

# The LUAXML library

Paul Chakravarti      Michal Hoftich

Version 0.11  
2019-09-16

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The DOM_Object library</b>	<b>2</b>
2.1	Node selection methods . . . . .	3
2.1.1	The DOM_Object:get_path method . . . . .	3
2.1.2	The DOM_Object:query_selector method . . . . .	3
2.2	Element traversing . . . . .	4
2.2.1	The DOM_Object:traverse_elements method . . . . .	4
2.3	DOM modifications . . . . .	4
<b>3</b>	<b>The CssQuery library</b>	<b>5</b>
3.1	Example usage . . . . .	5
<b>4</b>	<b>The API documentation</b>	<b>6</b>
4.1	luaxml-domobject . . . . .	6
4.1.1	Class: function . . . . .	6
4.1.2	Class: Class DOM_Object . . . . .	7
4.2	luaxml-cssquery . . . . .	11
4.2.1	Class: function . . . . .	11
4.2.2	Class: Class CssQuery . . . . .	11
<b>5</b>	<b>Low-level functions usage</b>	<b>12</b>
5.1	The simpleTreeHandler . . . . .	12
5.2	The domHandler . . . . .	14
<b>I</b>	<b>Original LuaXML documentation by Paul Chakravarti</b>	<b>16</b>
<b>6</b>	<b>Overview</b>	<b>16</b>
<b>7</b>	<b>Features</b>	<b>16</b>
<b>8</b>	<b>Limitations</b>	<b>16</b>

<b>9 API</b>	<b>17</b>
<b>10 Options</b>	<b>18</b>
<b>11 Usage</b>	<b>18</b>
<b>12 Handlers</b>	<b>18</b>
12.1 Overview . . . . .	18
12.2 Features . . . . .	19
12.3 API . . . . .	19
12.3.1 printHandler . . . . .	19
12.3.2 domHandler . . . . .	19
12.3.3 simpleTreeHandler . . . . .	19
12.4 Options . . . . .	20
12.5 Usage . . . . .	20
<b>13 History</b>	<b>20</b>
<b>14 License</b>	<b>20</b>

## 1 Introduction

LuaXML is pure lua library for processing and serializing of the `xml` files. The base code has been written by Paul Chakravarti, with minor changes which brings Lua 5.3 or HTML 5 support. On top of that, new modules for accessing the `xml` files using DOM like methods or CSS selectors<sup>1</sup> have been added.

The documentation is divided to three parts – first part deals with the `DOM` library, second part describes the low-level libraries and the third part is original documentation by Paul Chakravarti.

## 2 The `DOM_Object` library

This library can process a `xml` sources using DOM like functions. To load it, you need to require `luaxml-domobject.lua` file. The `parse` function provided by the library creates `DOM_Object` object, which provides several methods for processing the `xml` tree.

```
local dom = require "luaxml-domobject"
local document = [[
<html>
<head><title>sample</title></head>
<body>
<h1>test</h1>
<p>hello</p>
</body>
]]
```

---

<sup>1</sup>Thanks to Leaf Corcoran for CSS selector parsing code.

```

</html>
[]

-- dom.parse returns the DOM_Object
local obj = dom.parse(document)
-- it is possible to call methods on the object
local root_node = obj:root_node()
for _, x in ipairs(root_node:get_children()) do
    print(x:get_element_name())
end

```

The details about available methods can be found in the API docs, section 4.1. The above code will load a `xml` document, it will get the ROOT element and print all it's children element names. The `DOM_Object:get_children` function returns Lua table, so it is possible to loop over it using standard table functions.

html

## 2.1 Node selection methods

There are some other methods for element retrieving.

### 2.1.1 The `DOM_Object:get_path` method

If you want to print text content of all child elements of the body element, you can use `DOM_Object:get_path`:

```

local path = obj:get_path("html body")
for _, el in ipairs(path[1]:get_children()) do
    print(el:get_text())
end

```

The `DOM_Object:get_path` function always return array with all elements which match the requested path, even if there is only one such element. In this case, it is possible to use standard Lua table indexing to get the first and only one matched element and get it's children using `DOM_Object:get_children` method. If the children node is an element, it's text content is printed using `DOM_Object:get_text`.

test  
 hello

### 2.1.2 The `DOM_Object:query_selector` method

This method uses CSS `selector` syntax to select elements, similarly to JavaScript `jQuery` library.

```
for _, el in ipairs(obj:query_selector("h1,p")) do
    print(el:get_text())
end
```

```
test
hello
```

It supports also XML namespaces, using `namespace|element` syntax.

## 2.2 Element traversing

### 2.2.1 The `DOM_Object:traverse_elements` method

It may be useful to traverse over all elements and apply a function on all of them.

```
obj:traverse_elements(function(node)
    print(node:get_text())
end)
```

```
sample test hello
sample test hello
sample
sample
test hello
test
hello
```

The `get_text` method gets text from all children elements, so the first line shows all text contained in the `<html>` element, the second one in `<head>` element and so on.

## 2.3 DOM modifications

It is possible to add new elements, text nodes, or to remove them.

```
local headers = obj:query_selector("h1")
for _, header in ipairs(headers) do
    header:remove_node()
end
-- query selector returns array, we must retrieve the first element
-- to get the actual body element
local body = obj:query_selector("body")[1]
local paragraph = body:create_element("p", {})
body:add_child_node(paragraph)
paragraph:add_child_node(paragraph:create_text_node("This is a second paragraph"))
```

```

for _, el in ipairs(body:get_children()) do
    if el:is_element() then
        print(el:get_element_name().. ":" .. el:get_text())
    end
end

```

In this example, <h1> element is being removed from the sample document, and new paragraph is added. Two paragraphs should be shown in the output:

```

p: hello
p: This is a second paragraph

```

### 3 The CssQuery library

This library serves mainly as a support for the `DOM_Object:query_selector` function. It also supports adding information to the DOM tree.

#### 3.1 Example usage

```

local cssobj = require "luaxml-cssquery"
local domobj = require "luaxml-domobject"

local xmldata = [[
<html>
<body>
<h1>Header</h1>
<p>Some text, <i>italics</i></p>
</body>
</html>
]]]

local dom = domobj.parse(xmldata)
local css = cssobj()

css:add_selector("h1", function(obj)
    print("header found: " .. obj:get_text())
end)

css:add_selector("p", function(obj)
    print("paragraph found: " .. obj:get_text())
end)

css:add_selector("i", function(obj)
    print("found italics: " .. obj:get_text())
end)

dom:traverse_elements(function(el)

```

```
-- find selectors that match the current element
local querylist = css:match_querylist(el)
-- add templates to the element
css:apply_querylist(el,querylist)
end)
```

header found: Header  
 paragraph found: Some text, italics  
 found italics: italics

More complete example may be found in the `examples` directory in the LuaXML source code repository<sup>2</sup>.

## 4 The API documentation

### 4.1 luaxml-domobject

DOM module for LuaXML

#### 4.1.1 Class: function

**serialize\_dom(parser, current, level, output)**

It serializes the DOM object back to the XML.

**Parameters:**

`parser`: DOM object

`current`: Element which should be serialized

`level`:

`output`:

**Return:**

table Table with XML strings. It can be concatenated using `table.concat()` function to get XML string corresponding to the DOM\_Object.

**parse(xmltext, voidElements)**

XML parsing function Parse the XML text and create the DOM object.

**Parameters:**

`xmltext`:

`voidElements`: hash table with void elements

**Return:**

DOM\_Object

---

<sup>2</sup><https://github.com/michal-h21/LuaXML/blob/master/examples/xmltotex.lua>

#### 4.1.2 Class: Class DOM\_Object

**DOM\_Object:root\_node()**

Returns root element of the DOM\_Object

**Parameters:**

**Return:**

DOM\_Object

**DOM\_Object:get\_node\_type(el)**

Get current node type

**Parameters:**

el: [optional] node to get the type of

**DOM\_Object:is\_element(el)**

Test if the current node is an element.

**Parameters:**

el: [optional] element to test

**Return:**

boolean

**DOM\_Object:is\_text(el)**

Test if current node is text

**Parameters:**

el: [optional] element to test

**Return:**

boolean

**DOM\_Object:get\_element\_name(el)**

Return name of the current element

**Parameters:**

el: [optional] element to test

**Return:**

string

**DOM\_Object:get\_attribute(name)**

Get value of an attribute

**Parameters:**

name: Attribute name

**Return:**

string

**DOM\_Object:set\_attribute(name, value)**

Set value of an attribute

**Parameters:**

name:

value: Value to be set

**Return:**

boolean

**DOM\_Object:serialize(current)**

Serialize the current node back to XML

**Parameters:**

current: [optional] element to be serialized

**Return:**

string

**DOM\_Object:get\_text(current)**

Get text content from the node and all of it's children

**Parameters:**

current: [optional] element which should be converted to text

**Return:**

string

**DOM\_Object:get\_path(path, current)**

Retrieve elements from the given path.

**Parameters:**

path:

current: [optional] element which should be traversed. Default element is the root element of the DOM\_Object

**Return:**

table of elements which match the path

**DOM\_Object:query\_selector(selector)**

Select elements chidlren using CSS selector syntax

**Parameters:**

selector: String using the CSS selector syntax

**Return:**

table with elements matching the selector.

**DOM\_Object:get\_children(el)**

Get table with children of the current element

**Parameters:**

el: [optional] element to be selected

**Return:**

table with children of the selected element

**DOM\_Object:get\_parent(el)**

Get the parent element

**Parameters:**

el: [optional] element to be selected

**Return:**

DOM\_Object parent element

**DOM\_Object:traverse\_elements(fn, current)**

Execute function on the current element and all it's children elements.

**Parameters:**

fn: function which will be executed on the current element and all it's children

current: [optional] element to be selected

**Return:**

nothing

**DOM\_Object:traverse\_node\_list(nodelist, fn)**

Execute function on list of elements returned by DOM\_Object:get\_path()

**Parameters:**

nodelist:

fn: function to be executed

**DOM\_Object:replace\_node(new)**

Replace the current node with new one

**Parameters:**

new: element which should replace the current element

**Return:**

boolean, message

**DOM\_Object:add\_child\_node(child, position)**

Add child node to the current node

**Parameters:**

child: element to be inserted as a current node child

position: [optional] position at which should the node be inserted

**DOM\_Object:copy\_node(element)**

Create copy of the current node

**Parameters:**

element: [optional] element to be copied

**Return:**

DOM\_Object element

**DOM\_Object:create\_element(name, attributes, parent)**

Create a new element

**Parameters:**

name: New tag name

attributes: Table with attributes

parent: [optional] element which should be saved as the element's parent

**Return:**

DOM\_Object element

**DOM\_Object:create\_text\_node(text, parent)**

Create new text node

**Parameters:**

text: string

parent: [optional] element which should be saved as the element's parent

**Return:**

DOM\_Object text object

**DOM\_Object:remove\_node(element)**

Delete current node

**Parameters:**

element: [optional] element to be removed

**DOM\_Object:find\_element\_pos(el)**

Find the element position in the current node list

**Parameters:**

el: [optional] element which should be looked up

**Return:**

integer position of the current element in the element table

**DOM\_Object:get\_siblings(el)**

Get node list which current node is part of

**Parameters:**

el: [optional] element for which the sibling element list should be retrieved

**Return:**

table with elements

**DOM\_Object:get\_sibling\_node(change)**

Get sibling node of the current node

**Parameters:**

change: Distance from the current node

**Return:**

DOM\_Object node

**DOM\_Object:get\_next\_node(el)**

Get next node

**Parameters:**

el: [optional] node to be used

**Return:**

DOM\_Object node

**DOM\_Object:get\_prev\_node(el)**

Get previous node

**Parameters:**

el: [optional] node to be used

**Return:**

DOM\_Object node

## 4.2 luaxml-cssquery

CSS query module for LuaXML

### 4.2.1 Class: function

**cssquery()**

CssQuery constructor

**Parameters:**

**Return:**

CssQuery object

### 4.2.2 Class: Class CssQuery

**CssQuery:calculate\_specificity(query)**

Calculate CSS specificity of the query

**Parameters:**

query: table created by CssQuery:prepare\_selector() function

**Return:**

integer specificity value

**CssQuery:match\_querylist(domobj, querylist)**

Test prepared querylist

**Parameters:**

domobj: DOM element to test

querylist: [optional] List of queries to test

**Return:**

table with CSS queries, which match the selected DOM element

**CssQuery:get\_selector\_path(domobj, selectorlist)**

Get elements that match the selector

**Parameters:**

domobj: DOM\_Object

selectorlist: prepare\_selector

**Return:**

table with DOM\_Object elements

**CssQuery:prepare\_selector(selector)**

Parse CSS selector to a query table.

**Parameters:**

selector: string CSS selector query

**Return:**

table querylist

### **CssQuery:add\_selector(selector, func, params)**

Add selector to CSS object list of selectors, func is called when the selector matches a DOM object params is table which will be passed to the func

#### **Parameters:**

**selector:** CSS selector string

**func:** function which will be executed on matched elements

**params:** table with parameters for the function

#### **Return:**

integer number of elements in the prepared selector

### **CssQuery:sort\_querylist(querylist)**

Sort selectors according to their specificity It is called automatically when the selector is added

#### **Parameters:**

**querylist:** [optional] querylist table

#### **Return:**

querylist table

### **CssQuery:apply\_querylist(domobj, querylist)**

It tests list of queries agaings a DOM element and executes the coresponding function that is saved for the matched query.

#### **Parameters:**

**domobj:** DOM element

**querylist:** querylist table

#### **Return:**

nothing

## **5 Low-level functions usage**

The original LuaxML library provides some low-level functions for XML handling. First of all, we need to load the libraries:

```
xml = require('luaxml-mod-xml')
handler = require('luaxml-mod-handler')
```

The luaxml-mod-xml file contains the xml parser and also the serializer. In luaxml-mod-handler, various handlers for dealing with xml data are defined. Handlers transforms the xml file to data structures which can be handled from the Lua code. More information about handlers can be found in the original documentation, section 12.

### **5.1 The simpleTreeHandler**

```
sample = [[
<a>
```

```

<d>hello</d>
<b>world.</b>
<b at="Hi">another</b>
</a>]]
treehandler = handler.simpleTreeHandler()
x = xml.xmlParser(treehandler)
x:parse(sample)

```

You have to create handler object, using `handler.simpleTreeHandler()` and `xml parser` object using `xml.xmlParser(handler object)`. `simpleTreehandler` creates simple table hierarchy, with top root node in `treehandler.root`

```

-- pretty printing function
function printable(tb, level)
    level = level or 1
    local spaces = string.rep(' ', level*2)
    for k,v in pairs(tb) do
        if type(v) ~= "table" then
            print(spaces .. k..'= '..v)
        else
            print(spaces .. k)
            level = level + 1
            printable(v, level)
        end
    end
end

-- print table
printable(treehandler.root)
-- print xml serialization of table
print(xml.serialize(treehandler.root))
-- direct access to the element
print(treehandler.root["a"]["b"][1])

```

This code produces the following output:

```

output:
a
  d=hello
  b
    1=world.
    2
      1=another
      _attr
        at=Hi
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <d>hello</d>
  <b>world.</b>
  <b at="Hi">

```

```

        another
    </b>
</a>

world.
```

First part is pretty-printed dump of Lua table structure contained in the handler, the second part is `xml` serialized from that table and the last part demonstrates direct access to particular elements.

Note that `simpleTreeHandler` creates tables that can be easily accessed using standard lua functions, but if the `xml` document is of mixed-content type<sup>3</sup>:

```

<a>hello
    <b>world</b>
</a>
```

then it produces wrong results. It is useful mostly for data `xml` files, not for text formats like `xhtml`.

## 5.2 The `domHandler`

For complex `xml` documents, it is best to use the `domHandler`, which creates object which contains all information from the `xml` document.

```
-- file dom-sample.lua
-- next line enables scripts called with texlua to use luatex libraries
--kpse.set_program_name("luatex")
function traverseDom(current,level)
    local level = level or 0
    local spaces = string.rep(" ",level)
    local root= current or current.root
    local name = root._name or "unnamed"
    local xtype = root._type or "untyped"
    local attributes = root._attr or {}
    if xtype == "TEXT" then
        print(spaces .."TEXT : " .. root._text)
    else
        print(spaces .. xtype .. " : " .. name)
    end
    for k, v in pairs(attributes) do
        print(spaces .. " .. k..=\""..v)
    end
    local children = root._children or {}
    for _, child in ipairs(children) do
        traverseDom(child, level + 1)
    end
end
```

---

<sup>3</sup>This means that element may contain both children elements and text.

```

local xml = require('luaxml-mod-xml')
local handler = require('luaxml-mod-handler')
local x = '<p>hello <a href="http://world.com/">world</a>, how are you?</p>'
local domHandler = handler.domHandler()
local parser = xml.xmlParser(domHandler)
parser:parse(x)
traverseDom(domHandler.root)

```

The ROOT element is stored in `domHandler.root` table, it's child nodes are stored in `_children` tables. Node type is saved in `_type` field, if the node type is ELEMENT, then `_name` field contains element name, `_attr` table contains element attributes. TEXT node contains text content in `_text` field.

The previous code produces following output in the terminal:

```

ROOT : unnamed
ELEMENT : p
TEXT : hello
ELEMENT : a
    href=http://world.com/
TEXT : world
TEXT : , how are you?

```

# Part I

## Original LuaXML documentation by Paul Chakravarti

This document was created automatically from the original source code comments using Pandoc<sup>4</sup>

## 6 Overview

This module provides a non-validating XML stream parser in Lua.

## 7 Features

- Tokenises well-formed XML (relatively robustly)
- Flexible handler based event api (see below)
- Parses all XML Infoset elements - ie.
  - Tags
  - Text
  - Comments
  - CDATA
  - XML Decl
  - Processing Instructions
  - DOCTYPE declarations
- Provides limited well-formedness checking (checks for basic syntax & balanced tags only)
- Flexible whitespace handling (selectable)
- Entity Handling (selectable)

## 8 Limitations

- Non-validating
- No charset handling
- No namespace support
- Shallow well-formedness checking only (fails to detect most semantic errors)

---

<sup>4</sup><http://johnmacfarlane.net/pandoc/>

## 9 API

The parser provides a partially object-oriented API with functionality split into tokeniser and hanlder components.

The handler instance is passed to the tokeniser and receives callbacks for each XML element processed (if a suitable handler function is defined). The API is conceptually similar to the SAX API but implemented differently.

The following events are generated by the tokeniser

```
handler:starttag      - Start Tag
handler:endtag        - End Tag
handler:text          - Text
handler:decl          - XML Declaration
handler:pi            - Processing Instruction
handler:comment       - Comment
handler:dtd           - DOCTYPE definition
handler:cdata         - CDATA
```

The function prototype for all the callback functions is

```
callback(val, attrs, start, end)
```

where attrs is a table and val/attrs are overloaded for specific callbacks - ie.

Callback	val	attrs (table)
starttag	name	{ attributes (name=val).. }
endtag	name	nil
text	<text>	nil
cdata	<text>	nil
decl	"xml"	{ attributes (name=val).. }
pi	pi name	{ attributes (if present).. _text = <PI Text> }
comment	<text>	nil
dtd	root element	{ _root = <Root Element>, _type = SYSTEM PUBLIC, _name = <name>, _uri = <uri>, _internal = <internal dtd> }

(starttag & endtag provide the character positions of the start/end of the element)

XML data is passed to the parser instance through the ‘parse’ method (Note: must be passed as single string currently)

## 10 Options

Parser options are controlled through the ‘self.options’ table. Available options are -

- stripWS

Strip non-significant whitespace (leading/trailing) and do not generate events for empty text elements

- expandEntities

Expand entities (standard entities + single char numeric entities only currently - could be extended at runtime if suitable DTD parser added elements to table (see obj.\_ENTITIES). May also be possible to expand multibyte entities for UTF-8 only

- errorHandler

Custom error handler function

NOTE: Boolean options must be set to ‘nil’ not ‘0’

## 11 Usage

Create a handler instance -

```
h = { starttag = function(t,a,s,e) .... end,
      endtag = function(t,a,s,e) .... end,
      text = function(t,a,s,e) .... end,
      cdata = text }
```

(or use predefined handler - see luaxml-mod-handler.lua)

Create parser instance -

```
p = xmlParser(h)
```

Set options -

```
p.options.xxxx = nil
```

Parse XML data -

```
xmlParser:parse("<?xml... ")
```

## 12 Handlers

### 12.1 Overview

Standard XML event handler(s) for XML parser module (luaxml-mod-xml.lua)

## 12.2 Features

```
printHandler      - Generate XML event trace
domHandler       - Generate DOM-like node tree
simpleTreeHandler - Generate 'simple' node tree
simpleTeXhandler - SAX like handler with support for CSS selectros
```

## 12.3 API

Must be called as handler function from xmlParser and implement XML event callbacks (see xmlParser.lua for callback API definition)

### 12.3.1 printHandler

printHandler prints event trace for debugging

### 12.3.2 domHandler

domHandler generates a DOM-like node tree structure with a single ROOT node parent - each node is a table comprising fields below.

```
node = { _name = <Element Name>,
         _type = ROOT|ELEMENT|TEXT|COMMENT|PI|DECL|DTD,
         _attr = { Node attributes - see callback API },
         _parent = <Parent Node>
         _children = { List of child nodes - ROOT/NODE only }
     }
```

### 12.3.3 simpleTreeHandler

simpleTreeHandler is a simplified handler which attempts to generate a more ‘natural’ table based structure which supports many common XML formats.

The XML tree structure is mapped directly into a recursive table structure with node names as keys and child elements as either a table of values or directly as a string value for text. Where there is only a single child element this is inserted as a named key - if there are multiple elements these are inserted as a vector (in some cases it may be preferable to always insert elements as a vector which can be specified on a per element basis in the options). Attributes are inserted as a child element with a key of ‘\_attr’.

Only Tag/Text & CDATA elements are processed - all others are ignored.

This format has some limitations - primarily

- Mixed-Content behaves unpredictably - the relationship between text elements and embedded tags is lost and multiple levels of mixed content does not work
- If a leaf element has both a text element and attributes then the text must be accessed through a vector (to provide a container for the attribute)

In general however this format is relatively useful.

## 12.4 Options

```
simpleTreeHandler.options.noReduce = { <tag> = bool,.. }

- Nodes not to reduce children vector even if only
  one child

domHandler.options.(comment|pi|dtd|decl)Node = bool

- Include/exclude given node types
```

## 12.5 Usage

Pased as delegate in xmlParser constructor and called as callback by xmlParser:parse(xml) method.

## 13 History

This library is fork of LuaXML library originally created by Paul Chakravarti. Some files not needed for use with luatex were droped from the distribution. Documentation was converted from original comments in the source code.

## 14 License

This code is freely distributable under the terms of the Lua license (<http://www.lua.org/copyright.html>)