The LuaXML library

Paul Chakravarti Michal Hoftich

August 1, 2012

Contents

Introduction

`LuaXML` is pure lua library for reading and serializing of the `xml` files. Current release is aimed mainly as support for the odsfile package. In first release it was included with the odsfile package, but as it is general library which can be used also with other packages, I decided to distribute it as separate library.

Example of usage:

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

First load the libraries. In `luaxml-mod-xml`, there is xml parser and also serializer. In `luaxml-mod-handler`, there are various handlers for dealing with xml data. Handlers are objects with callback functions which are invoked for every type of content in the `xml` file. More information about handlers can be found in the original documentation, section 7.

```
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])

-- 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.
```

Note that `simpleTreeHandler` creates tables that can be easily accessed using standard lua functions, but in case of mixed content, like

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

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

Because at the moment it is intended mainly as support for the odsfile package, there is little documentation, what follows is the original documentation of `LuaXML`, which may be little bit obsolete now.

Original documentation

This document was created automatically from original source code comments using Pandoc[1]

1 Overview

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

2 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)

3 Limitations

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

---

[1] `http://johnmacfarlane.net/pandoc/`

4 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:start       - Start Tag
handler:end         - 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)
start	name	`{ attributes (name=val).. }`
end	name	nil
text	`<text>`	nil
cdata	`<text>`	nil
decl	”xml”	`{ attributes (name=val).. }`
pi	pi name	

```
{ attributes (if present)..
  _text = <PI Text>
}
```

Callback	val	attrs (table)
comment	`<text>`	nil
dtd	root element	

```
{ _root = <Root Element>,
  _type = SYSTEM|PUBLIC,
  _name = <name>,
  _uri = <uri>,
  _internal = <internal dtd>
}
```

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

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

5 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 multibyre entities for UTF–8 only

- errorHandler

  Custom error handler function

NOTE: Boolean options must be set to 'nil' not '0'

6 Usage

Create a handler instance -

```
h = { start = function(t,a,s,e) .... end,
      end = 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... ")
```

7 Handlers

7.1 Overview

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

7.2 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
```

7.3 API

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

7.3.1 printHandler

printHandler prints event trace for debugging

7.3.2 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 }
      }
```

The dom structure is capable of representing any valid XML document

7.3.3 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.

7.4 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
```

7.5 Usage

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

8 History

This library is fork of LuaXML library originaly created by Paul Chakravarti. Its original version can be found at `http://manoelcampos.com/files/LuaXML--0.0.0-lua5.1.tgz`. Some files not needed for use with luatex were droped from the distribution. Documentation was converted from original comments in the source code.

9 License

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