libxmlquery Documentation

Release 0.1.4

Frederico Gonçalves, Vasco Fernandes

November 06, 2010

CONTENTS

libxmlquery is a colection of C functions to help **parse**, **create** and **query** xml documents.

If you're new to **libxmlquery** start by reading the *Tutorial*.

CHAPTER

ONE

TUTORIAL

The following code shows how to use the library to rename a node in a xml file:

```
#include <stdio.h>
#include <xmlquery/parse.h>
#include <xmlquery/dom.h>
#include <xmlquery/serialize.h>

dom_node* node = xmlquery_parsefile("test.xml");
node_list* q_result = xmlquery_query("@hi[foo=bar]");
dom_node* n_result = xmlquery_pop_node(q_result);
xmlquery_set_node_name(n_result, "bye");

printf("%s\n", xmlquery_dom_node_to_string(node, XML);
```

if `test.xml` contains:

```
<root>
  <hi />
  <hi foo="bar" />
</root>
```

the resulting output would change the node:

```
<hi foo="bar" />
```

to:

```
<bye foo="bar" />
```

this code will read an xml file in `test.xml` into our abstract representation of the DOM (**D**ocument **O**bject **M**odel) tree:

```
dom_node* node = xmlquery_parsefile("test.xml");
```

CHAPTER

TWO

DOCUMENT OBJECT MODEL

dom_node
Abstract representation of an xml node.

2.1 DOM

After the parsing of the xml is done, a DOM structure is created and can be accessed through a global variable denominated “document”. Bear in mind that this variable is dependent of the current parser. In other words, if the parser module changes, then the variable name may also change.

The DOM structure is a tree, where each node is an xml element. Since the input xml has already been validated we know it has only one root , so the document variable will point to a single node in the dom tree, the root node (This can be accessed through document->root).

The dom structure provides flexible and extendable xml document handling. It has the capability to add, remove and modify elements in the xml. The next section will explain in deeper detail the DOM module functions.

2.2 DOM functions

To create a dom structure, the function new_document should be called. If the input xml has the form:

```
<?xml version="1.0"?>
<root>
  ...
</root>
```

And you wish to store the <?xml version=”1.0” ?>, you can do this by creating a dom_node with the name “xml” and attribute “version”. This dom_node must be passed to the function new_document. This is done to keep the dom structure simpler and with only one root. As we said before, this implementation makes the assumption that the xml has already been validated.

Although we provide in the header file the structures dom_node and doc, thus exposing their implementation, we highly recommend that you use the proper access functions to obtain each field. If we change the structures, we will not change the functions’ signature, only their implementation if needed. By using the access methods, you won’t need to change any code done. For the same reason, we recommend that you use the proper setter functions to store the values in each node.

CHAPTER

THREE

QUERYING

Querying xml nodes to provide quick *random* access to inner nodes is implemented through CSS3 selectors.

Note: CSS stands for cascading style sheets, it is primarily used for styling (x)html documents as method to separate content from presentation.

The CSS3 selectors specification is a W3C proposed recomendation widely implemented by most modern browsers, it has also been used, in its current and earlier forms (CSS and CSS2), by web developers for over 13 years.

The W3C recomendation does not tie CSS rules to the presentation of of (x)html documents, in fact it does define it as a query language capable of selecting xml/html nodes. With the rise in popularity of AJAX technologies several javascript libraries became widely used, these libraries (most notabily JQuery) first proposed the utilization of CSS rules not only for presentation but also to help manipulation and transversion of the DOM.

This library embraces CSS rules as a better/simpler alternative to manual transversion as well as other W3C recomendations such as XPath and XQuery.

3.1 Selectors

The function *xmlquery_query* is defined to help filter and match nodes in the *DOM tree*:

generic_list* **xmlquery_query** (const char* *pattern*, dom_node* *root*)

dom_node* **xmlquery_query_one** (const char* *pattern*, dom_node* *root*)

This is a helper function designed to facilitate selection of single nodes. Selects the first node returned by a given query, it is equivalent to the following code:

```
dom_node* single = NULL;
generic_list* l = xmlquery_query(patterm, root)
if((l != NULL && l->count > 0)
    single = (dom_node*) get_element_at(l, 0);
```

3.1.1 Introduction

Node selection is implemented as a **pattern string** to be matched against a *DOM tree*. The pattern string matches some number of nodes that are then returned as a flat list.

A pattern example:

```
foo
```

this simple pattern matches all nodes named *foo*. Upon matching the DOM tree with this pattern, a reference to every node named *foo* would be returned. Independent of their position in the tree. Returned references may be manipulated, rearanged or deleted.

Pattern strings match dom nodes in three different realms, according to the node name (as we've seen), according to the node attributes and according to *pseudo-filters*, a special kind of selectors aware of the node context.

```
foo[attr='bar']:only-child
```

A more complex selector, this selector will match nodes with the name *foo*, with an attribute named *attr* with the value *bar*, and the node must also be the *only child* of its parent node.

CHAPTER

FOUR

DATA STRUCTURES

4.1 Generic List

`generic_list`

Generic array list optimized for fast access.

4.1.1 List

4.1.2 Stack

4.1.3 Queue

4.2 Generic Red Black Tree

You may also find specific object documentation in the *genindex*.

INDEX