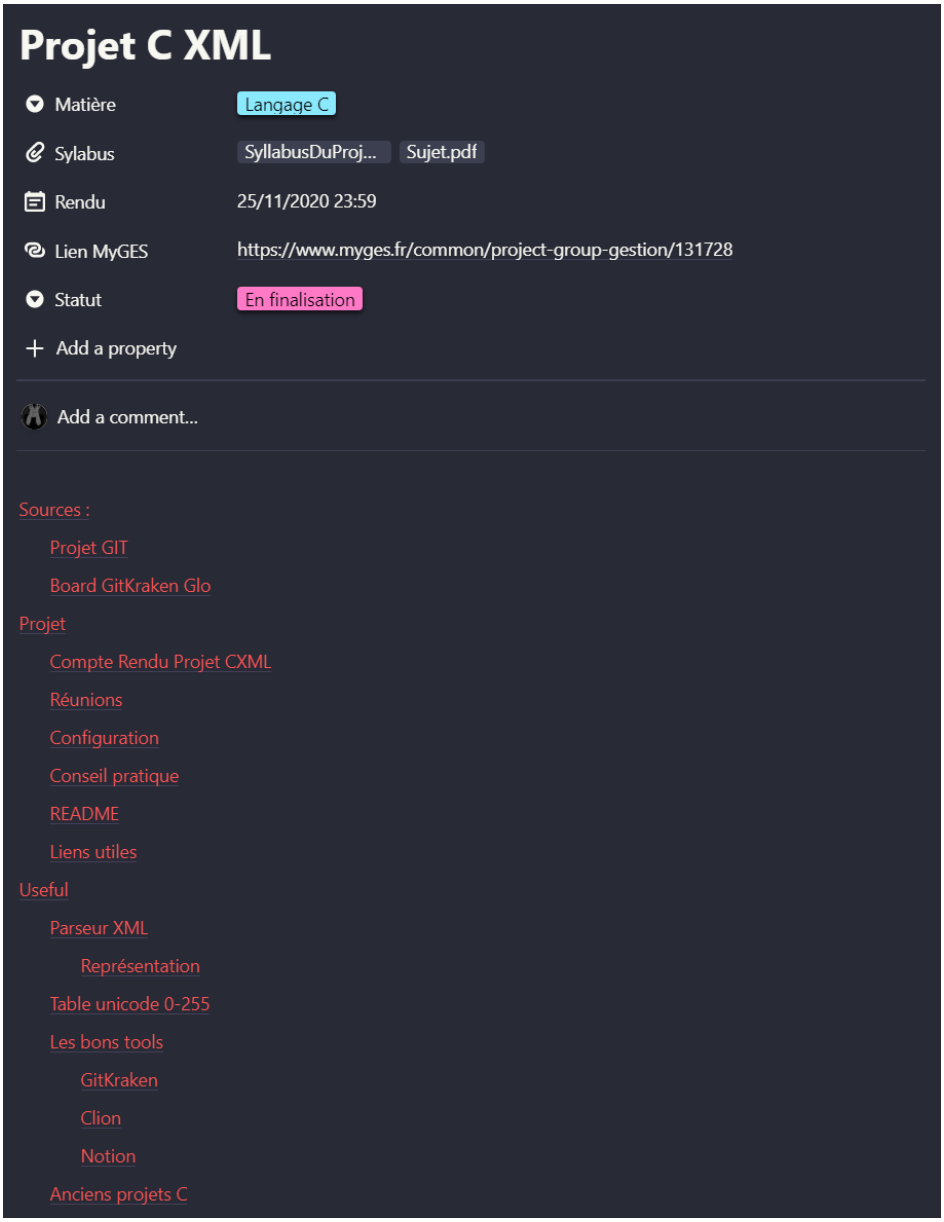
V. BILAN DU PROJET

1. APPRECIATION GENERALE ET ORGANISATION DU PROJET

Globalement le projet s'est bien déroulé. La bonne cohésion entre tous les membres du groupe nous a permis de le mener à bien. Sur les conseils de M. Briatte, nous n'avons pas attendu longtemps pour démarrer le projet. C'est pourquoi dès le début nous avons bien évidemment mis en place un dépôt sur GitHub ainsi qu’un board Kanban sur GitKraken.

Nous avons aussi mis en place un bloc note Notion commun pour y mettre les différentes informations et idées que nous trouvions sur le sujet.

Voici la table des matières de notre bloc note à la fin du projet :



Nous avons choisi comme angle d'attaque, de diviser les parties en un maximum de tâches pour simplifier la compréhension ainsi que la réalisation du projet. De cette manière, nous avons pu attribuer une tache à un membre de l'équipe et suivre de manière claire et précise l'avancée du projet.

Nous avons fait régulièrement des réunions pour voir l’état d’avancement de chacun, les difficultés rencontrées et établir de nouvelle répartition des tâches.

Exemple de compte rendu de réunion :

Points à aborder

- [x] ~~Vérifier si une balise est bien du type #PCDATA @Audrey~~
  - [x] ~~Return 1 si c'est bon, 0 sinon (et loglt l'erreur, si t'es chaud mettre dans le message le nom de la balise concernée et la règle DTD, en l'occurrence #PCDATA)~~
  - [x] ~~Parcourir la structure document xml pour trouver les balises contenant le nom cherché, puis vérifier que son "inner_text" n'est pas vide et qu'elle ne contient aucun enfants.~~
- [x] ~~Présentation du schéma sur le parseur et navigation dans le document XML à travers les structures. @Noé Larrieu-Lacoste~~
- [ ] Liste chainée pour les règles DTD @Isaac Ouslimane
- [ ] What about DTD parser ? @Noé Larrieu-Lacoste & @Isaac Ouslimane
- [ ] Mise au point installation MinGW @Audrey

Notes

Petite mise au point sur l'IDE et Gitkraken d'@Audrey (bug du cmake)

@Audrey actuellement sous compilation via visual studio et non MinGW

Changement de direction concernant la méthode de vérification #PCDATA

Prochaine fois

- Vérifier si une balise est bien du type #PCDATA @Audrey
  - Return 1 si c'est bon, 0 sinon (et loglt)
  - Parcourir la structure document xml pour trouver les balises contenant le nom cherché, puis vérifier que son "inner_text" n'est pas vide et qu'elle ne contient aucun enfants.

Pour conclure, nous pouvons voir ce projet comme une réussite était donné que les objectifs ont été pratiquement atteints grâce à la mise en place de différents outils de gestion de projet efficace qui nous permettait de ne pas nous perdre.

2. POINTS NON RESOLUS

Auto-complétions

Il est indiqué dans la partie 4 du projet que l'application doit proposer des suggestions des prochains éléments ou attributs lors de l'écriture de fichier XML, autrement dit une auto-complétions.

Cependant faute de temps c'est un point que nous n'avons malheureusement pas pu aborder.

Nous avons cependant réfléchi à une solution qui consiste à exploiter notre document DTD parsé, on pourrait alors déterminer sur quel balise l’utilisateur est en train d’éditer et lui proposer des complétions d’attributs ou d’enfants en fonction des règles associés au même nom de balise.

3. PROBLEMES RENCONTRES

Temps imparti

Le délai a respecté nous a posé quelques soucis, comme dit précédemment faute de temps nous n'avons pas pu finir complètement le projet. Pour organiser des points avec tous les membres de l'équipe nous avons dû composer avec les agendas de chacun.

Connaissances sur le sujet

Pour ce projet, nous connaissions très mal le XML et pas du tout la DTD. Il a donc fallu apprendre, comprendre toute les subtilités et s’assurer que toute les personnes du groupe ai compris.

Sur le langage C, nous n’étions pas tous au même niveau et comme le projet était assez conséquent, il a fallu faire de la mise à niveau au fur et à mesure que l’on attaquait des fonctionnalités avancées.

Méthodes de parsing

Aucun de nous n’a jamais vraiment fait de parsing ou même approcher la notion.

Il a donc fallu partir de 0, essayer, effacer, recommencer tout depuis le début de nombreuses fois avant de trouver un moyen viable de le faire…

Finalement il y a eu 5 versions intermédiaires très différentes pour parser un document avant d’avoir la méthode et les structures de données finales.

4. APPRECIATION INDIVIDUELLE

Audrey DI VITO

Ce projet m'a permis de m'améliorer en C et de mieux savoir gérer un projet. En effet, au début, il m'était inconcevable de savoir faire le projet, peu importe la partie concernée. Je ne connaissais pas du tout les fichiers DTD et concernant les fichiers XML, je savais que c’étaient des fichiers de configuration mais je n'avais jamais vraiment travaillé avec. J'ai aussi appris à faire une interface graphique en C, chose que je pensais possible qu’en C++.

Noé LARRIEU-LACOSTE

Etant déjà familier avec le langage, le défis a plus été pour moi de consolider mes connaissances et les approfondir. Cela s’est traduit par un gros investissement personnel sur certains points, notamment sur le parseur XML sur lequel j’ai passé beaucoup trop de temps à le rendre aussi parfait que possible.

Le fait de mettre un point d’honneur sur la compatibilité cross plateforme m’a forcé à investiguer sur les spécificités du langage sur chaque OS, notamment pour GTK et la coloration syntaxique dans un terminal.

Au final, j’ai vraiment eu le sentiment de progresser tout au long de ce projet, de gagner en compétence sur les techniques de développements et de gestion de projet.

Isaac OUSLIMANE

Comme je l'espérais ce projet m'a permis de progresser en C ainsi qu'en algorithmique. Au début du projet je me suis heurté à certaines difficultés dues à ma faible connaissance du C. J'ai rencontré des problèmes pour réaliser ce que je souhaitais faire, mais au fur et à mesure de recherches j'ai pu me débloquer.

D'un autre côté ce projet m'a également permis de progresser dans l'aspect gestion de projet. Il y a énormément de choses à apprendre pour gérer un projet et le mener à bien.