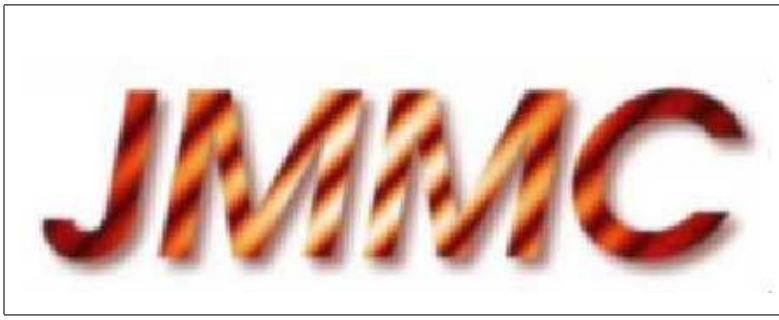

JMMC-MEM-2600-0002

Revision 1.0

Date: 07/04/2004

JMMC

Utilisation de librairies C pour parser des fichiers XML

Authors:
Sylvain Cetre <scetre@obs.ujf-grenoble.fr> — LAOG/JMMC

Change record

Revision	Date	Authors	Sections/Pages affected
	Remarks		
1.0	07/04/2005	S.Cetre	all
	première version		

Table des matières

1 Introduction

1.1 Object

De nombreuses applications utilisent pour l'échange des données le format XML. Portable, ce format permet de stocker de manière logique et universelle de l'information. Le but de ce document est d'expliquer comment extraire, stocker ou juste modifier des données en utilisant des parsers XML que l'on trouve dans différentes librairies C. Ces librairies s'appellent **expat**, **libxml2** ou encore **libgdome**. Ce document explique comment utiliser les librairies **libxml2** et **libgdome** en présentant les différentes étapes à suivre d'un point de vue programmation.
XML (Extensible Markup Language, ou Langage Extensible de Balisage) est, comme HTML (Hypertext Markup Language), un langage de balisage (markup), c'est-à-dire un langage qui présente de l'information encadrée par des balises. Mais contrairement à HTML, qui présente un jeu limité de balises orientées présentation (titre, paragraphe, image, lien hypertexte, etc.), XML est un métalangage, qui va permettre d'inventer à volonté de nouvelles balises pour isoler toutes les informations élémentaires

1.2 Reference documents

[1] **http ://www.xmlsoft.org/**, site officiel, The XML C parser and toolkit of Gnome libxml

[2] **http ://gdome2.cs.unibo.it/**, site internet, Gnome DOM Engine libgdome, a.k.a. gdome2

1.3 Abbreviations and acronyms

CDS	Centre de Données astronomiques de Strasbourg
XML	Extensible Markup Language
HTML	HyperText Markup Language
JMMC	Jean-Marie Mariotti Center
API	Application Programming Interface
DOM	Document Object Model
SAX	Simple API for XML

2 Les fichiers XML

2.1 La structure

Afin de visualiser un fichier XML, nous allons prendre un exemple de fichier XML que l'on peut récupérer au CDS, à l'aide d'un navigateur en entrant : **http ://vizier.u-strasbg.fr/viz-bin/asu-xml ?-source=I/280** ou dans une console en tapant : **GET /vizier.u-strasbg.fr/viz-bin/asu-xml ?-source=I/280**. Le CDS lui répond en envoyant un fichier XML. Les fichiers XML envoyés par le CDS sont structurées de manières suivantes :

1. des balises donnant la provenance du fichier
2. des balises de description des champs
3. une table des étoiles.

Une table d'étoile est situées dans **RESOURCE**. Si l'on souhaite extraire cette table d'étoile, il va falloir accéder à cette partie en se déplaçant en profondeur dans l'arbre XML. Schématiquement, la structure est la suivante :

```
<ASTRO>
        <DEFINITION>
        </DEFINITION>
        <RESOURCE>
                <NAME>
                </NAME>
                <TITLE>
                </TITLE>
                <TABLE>
                        <FIELD>
                        </FIELD>
                        ...
                        <DATA>
                        </DATA>
                </TABLE>
        </RESOURCE>
</ASTRO>
```

2.2 Les informations dans l'arbre XML

L'information à extraire est donc dans la partie **DATA** sous forme de table. L'organisation de l'information à l'intérieur de celle-ci est donnée plus haut dans l'arbre par les balises **FIELD** présentes au dessus et qui décrivent chacune des colonnes de la table. Chaque balise **FIELD** donne le nom d'une colonne, sa taille et le type de valeurs qui s'y trouve. Afin d'obtenir les renseignements sur chaque étoile, il faut dans un premier temps, repérer les balises FIELD, lire l'information qu'elles contiennent, repérer la balise DATA et la lire (ou la stocker) en fonction des renseignements obtenus précédemment. Il reste ensuite à analyser la table de données. Cela se fera par une analyse plus classique de Comma Separated Value. La description de l'organisation de l'information à l'intérieur de celle-ci est décrit dans les balises XML présentes avant dans le fichiers (**NAME, TITLE...**). Il est mis en évidence que la recherche d'informations dans cet arbre XML peut-être automatisé puisque la structure est "répétitive" (à chaque noeuds de l'arbre).

2.3 SAX/DOM

Il existe deux API majeures pour l'analyse de fichiers XML.
SAX : Simple API for XML, basée sur un modèle événementiel, cela signifie que SAX permet de déclencher

des événements au cours de l'analyse du document XML. Une application utilisant SAX implémente généralement des gestionnaires d'événements, lui permettant d'effectuer des opérations selon le type d'élément rencontré.
DOM : Document Object Model, traduisez modèle objet de document, est une spécification du W3C (World Wide Web Consortium) définissant la structure d'un document sous forme d'une hiérarchie d'objets, afin de simplifier l'accès aux éléments constitutifs du document.

Un fichier XML peut-être représenté par un arbre. Il a donc les caractéristiques d'un arbre : il possède une racine, et différents noeuds permettant d'accéder aux différentes branches de l'arbre. Il existe deux manières d'utiliser un fichier XML : soit le traitement du fichier se fait "à la volée"(SAX), soit il s'effectue en stockant l'arbre en mémoire (DOM). SAX à l'avantage de ne pas avoir à stocker l'arbre en mémoire lors de l'utilisation de fichiers volumineux : cela entraine un gain de temps grâce à une diminution des opérations. Par contre, la modification d'un fichier XML avec SAX n'est pas aisée car on ne peut se déplacer simplement entre les noeuds de l'arbre. Lors de l'utilisation de fichiers pas trop gros, comme ceux fournis par le CDS, il est intéressant d'utiliser DOM car une fois le fichier stocké en mémoire, il suffit de se "promener" dans l'arbre à l'aide des fonctions existantes dans les librairies. Il est présenté dans la suite de ce document comment mettre en oeuvre un outil d'analyse de ces fichiers XML (parser XML) à partir de librairies existantes.

3 Les librairies C

Les premiers test ont été codés en utilisant la librairie **expat**. cette librairie n'a pas retenue notre attention car elle reste compliquée à mettre en oeuvre. Il a donc fallu se tourner vers d'autres librairies offertes à la communauté. La première librairie, **libxml2**, offre un nombre impressionant de solutions. Elle reste cependant assez complexe. Enfin, il a été testé la librairie **libgdome**, qui, basée sur **libxml2**, offre toutes les fonctionnalités attendues, avec une simplicité en plus. Les explications suivantes indiquent comment analyser un document XML par une API DOM par l'utilisation de **libxml2** et **libgdome**.

3.1 Librairie libxml2

Nous allons étudier dans cette partie un exemple de programme qui utilise **libxml2**. L'exemple consiste à charger un arbre en mémoire (DOM) et à naviguer dans un arbre afin d'imprimer les noms des éléments qui le composent. Cet exemple est récupérable sur le site de **libxml2**. Nous avons le fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<memo>
        <de>John</de>
        <a>Bob</a>
        <contenu>
                <titre>Bonjour</titre>
                <texte>Ca va ?</texte>
</contenu>
</memo>
```

Code d'exemple :

```
#include<stdio.h>
#include<libxml/parser.h>
#include<libxml/tree.h>

#ifdef LIBXML_TREE_ENABLED


/**
 *print_element_names :
 */
static void print_element_names(xmlNode * a_node)
{
    xmlNode *cur_node = NULL;

    for (cur_node = a_node; cur_node; cur_node = cur_node->next) {
        if (cur_node->type == XML_ELEMENT_NODE) {
            printf("node type : Element, name : %s\n", cur_node->name);
        }

        print_element_names(cur_node->children);
    }
}


/**
 * Simple example to parse a file called "file.xml",
 * walk down the DOM, and print the name of the
```

```
 * xml elements nodes.
 */
int
main(int argc, char **argv)
{
    xmlDoc *doc = NULL;
    xmlNode *root_element = NULL;

    if (argc != 2)
        return(1);

    /*
     * this initialize the library and check potential ABI mismatches
     * between the version it was compiled for and the actual shared
     * library used.
     */
    LIBXML_TEST_VERSION

    /*parse the file and get the DOM */
    doc = xmlReadFile(argv[1], NULL, 0);

    if (doc == NULL) {
        printf("error : could not parse file %s\n", argv[1]);
    }

    /*Get the root element node */
    root_element = xmlDocGetRootElement(doc);

    print_element_names(root_element);

    /*free the document */
    xmlFreeDoc(doc);

    /*
     *Free the global variables that may
     *have been allocated by the parser.
     */
    xmlCleanupParser();

    return 0;
}
#else
int main(void) {
    fprintf(stderr, "Tree support not compiled in\n");
    exit(1);
}
#endif
```

Plaçons-nous au niveau du main du programme précédent (ligne 31). Il faut commencer par créer deux pointeurs (ligne 33-34) : un pointeur de xmlDoc et un pointeur d'élément (qui pointera plus tard sur la racine de l'arbre afin d'avoir un point d'entrée dans celui-ci).
A l'aide de la méthode

doc = xmlReadFile(argv[1], NULL, 0); (ligne 47)

nous chargeons le fichier passé en paramètre au programme (argv[1]). Il faut ensuite pouvoir entrer dans l'arbre. Pour cela, il faut faire pointer le pointeur créé précédemment sur la racine de l'arbre. On la récupère à l'aide de la méthode :

root_element = xmlDocGetRootElement(doc); (ligne 54)

On écrit ensuite la nom de l'élément root à l'aide de la fonction

print element names(root_element) ; (ligne 5)

qui est une fonction récursive. Cette fonction est codée ligne 11-23. Le principe est de créer un pointeur sur un noeud, de le placer sur celui passé en paramètre de la méthode, et de récupérer tous les noeuds présents au même niveau de l'arbre. Si on arrive à récupérer l'élément (test ligne 16) on écrit à l'écran la valeur de **cur_node->name**. On relance la méthode récursive sur chaque noeud parcouru. On affiche ainsi tous les noms des éléments de l'arbre.
En sortie du programme, il ne faut jamais oublier de libérer le document charger en mémoire.

xmlFreeDoc(doc) ; (ligne 59)

Le parser créé contient plusieurs variables qui ont pu être alouées par ce dernier. L'appel de la function

xmlCleanupParser() ; (ligne 65)

permet d'être sur d'avoir bien libéré toute la mémoire alouée. Le résultat obtenu dans un console est le suivant :

```
node type : Element, name : memo

node type : Element, name : de

node type : Element, name : a

node type : Element, name : contenu

node type : Element, name : titre

node type : Element, name : texte
```

3.2 Librairie gdome

Comme indiquer précédemment, libgdome offre une interface de **libxml2**, c'est à dire qu'elle est basée sur libxml2 : elle en reprend la structure de données générale, mais elle offre de nombreuses méthodes qui permettent une utilisation plus simple des outils de parsing.
Nous allons nous appuyer sur un exemple d'utilisation provenant du site officiel de **libgdome**. Afin de bien comprendre comment on analyse un fichier XML, nous allons en charger un en mémoire, le modifier, et l'enregistrer sur le disque.
Soit le fichier XML example.xml suivant (cette exemple est disponible sur le site de **libgdome**) :

```
<TEST>
        <NODE1/>
        <NODE2/>
        <NODE3/>
        <NODE4/>
        <NODE5/>
</TEST>
```

Voici le code source utilisé comme exemple :

```
#include <libgdome/gdome.h>

int main (int argc, char **argv) {
        GdomeDOMImplementation *domimpl;
        GdomeDocument *doc;
        GdomeElement *root, *el;
        GdomeNodeList *childs;
        GdomeException exc;
        GdomeDOMString *name, *value;
        unsigned long i, nchilds;                                                    10

        /* First I get a DOMImplementation reference */
        domimpl = gdome_di_mkref ();

        /* I load a new document from the file name "example.xml */
        doc = gdome_di_createDocFromURI(domimpl, LOCALDIR"/example.xml", GDOME_LOAD_PARSING, &exc);
        if (doc == NULL) {
                fprintf (stderr, "DOMImplementation.createDocFromURI : failed\n\tException #%d\n", exc);
                return 1;
        }                                                                            20

        /* I get reference to the root element of the document */
        root = gdome_doc_documentElement (doc, &exc);
        if (root == NULL) {
                fprintf (stderr, "Document.documentElement : NULL\n\tException #%d\n", exc);
                return 1;
        }

        /* I get the reference to the childrens NodeList of the root element */
        childs = gdome_el_childNodes (root, &exc);                                   30
        if (childs == NULL) {
                fprintf (stderr, "Element.childNodes : NULL\n\tException #%d\n", exc);
                return 1;
        }

        /* I add the attribute CHECKED="yes" to all element childs of the root element */
        name = gdome_str_mkref ("CHECKED");
        value = gdome_str_mkref ("yes");
        nchilds = gdome_nl_length (childs, &exc);
        for (i = 0; i < nchilds; i++) {                                              40
                el = (GdomeElement *)gdome_nl_item (childs, i, &exc);
                if (el == NULL) {
                        fprintf (stderr, "NodeList.item(%d) : NULL\n\tException #%d\n", (int)i, exc);
                        return 1;
                }
                if (gdome_el_nodeType (el, &exc) == GDOME_ELEMENT_NODE) {
                        gdome_el_setAttribute (el, name, value, &exc);
                        if (exc) {
                                fprintf (stderr, "Element.setAttribute : failed\n\tException #%d\n", exc);
                                return 1;                                            50
                        }
                }
                gdome_el_unref (el, &exc);
        }

        /* I save the modified document to a file named "example_out.xml */
        if (!gdome_di_saveDocToFile (domimpl, doc, LOCALDIR"/example_out.xml", GDOME_SAVE_STANDARD, &exc)) {
                fprintf (stderr, "DOMImplementation.saveDocToFile : failed\n\tException #%d\n", exc);
                return 1;
        }                                                                            60

        /* I free the document s_ture and the DOMImplementation */
        gdome_di_freeDoc (domimpl, doc, &exc);
```

```
        gdome_di_unref (domimpl, &exc);

        return 0;
}
```

La première étape de l'analyse consiste à créer une implementation DOM (ligne 13). L'implémentation fournit un ensemble de fonctions qui permettent de "faire" du DOM.
Il faut ensuite charger le fichier xml en mémoire. Cela s'effectue par l'appel de la fonction

doc = gdome_di_createDocFromURI(domimpl, LOCALDIR"/example.xml", GDOME_LOAD_PARSING, &exc) (ligne 16)

Cette fonction prend en paramètres l'implémentation définit précédemment, l'adresse du fichier à charger en mémoire, le mode de chargement du fichier (est-ce que l'on ne veut pouvoir charger que des fichiers valides par exemple).
Après ces deux étapes, le fichier XML est chargé en mémoire et les fonctions de modifications, de déplacement, etc... sont fournis grâce à l'implémentation DOM.
Pour pouvoir se promener dans l'arbre, il faut récupérer une référence sur sa racine. On l'obtient grâce à la fonction

root = gdome_doc_documentElement (doc, &exc) (ligne 23)

qui prend en paramètre "doc", l'arbre XML créé ci-dessus. La racine est un élément, au même titre que les autres éléments que l'on rencontre plus profond dans l'arbre.
A ce moment, nous avons donc l'arbre chargé en mémoire, une entrée dans cet arbre par sa racine. Nous allons voir ensuite comment lire toutes les branches et comment modifier si l'on le souhaite les données présentes dans cet arbre. Il a déjà été précisé qu'un parcours d'arbre est répétitif : en effet, à une profondeur donnée de l'arbre, on peut considérer que l'on est à une racine, celle du sous arbre qui commence à elle. Si l'on arrive à récupérer les enfants de la racine, il sera alors simple de lancer une récursivité afin d'atteindre tous les noeuds de l'arbre.
Pour récupérer tous les enfants d'un élément d'un noeud (ici, "root"), il faut appeler la fonction

childs = gdome_el_childNodes (root, &exc) (ligne 30)

Cette fonction renvoie la liste des enfants de l'élément. Le parcours de l'arbre sera réalisé en appliquant la même méthode à tous les éléments de la listes (ligne 40).

Si l'on veut en plus du parcours de l'arbre, modifier le fichier en marquant dans chaque élément l'information que l'élément a été vérifié, il faut procéder de la manière suivante :
il faut créer deux DOMString "name" et value" dans lesquelles nous marquons respectivement "VERIFIE" et "oui". En fonction du nombre d'enfants récupérés (ligne 39), nous affectons les deux DOMString (ligne 47) à un element si celui-ci a bien été reconnu par **libgdome** comme un GDOME_ELEMENT_NODE (ligne 46).
A ce moment, l'arbre stocké en mémoire a été modifié. Il est alors possible de sauver ces modifications dans le fichier example.xml que l'on a chargé au début par l'appel de la fonctions

gdome_di_saveDocToFile (domimpl, doc, LOCALDIR"/example_out.xml", GDOME_SAVE_STANDARD, &exc)

Nous obtenons alors dans le fichier résultat :

```
<TEST>
        <NODE1/ VERIFIE="oui">
        <NODE2/ VERIFIE="oui">
        <NODE3/ VERIFIE="oui">
        <NODE4/ VERIFIE="oui">
        <NODE5/ VERIFIE="oui">
</TEST>
```

Le fichier étant organisé en arbre, il est possible de répéter l'opération à chaque noeuds de manière récursive si contrairement à l'exemple il est présent plusieurs niveau de profondeur.
D'un point de vue gestion mémoire, **libgdome** fournit toutes les méthodes permettant de libérer toutes les références créées sur les objets. Cette libération est un inconvénient car elle nécessite de savoir exactement quels sont les objets utilisés dans le programme (ligne 53, 63, 64). La libération complète de la mémoire nécessite un volume de travail conséquent par rapport au temps de codage général du parser.

4 Conclusion

Les librairies C testées offrent toutes les fonctionnalités standards demandées. (elles répondent aux besoins pour parser un document du CDS par exemple). La différence essentielle réside dans la complexité d'utilisation. **libxml2**, qui offre certainement le plus de possibilités reste cependant complexe pour des utilisations simples. **libgdome** reste pour sa part une solution assez simple à mettre en oeuvre. Dans tous les cas, le problème réside dans la gestion de la mémoire : pour fournir un code propre, il est nécessaire d'effectuer un travail conséquent sur les libérations de mémoire.
D'autres solutions vers lesquelles se tourner sont les librairies C++ (Xerces) ou la gestion de la mémoire, qui posait problème en C devrait être résolue grâce à l'utilisation des constructeurs et des destructeurs. Ces librairies C++ sont malheureusement inutilisable dans le cadre de la programmation en C.