# 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 |
| Revision 8 | July 24, 2003 |
| | add XPath example |
| Revision 9 | Feb. 14, 2004 |
| | Fix bug in XPath example |
| Revision 7 | Aug. 24, 2004 |
| | Fix another bug in XPath example |

## 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*

```
¶ 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;
}
```

¶ Declare the pointer that will point to your parsed document.
Declare a node pointer (you'll need this in order to interact with individual nodes).
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.
Retrieve the document's root element.
Check to make sure the document actually contains something.
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`.

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

        cur = cur->next;
        }
```

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

• This loop iterates through the elements that are children of "story", looking for one called "storyinfo". That is the element that will contain the "keywords" 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;
        ¶ 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;
}
```

¶ Again we get the first child node.

• 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".

, 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.

## Using XPath to Retrieve Element Content

In addition to walking the document tree to find an element, Libxml2 includes support for use of XPath expressions to retrieve sets of nodes that match a specified criteria. Full documentation of the XPath API is here.

XPath allows searching through a document for nodes that match specified criteria. In the example below we search through a document for the contents of all `keyword` elements.

## Note

A full discussion of XPath is beyond the scope of this document. For details on its use, see the XPath specification.

Full code for this example is at Appendix D, *Code for XPath Example*.

Using XPath requires setting up an xmlXPathContext and then supplying the XPath expression and the context to the `xmlXPathEvalExpression` function. The function returns an xmlXPathObjectPtr, which includes the set of nodes satisfying the XPath expression.

```
xmlXPathObjectPtr
getnodeset (xmlDocPtr doc, xmlChar *xpath){

¶xmlXPathContextPtr context;
xmlXPathObjectPtr result;

•context = xmlXPathNewContext(doc);
,result = xmlXPathEvalExpression(xpath, context);
„if(xmlXPathNodeSetIsEmpty(result->nodesetval)){
        xmlXPathFreeObject(result);
        printf("No result\n");
        return NULL;
```

¶ First we declare our variables.
• Initialize the `context` variable.
, Apply the XPath expression.
„ Check the result and free the memory allocated to `result` if no result is found.

The xmlPathObjectPtr returned by the function contains a set of nodes and other information needed to iterate through the set and act on the results. For this example, our functions returns the `xmlXPathObjectPtr`. We use it to print the contents of `keyword` nodes in our document. The node set object includes the number of elements in the set (`nodeNr`) and an array of nodes (`nodeTab`):

```
¶for (i=0; i < nodeset->nodeNr; i++) {
•keyword = xmlNodeListGetString(doc, nodeset->nodeTab[i]->xmlChildrenN
        printf("keyword: %s\n", keyword);
        xmlFree(keyword);
}
```

¶ The value of `nodeset->Nr` holds the number of elements in the node set. Here we use it to iterate through the array.
• Here we print the contents of each of the nodes returned.