Libxml Tutorial

John Fleck <jfleck@inkstain.net>

Revision History

Revision 1	June 4, 2002	Initial draft
Revision 2	June 12, 2002	retrieving attribute value added
Revision 3	Aug. 31, 2002	freeing memory fix
Revision 4	Nov. 10, 2002	encoding discussion added
Revision 5	Dec. 15, 2002	more memory freeing changes
Revision 6	Jan. 26. 2003	add index
Revision 7	April 25, 2003	add compilation appendix

Table of Contents

Abstract

Libxml is a freely licensed C language library for handling XML, portable across a large number of platforms. This tutorial provides examples of its basic functions.

Introduction

Libxml is a C language library implementing functions for reading, creating and manipulating XML data. This tutorial provides example code and explanations

of its basic functionality.

Libxml and more details about its use are available on the project home page. Included there is complete API documentation. This tutorial is not meant to substitute for that complete documentation, but to illustrate the functions needed to use the library to perform basic operations.

The tutorial is based on a simple XML application I use for articles I write. The format includes metadata and the body of the article.

The example code in this tutorial demonstrates how to:

- Parse the document.
- Extract the text within a specified element.
- Add an element and its content.
- Add an attribute.
- Extract the value of an attribute.

Full code for the examples is included in the appendices.

Data Types

Libxml declares a number of data types we will encounter repeatedly, hiding the messy stuff so you do not have to deal with it unless you have some specific need.

xmlChar	A basic replacement for char, a byte in a UTF-8 encoded string. If your data uses another encoding, it must be converted to UTF-8 for use with libxml's functions. More information on encoding is available on the libxml encoding support web page.
xmlDoc	A structure containing the tree created by a parsed doc. xmlDocPtr is a pointer to the structure.
xmlNodePtr and xmlNode	A structure containing a single node. xmlNodePtr is a pointer to the structure, and is used in traversing the document tree.

Parsing the file

Parsing the file requires only the name of the file and a single function call, plus error checking. Full code: Appendix C, *Code for Keyword Example*

```
❶ xmlDocPtr doc;
❷ xmlNodePtr cur;
```

```
❸ doc = xmlParseFile(docname);

❹ if (doc == NULL ) {
        fprintf(stderr,"Document not parsed successfully. \n");
        return;
}

❺ cur = xmlDocGetRootElement(doc);

❻ if (cur == NULL) {
        fprintf(stderr,"empty document\n");
        xmlFreeDoc(doc);
        return;
}

❼ if (xmlStrcmp(cur->name, (const xmlChar *) "story")) {
        fprintf(stderr,"document of the wrong type, root node != story");
        xmlFreeDoc(doc);
        return;
}
```

❶ Declare the pointer that will point to your parsed document.
❷ Declare a node pointer (you'll need this in order to interact with individual nodes).
❹ Check to see that the document was successfully parsed. If it was not, libxml will at this point register an error and stop.

Note

One common example of an error at this point is improper handling of encoding. The XML standard requires documents stored with an encoding other than UTF-8 or UTF-16 to contain an explicit declaration of their encoding. If the declaration is there, libxml will automatically perform the necessary conversion to UTF-8 for you. More information on XML's encoding requirements is contained in the standard.

❺ Retrieve the document's root element.
❻ Check to make sure the document actually contains something.
❼ In our case, we need to make sure the document is the right type. "story" is the root type of the documents used in this tutorial.

Retrieving Element Content

Retrieving the content of an element involves traversing the document tree until you find what you are looking for. In this case, we are looking for an element called "keyword" contained within element called "story". The process to find the node we are interested in involves tediously walking the tree. We assume you already have an xmlDocPtr called `doc` and an xmlNodPtr called `cur`.

```
❶cur = cur->xmlChildrenNode;
❷while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *)"storyinfo"))){
                parseStory (doc, cur);
        }

cur = cur->next;
}
```

❶ Get the first child node of `cur`. At this point, `cur` points at the document root, which is the element "story".

❷ This loop iterates through the elements that are children of "story", looking for one called "storyinfo". That is the element that will contain the key-"words" we are looking for. It uses the libxml string comparison function, `xmlStrcmp`. If there is a match, it calls the function `parseStory`.

```
void
parseStory (xmlDocPtr doc, xmlNodePtr cur) {

        xmlChar *key;
        ❶ cur = cur->xmlChildrenNode;
        ❷ while (cur != NULL) {
            if ((!xmlStrcmp(cur->name, (const xmlChar *)"keyword"))) {
        ❸     key = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);
                  printf("keyword: %s\n", key);
                  xmlFree(key);
             }
        cur = cur->next;
        }
    return;
}
```

❶ Again we get the first child node.

❷ Like the loop above, we then iterate through the nodes, looking for one that matches the element we're interested in, in this case "keyword".

❸ When we find the "keyword" element, we need to print its contents. Remember that in XML, the text contained within an element is a child node of that element, so we turn to `cur->xmlChildrenNode`. To retrieve it, we use the function `xmlNodeListGetString`, which also takes the `doc` pointer as an argument. In this case, we just print it out.

Note

Because `xmlNodeListGetString` allocates memory for the string it returns, you must use `xmlFree` to free it.

Writing element content

Writing element content uses many of the same steps we used above — parsing the document and walking the tree. We parse the document, then traverse the tree to find the place we want to insert our element. For this example, we want to again find the "storyinfo" element and this time insert a keyword. Then we'll write the file to disk. Full code: Appendix D, *Code for Add Keyword Example*

The main difference in this example is in `parseStory`:

```
void
parseStory (xmlDocPtr doc, xmlNodePtr cur, char *keyword) {

        ❶ xmlNewTextChild (cur, NULL, "keyword", keyword);
    return;
}
```

❶ The `xmlNewTextChild` function adds a new child element at the current

node pointer's location in the tree, specified by `cur`.

Once the node has been added, we would like to write the document to file. Is you want the element to have a namespace, you can add it here as well. In our case, the namespace is NULL.

```
xmlSaveFormatFile (docname, doc, 1);
```

The first parameter is the name of the file to be written. You'll notice it is the same as the file we just read. In this case, we just write over the old file. The second parameter is a pointer to the xmlDoc structure. Setting the third parameter equal to one ensures indenting on output.

Writing Attribute

Writing an attribute is similar to writing text to a new element. In this case, we'll add a reference URI to our document. Full code:Appendix E, *Code for Add Attribute Example*.

A `reference` is a child of the `story` element, so finding the place to put our new element and attribute is simple. As soon as we do the error-checking test in our `parseDoc`, we are in the right spot to add our element. But before we do that, we need to make a declaration using a data type we have not seen yet:

```
xmlAttrPtr newattr;
```

We also need an extra xmlNodePtr:

```
xmlNodePtr newnode;
```

The rest of `parseDoc` is the same as before until we check to see if our root element is `story`. If it is, then we know we are at the right spot to add our element:

```
❶ newnode = xmlNewTextChild (cur, NULL, "reference", NULL);
❷ newattr = xmlNewProp (newnode, "uri", uri);
```

❶ First we add a new node at the location of the current node pointer, `cur`. using the xmlNewTextChild function.

Once the node is added, the file is written to disk just as in the previous example in which we added an element with text content.

Retrieving Attributes

Retrieving the value of an attribute is similar to the previous example in which we retrieved a node's text contents. In this case we'll extract the value of the URI we added in the previous section. Full code: Appendix F, *Code for Retrieving Attribute Value Example*.

The initial steps for this example are similar to the previous ones: parse the doc, find the element you are interested in, then enter a function to carry out the spe-

cific task required. In this case, we call `getReference`:

```
void
getReference (xmlDocPtr doc, xmlNodePtr cur) {

        xmlChar *uri;
        cur = cur->xmlChildrenNode;
        while (cur != NULL) {
            if ((!xmlStrcmp(cur->name, (const xmlChar *)"reference"))) {
                  ❶ uri = xmlGetProp(cur, "uri");
                   printf("uri: %s\n", uri);
                   xmlFree(uri);
            }
            cur = cur->next;
        }
        return;
}
```

❶ The key function is `xmlGetProp`, which returns an `xmlChar` containing the attribute's value. In this case, we just print it out.

Note

If you are using a DTD that declares a fixed or default value for the attribute, this function will retrieve it.

Encoding Conversion

Data encoding compatibility problems are one of the most common difficulties encountered by programmers new to XML in general and libxml in particular. Thinking through the design of your application in light of this issue will help avoid difficulties later. Internally, libxml stores and manipulates data in the UTF-8 format. Data used by your program in other formats, such as the commonly used ISO-8859-1 encoding, must be converted to UTF-8 before passing it to libxml functions. If you want your program's output in an encoding other than UTF-8, you also must convert it.

Libxml uses iconv if it is available to convert data. Without iconv, only UTF-8, UTF-16 and ISO-8859-1 can be used as external formats. With iconv, any format can be used provided iconv is able to convert it to and from UTF-8. Currently iconv supports about 150 different character formats with ability to convert from any to any. While the actual number of supported formats varies between implementations, every iconv implementation is almost guaranteed to support every format anyone has ever heard of.

Warning

A common mistake is to use different formats for the internal data in different parts of one's code. The most common case is an application that assumes ISO-8859-1 to be the internal data format, combined with libxml, which assumes UTF-8 to be the internal data format. The result is an application that treats internal data differently, depending on which code section is executing. The one or the other part of code will then, naturally, misinterpret the data.

This example constructs a simple document, then adds content provided at the command line to the document's root element and outputs the results to `stdout` in the proper encoding. For this example, we use ISO-8859-1 encoding. The en-

coding of the string input at the command line is converted from ISO-8859-1 to UTF-8. Full code: Appendix G, *Code for Encoding Conversion Example*

The conversion, encapsulated in the example code in the `convert` function, uses libxml's `xmlFindCharEncodingHandler` function:

```
        ❶xmlCharEncodingHandlerPtr handler;
        ❷size = (int)strlen(in)+1;
        out_size = size*2-1;
        out = malloc((size_t)out_size);

…
        ❸handler = xmlFindCharEncodingHandler(encoding);
…
        ❹handler->input(out, &out_size, in, &temp);
…
        ❺xmlSaveFormatFileEnc("-", doc, encoding, 1);
```

❶ `handler` is declared as a pointer to an `xmlCharEncodingHandler` function.

❷ The `xmlCharEncodingHandler` function needs to be given the size of the input and output strings, which are calculated here for strings `in` and `out`.

❸ `xmlFindCharEncodingHandler` takes as its argument the data's initial encoding and searches libxml's built-in set of conversion handlers, returning a pointer to the function or NULL if none is found.

❹ The conversion function identified by `handler` requires as its arguments pointers to the input and output strings, along with the length of each. The lengths must be determined separately by the application.

❺ To output in a specified encoding rather than UTF-8, we use `xmlSaveFormatFileEnc`, specifying the encoding.

A. Compilation

Libxml includes a script, xml2-config, that can be used to generate flags for compilation and linking of programs written with the library. For pre-processor and compiler flags, use **xml2-config --cflags**. For library linking flags, use **xml2-config --libs**. Other options are available using **xml2-config --help**.

B. Sample Document

```
<?xml version="1.0"?>
<story>
  <storyinfo>
    <author>John Fleck</author>
    <datewritten>June 2, 2002</datewritten>
    <keyword>example keyword</keyword>
  </storyinfo>
  <body>
    <headline>This is the headline</headline>
    <para>This is the body text.</para>
  </body>
</story>
```

C. Code for Keyword Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

void
parseStory (xmlDocPtr doc, xmlNodePtr cur) {

        xmlChar *key;
        cur = cur->xmlChildrenNode;
        while (cur != NULL) {
            if ((!xmlStrcmp(cur->name, (const xmlChar *)"keyword"))) {
                    key = xmlNodeListGetString(doc, cur->xmlChildrenNode, 1);
                    printf("keyword: %s\n", key);
                    xmlFree(key);
             }
        cur = cur->next;
        }
    return;
}

static void
parseDoc(char *docname) {

        xmlDocPtr doc;
        xmlNodePtr cur;

        doc = xmlParseFile(docname);

        if (doc == NULL ) {
                fprintf(stderr,"Document not parsed successfully. \n");
                return;
        }

        cur = xmlDocGetRootElement(doc);

        if (cur == NULL) {
                fprintf(stderr,"empty document\n");
                xmlFreeDoc(doc);
                return;
        }

        if (xmlStrcmp(cur->name, (const xmlChar *) "story")) {
                fprintf(stderr,"document of the wrong type, root node != story");
                xmlFreeDoc(doc);
                return;
        }

        cur = cur->xmlChildrenNode;
        while (cur != NULL) {
                if ((!xmlStrcmp(cur->name, (const xmlChar *)"storyinfo"))){
                        parseStory (doc, cur);
                }

        cur = cur->next;
        }

        xmlFreeDoc(doc);
        return;
}

int
main(int argc, char **argv) {

        char *docname;

        if (argc <= 1) {
                printf("Usage: %s docname\n", argv[0]);
                return(0);
```

```
        }

        docname = argv[1];
        parseDoc (docname);

        return (1);
}
```

D. Code for Add Keyword Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

void
parseStory (xmlDocPtr doc, xmlNodePtr cur, char *keyword) {

        xmlNewTextChild (cur, NULL, "keyword", keyword);
    return;
}

xmlDocPtr
parseDoc(char *docname, char *keyword) {

        xmlDocPtr doc;
        xmlNodePtr cur;

        doc = xmlParseFile(docname);

        if (doc == NULL ) {
                fprintf(stderr,"Document not parsed successfully. \n");
                return (NULL);
        }

        cur = xmlDocGetRootElement(doc);

        if (cur == NULL) {
                fprintf(stderr,"empty document\n");
                xmlFreeDoc(doc);
                return (NULL);
        }

        if (xmlStrcmp(cur->name, (const xmlChar *) "story")) {
                fprintf(stderr,"document of the wrong type, root node != story");
                xmlFreeDoc(doc);
                return (NULL);
        }

        cur = cur->xmlChildrenNode;
        while (cur != NULL) {
                if ((!xmlStrcmp(cur->name, (const xmlChar *)"storyinfo"))){
                        parseStory (doc, cur, keyword);
                }

        cur = cur->next;
        }
        return(doc);
}

int
main(int argc, char **argv) {

        char *docname;
        char *keyword;
```

```
        xmlDocPtr doc;

        if (argc <= 2) {
                printf("Usage: %s docname, keyword\n", argv[0]);
                return(0);
        }

        docname = argv[1];
        keyword = argv[2];
        doc = parseDoc (docname, keyword);
        if (doc != NULL) {
                xmlSaveFormatFile (docname, doc, 0);
                xmlFreeDoc(doc);
        }

        return (1);
}
```

E. Code for Add Attribute Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>


xmlDocPtr
parseDoc(char *docname, char *uri) {

        xmlDocPtr doc;
        xmlNodePtr cur;
        xmlNodePtr newnode;
        xmlAttrPtr newattr;

        doc = xmlParseFile(docname);

        if (doc == NULL ) {
                fprintf(stderr,"Document not parsed successfully. \n");
                return (NULL);
        }

        cur = xmlDocGetRootElement(doc);

        if (cur == NULL) {
                fprintf(stderr,"empty document\n");
                xmlFreeDoc(doc);
                return (NULL);
        }

        if (xmlStrcmp(cur->name, (const xmlChar *) "story")) {
                fprintf(stderr,"document of the wrong type, root node != story");
                xmlFreeDoc(doc);
                return (NULL);
        }

        newnode = xmlNewTextChild (cur, NULL, "reference", NULL);
        newattr = xmlNewProp (newnode, "uri", uri);
        return(doc);
}

int
main(int argc, char **argv) {

        char *docname;
        char *uri;
```

```
        xmlDocPtr doc;

        if (argc <= 2) {
                printf("Usage: %s docname, uri\n", argv[0]);
                return(0);
        }

        docname = argv[1];
        uri = argv[2];
        doc = parseDoc (docname, uri);
        if (doc != NULL) {
                xmlSaveFormatFile (docname, doc, 1);
                xmlFreeDoc(doc);
        }
        return (1);
}
```

F. Code for Retrieving Attribute Value Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

void
getReference (xmlDocPtr doc, xmlNodePtr cur) {

        xmlChar *uri;
        cur = cur->xmlChildrenNode;
        while (cur != NULL) {
            if ((!xmlStrcmp(cur->name, (const xmlChar *)"reference"))) {
                    uri = xmlGetProp(cur, "uri");
                    printf("uri: %s\n", uri);
                    xmlFree(uri);
            }
            cur = cur->next;
        }
        return;
}


void
parseDoc(char *docname) {

        xmlDocPtr doc;
        xmlNodePtr cur;

        doc = xmlParseFile(docname);

        if (doc == NULL ) {
                fprintf(stderr,"Document not parsed successfully. \n");
                return;
        }

        cur = xmlDocGetRootElement(doc);

        if (cur == NULL) {
                fprintf(stderr,"empty document\n");
                xmlFreeDoc(doc);
                return;
        }

        if (xmlStrcmp(cur->name, (const xmlChar *) "story")) {
```

```
                fprintf(stderr,"document of the wrong type, root node != story");
                xmlFreeDoc(doc);
                return;
        }

        getReference (doc, cur);
        xmlFreeDoc(doc);
        return;
}

int
main(int argc, char **argv) {

        char *docname;

        if (argc <= 1) {
                printf("Usage: %s docname\n", argv[0]);
                return(0);
        }

        docname = argv[1];
        parseDoc (docname);

        return (1);
}
```

G. Code for Encoding Conversion Example

```
#include <string.h>
#include <libxml/parser.h>


unsigned char*
convert (unsigned char *in, char *encoding)
{
        unsigned char *out;
        int ret,size,out_size,temp;
        xmlCharEncodingHandlerPtr handler;

        size = (int)strlen(in)+1;
        out_size = size*2-1;
        out = malloc((size_t)out_size);

        if (out) {
                handler = xmlFindCharEncodingHandler(encoding);

                if (!handler) {
                        free(out);
                        out = NULL;
                }
        }
        if (out) {
                temp=size-1;
                ret = handler->input(out, &out_size, in, &temp);
                if (ret || temp-size+1) {
                        if (ret) {
                                printf("conversion wasn't successful.\n");
                        } else {
                                printf("conversion wasn't successful. converted: %i oct
                        }
                        free(out);
                        out = NULL;
                } else {
                        out = realloc(out,out_size+1);
```

```
                        out[out_size]=0; /*null terminating out*/

                }
        } else {
                printf("no mem\n");
        }
        return (out);
}


int
main(int argc, char **argv) {

        unsigned char *content, *out;
        xmlDocPtr doc;
        xmlNodePtr rootnode;
        char *encoding = "ISO-8859-1";


        if (argc <= 1) {
                printf("Usage: %s content\n", argv[0]);
                return(0);
        }

        content = argv[1];

        out = convert(content, encoding);

        doc = xmlNewDoc ("1.0");
        rootnode = xmlNewDocNode(doc, NULL, (const xmlChar*)"root", out);
        xmlDocSetRootElement(doc, rootnode);

        xmlSaveFormatFileEnc("-", doc, encoding, 1);
        return (1);
}
```

H. Acknowledgements

A number of people have generously offered feedback, code and suggested improvements to this tutorial. In no particular order: Daniel Veillard, Marcus Labib Iskander, Christopher R. Harris, Igor Zlatkovic, Niraj Tolia

Index

X