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*

```
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