The luaextra package

Elie Roux
elie.roux@telecom-bretagne.eu

2009/04/15 v0.91

Abstract

Additional lua functions taken from the libs of ConTEXt. For an introduction on this package (among others), please refer to the document `luatex-reference.pdf`.

1 Overview

Lua is a very minimal language, and it does not have a lot of built-in functions. Some functions will certainly be needed by a lot of packages. Instead of making each of them implement these functions, the aim of this file is to provide a minimal set of functions. All functions are taken from ConTEXt libraries.

There are some differences with the ConTEXt funtions though, especially on names: for example the `file.*` funtions are renamed in `fpath.*`. It seems more logical as they deal with file paths, not files. Also the `file.is_readable` and `file.is_writable` are renamed `lfs.is_readable` and `lfs.is_writable`.

If you use a function you think is missing in this file, please tell the maintainer.

`Warning:` Even if the names will certainly remain the same, some implementations may differ, and some functions might appear or dissapear. As LuaTEX is not stable, this file is not neither.

All functions are described in this document, but the one of the functions you'll use most will certainly be `table.serialize` (also named `table.tostring`) that takes a table and returns an intented string describing the table. It describes the table so that LuaTEX can read it again as a table. You can do a lot of things with this functions, like printing a table for debugging, or saving a table into a file. Functions are also converted into bytecode to be saved.

2 luaextra.lua

```
1 do
2     local luatextra_module = {
3         name          = "luaextra",
4         version       = 0.91,
5         date          = "2009/04/15",
```

```
        description   = "Lua additional functions.",
        author        = "Hans Hagen, PRAGMA-ADE, Hasselt NL & Elie Roux",
        copyright     = "PRAGMA ADE / ConTeXt Development Team",
        license       = "See ConTeXt's mreadme.pdf for the license",
    }

    luatextra.provides_module(luatextra_module)
end
```

`string:stripspaces` A function to strip the spaces at the beginning and at the end of a string.

```

function string:stripspaces()
    return (self:gsub("^%s*(.-)%s*$", "%1"))
end

```

`string.is boolean` If the argument is a string describing a boolean, this function returns the boolean, otherwise it retuns nil.

```

function string.is_boolean(str)
    if type(str) == "string" then
        if str == "true" or str == "yes" or str == "on" or str == "t" then
            return true
        elseif str == "false" or str == "no" or str == "off" or str == "f" then
            return false
        end
    end
    return nil
end

```

`string.is number` Returns true if the argument string is a number.

```

function string.is_number(str)
    return str:find("^[%-%+]?[%d]-%.?[%d+]$") == 1
end

```

`lpeg.space and lpeg.newline` Two small helpers for `lpeg`, that will certainly be widely used: spaces and newlines.

```

lpeg.space    = lpeg.S(" \t\f\v")
lpeg.newline  = lpeg.P("\r\n") + lpeg.P("\r") +lpeg.P("\n")

```

`table.fastcopy` A function copying a table fastly.

```

if not table.fastcopy then do

```

```
43     local type, pairs, getmetatable, setmetatable = type, pairs, getmetatable, setmetatable
44
45     local function fastcopy(old) -- fast one
46         if old then
47             local new = { }
48             for k,v in pairs(old) do
49                 if type(v) == "table" then
50                     new[k] = fastcopy(v) -- was just table.copy
51                 else
52                     new[k] = v
53                 end
54             end
55             local mt = getmetatable(old)
56             if mt then
57                 setmetatable(new,mt)
58             end
59             return new
60         else
61             return { }
62         end
63     end
64
65     table.fastcopy = fastcopy
66
67 end end
68
```

`table.copy` A function copying a table in more cases than fastcopy, for example when a key is a table.

```
69
70 if not table.copy then do
71
72     local type, pairs, getmetatable, setmetatable = type, pairs, getmetatable, setmetatable
73
74     local function copy(t, tables) -- taken from lua wiki, slightly adapted
75         tables = tables or { }
76         local tcopy = {}
77         if not tables[t] then
78             tables[t] = tcopy
79         end
80         for i,v in pairs(t) do -- brrr, what happens with sparse indexed
81             if type(i) == "table" then
82                 if tables[i] then
83                     i = tables[i]
84                 else
85                     i = copy(i, tables)
86                 end
87             end
88             if type(v) ~= "table" then
89                 tcopy[i] = v
```

```
            elseif tables[v] then
                tcopy[i] = tables[v]
            else
                tcopy[i] = copy(v, tables)
            end
        end
        local mt = getmetatable(t)
        if mt then
            setmetatable(tcopy,mt)
        end
        return tcopy
    end

    table.copy = copy

end end

```

`table.serialize` A bunch of functions leading to `table.serialize`.

```

function table.sortedkeys(tab)
    local srt, kind = { }, 0 -- 0=unknown 1=string, 2=number 3=mixed
    for key,_ in pairs(tab) do
        srt[#srt+1] = key
        if kind == 3 then
            -- no further check
        else
            local tkey = type(key)
            if tkey == "string" then
            --  if kind == 2 then kind = 3 else kind = 1 end
                kind = (kind == 2 and 3) or 1
            elseif tkey == "number" then
            --  if kind == 1 then kind = 3 else kind = 2 end
                kind = (kind == 1 and 3) or 2
            else
                kind = 3
            end
        end
    end
    if kind == 0 or kind == 3 then
        table.sort(srt,function(a,b) return (tostring(a) < tostring(b)) end)
    else
        table.sort(srt)
    end
    return srt
end

do
    table.serialize_functions = true
    table.serialize_compact   = true
```

```
138     table.serialize_inline    = true
139
140     local function key(k)
141         if type(k) == "number" then -- or k:find("^%d+$") then
142             return "["..k.."]"
143         elseif noquotes and k:find("^%a[%a%d%_]*$") then
144             return k
145         else
146             return '["'..k..'"]'
147         end
148     end
149
150     local function simple_table(t)
151         if #t > 0 then
152             local n = 0
153             for _,v in pairs(t) do
154                 n = n + 1
155             end
156             if n == #t then
157                 local tt = { }
158                 for i=1,#t do
159                     local v = t[i]
160                     local tv = type(v)
161                     if tv == "number" or tv == "boolean" then
162                         tt[#tt+1] = tostring(v)
163                     elseif tv == "string" then
164                         tt[#tt+1] = ("%q"):format(v)
165                     else
166                         tt = nil
167                         break
168                     end
169                 end
170                 return tt
171             end
172         end
173         return nil
174     end
175
176     local function serialize(root,name,handle,depth,level,reduce,noquotes,indexed)
177         handle = handle or print
178         reduce = reduce or false
179         if depth then
180             depth = depth .. " "
181             if indexed then
182                 handle(("%s{"):format(depth))
183             else
184                 handle(("%s%s={"):format(depth,key(name)))
185             end
186         else
187             depth = ""
```

```
188             local tname = type(name)
189             if tname == "string" then
190                 if name == "return" then
191                     handle("return {")
192                 else
193                     handle(name .. "={")
194                 end
195             elseif tname == "number" then
196                 handle("[" .. name .. "]={")
197             elseif tname == "boolean" then
198                 if name then
199                     handle("return {")
200                 else
201                     handle("{")
202                 end
203             else
204                 handle("t={")
205             end
206         end
207         if root and next(root) then
208             local compact = table.serialize_compact
209             local inline  = compact and table.serialize_inline
210             local first, last = nil, 0 -- #root cannot be trusted here
211             if compact then
212               for k,v in ipairs(root) do -- NOT: for k=1,#root do (why)
213                     if not first then first = k end
214                     last = last + 1
215                 end
216             end
217             for _,k in pairs(table.sortedkeys(root)) do
218                 local v = root[k]
219                 local t = type(v)
220                 if compact and first and type(k) == "number" and k >= first and k <= last th
221                     if t == "number" then
222                         handle(("%s %s,"):format(depth,v))
223                     elseif t == "string" then
224                         if reduce and (v:find("^[%-%+]?[%d]-%.?[%d+]$") == 1) then
225                             handle(("%s %s,"):format(depth,v))
226                         else
227                             handle(("%s %q,"):format(depth,v))
228                         end
229                     elseif t == "table" then
230                         if not next(v) then
231                             handle(("%s {},"):format(depth))
232                         elseif inline then
233                             local st = simple_table(v)
234                             if st then
235                                 handle(("%s { %s },"):format(depth,table.concat(st,", ")))
236                             else
237                                 serialize(v,k,handle,depth,level+1,reduce,noquotes,true)
```

```
                        end
                    else
                        serialize(v,k,handle,depth,level+1,reduce,noquotes,true)
                    end
                elseif t == "boolean" then
                    handle(("%s %s,"):format(depth,tostring(v)))
                elseif t == "function" then
                    if table.serialize_functions then
                        handle(('%s loadstring(%q),'):format(depth,string.dump(v)))
                    else
                        handle(('%s "function",'):format(depth))
                    end
                else
                    handle(("%s %q,"):format(depth,tostring(v)))
                end
            elseif k == "__p__" then -- parent
                if false then
                    handle(("%s __p__=nil,"):format(depth))
                end
            elseif t == "number" then
                handle(("%s %s=%s,"):format(depth,key(k),v))
            elseif t == "string" then
                if reduce and (v:find("^[%-%+]?[%d]-%.?[%d+]$") == 1) then
                    handle(("%s %s=%s,"):format(depth,key(k),v))
                else
                    handle(("%s %s=%q,"):format(depth,key(k),v))
                end
            elseif t == "table" then
                if not next(v) then
                    handle(("%s %s={},"):format(depth,key(k)))
                elseif inline then
                    local st = simple_table(v)
                    if st then
                        handle(("%s %s={ %s },"):format(depth,key(k),table.concat(st,",
                    else
                        serialize(v,k,handle,depth,level+1,reduce,noquotes)
                    end
                else
                    serialize(v,k,handle,depth,level+1,reduce,noquotes)
                end
            elseif t == "boolean" then
                handle(("%s %s=%s,"):format(depth,key(k),tostring(v)))
            elseif t == "function" then
                if table.serialize_functions then
                    handle(('%s %s=loadstring(%q),'):format(depth,key(k),string.dump(v))
                else
                    handle(('%s %s="function",'):format(depth,key(k)))
                end
            else
                handle(("%s %s=%q,"):format(depth,key(k),tostring(v)))
```

```
288                 --  handle(('%s %s=loadstring(%q),'):format(depth,key(k),string.dump(functio
289                 end
290             end
291             if level > 0 then
292                 handle(("%s},"):format(depth))
293             else
294                 handle(("%s}"):format(depth))
295             end
296         else
297             handle(("%s}"):format(depth))
298         end
299     end
300
301     function table.serialize(root,name,reduce,noquotes)
302         local t = { }
303         local function flush(s)
304             t[#t+1] = s
305         end
306         serialize(root, name, flush, nil, 0, reduce, noquotes)
307         return table.concat(t,"\n")
308     end
309
310     function table.tostring(t, name)
311         return table.serialize(t, name)
312     end
313
314     function table.tohandle(handle,root,name,reduce,noquotes)
315         serialize(root, name, handle, nil, 0, reduce, noquotes)
316     end
317
318     -- sometimes tables are real use (zapfino extra pro is some 85M) in which
319     -- case a stepwise serialization is nice; actually, we could consider:
320     --
321     -- for line in table.serializer(root,name,reduce,noquotes) do
322     --     ...(line)
323     -- end
324     --
325     -- so this is on the todo list
326
327     table.tofile_maxtab = 2*1024
328
329     function table.tofile(filename,root,name,reduce,noquotes)
330         local f = io.open(filename,'w')
331         if f then
332             local concat = table.concat
333             local maxtab = table.tofile_maxtab
334             if maxtab > 1 then
335                 local t = { }
336                 local function flush(s)
337                     t[#t+1] = s
```

```
                    if #t > maxtab then
                        f:write(concat(t,"\n"),"\n") -- hm, write(sometable) should be nice
                        t = { }
                    end
                end
                serialize(root, name, flush, nil, 0, reduce, noquotes)
                f:write(concat(t,"\n"),"\n")
            else
                local function flush(s)
                    f:write(s,"\n")
                end
                serialize(root, name, flush, nil, 0, reduce, noquotes)
            end
            f:close()
        end
    end

end

```

`table.tohash` Returning a table with all values of the argument table as keys, and `false` as values. This is what we will call a hash.

```

function table.tohash(t)
    local h = { }
    for _, v in pairs(t) do -- no ipairs here
        h[v] = true
    end
    return h
end

```

`table.fromhash` Returning a table built from a hash, with simple integer keys.

```

function table.fromhash(t)
    local h = { }
    for k, v in pairs(t) do -- no ipairs here
        if v then h[#h+1] = k end
    end
    return h
end

```

`table.contains value` A function returning true if the value `val` is in the table `t`.

```

function table.contains_value(t, val)
    if t then
        for k, v in pairs(t) do
            if v==val then
```

```
                return true
            end
        end
    end
    return false
end

```

`table.contains key` A function returning true if the key `key` is in the table `t`

```

function table.contains_key(t, key)
    if t then
        for k, v in pairs(t) do
            if k==key then
                return true
            end
        end
    end
    return false
end

```

`table.value position` A function returning the position of a value in a table. This will be important to be able to remove a value.

```

function table.value_position(t, val)
    if t then
        local i=1
        for k, v in pairs(t) do
            if v==val then
                return i
            end
            i=i+1
        end
    end
    return 0
end

```

`table.key position` A function returning the position of a key in a table.

```

function table.key_position(t, key)
    if t then
        local i=1
        for k,v in pairs(t) do
            if k==key then
                return i
            end
            i = i+1
```

```
422         end
423     end
424     return -1
425 end
426
```

`table.remove value` Removes the first occurence of a value from a table.

```
427
428 function table.remove_value(t, v)
429     local p = table.value_position(t,v)
430     if p ~= -1 then
431         table.remove(t, table.value_position(t,v))
432     end
433 end
434
```

`table.remove key` Removing a key from a table.

```
435
436 function table.remove_key(t, k)
437     local p = table.key_position(t,k)
438     if p ~= -1 then
439         table.remove(t, table.key_position(t,k))
440     end
441 end
442
```

`table.is empty` Returns true if a table is empty.

```
443
444 function table.is_empty(t)
445     return not t or not next(t)
446 end
447
```

`fpath` will contain all the file path manipulation functions. Some functions certainly need a little update or cleanup...

```
448
449 fpath = { }
450
```

`fpath.removesuffix` A function to remove the suffix (extention) of a filename.

```
451
452 function fpath.removesuffix(filename)
453     return filename:gsub("%.[%a%d]+$", "")
454 end
455
```

`fpath.addsuffix` A function adding a suffix to a filename, except if it already has one.

```
456
```

```
457 function fpath.addsuffix(filename, suffix)
458     if not filename:find("%.[%a%d]+$") then
459         return filename .. "." .. suffix
460     else
461         return filename
462     end
463 end
464
```

`fpath.replacesuffix` A function replacing a suffix by a new one.

```
465
466 function fpath.replacesuffix(filename, suffix)
467     if not filename:find("%.[%a%d]+$") then
468         return filename .. "." .. suffix
469     else
470         return (filename:gsub("%.[%a%d]+$","."..suffix))
471     end
472 end
473
```

`fpath.dirname` A function returning the directory of a file path.

```
474
475 function fpath.dirname(name)
476     return name:match("^(.+)[/\\].-$") or ""
477 end
478
```

`fpath.basename` A function returning the basename (the name of the file, without the directories) of a file path.

```
479
480 function fpath.basename(fname)
481     if not fname then
482         return nil
483     end
484     return fname:match("^.+[/\\](.-)$") or fname
485 end
486
```

`fpath.nameonly` Returning the basename of a file without the suffix.

```
487
488 function fpath.nameonly(name)
489     return ((name:match("^.+[/\\](.-)$") or name):gsub("%..*$",""))
490 end
491
```

`fpath.suffix` Returns the suffix of a file name.

```
492
493 function fpath.suffix(name)
```

```
494     return name:match("^.+%.([^/\\]-)$") or  ""
495 end
496
```

`fpath.join` A function joining any number of arguments into a complete path.

```
497
498 function fpath.join(...)
499     local pth = table.concat({...},"/")
500     pth = pth:gsub("\\","/")
501     local a, b = pth:match("^(.*://)(.*)$")
502     if a and b then
503         return a .. b:gsub("//+","/")
504     end
505     a, b = pth:match("^(//)(.*)$")
506     if a and b then
507         return a .. b:gsub("//+","/")
508     end
509     return (pth:gsub("//+","/"))
510 end
511
```

`fpath.split` A function returning a table with all directories from a filename.

```
512
513 function fpath.split(str)
514     local t = { }
515     str = str:gsub("\\", "/")
516     str = str:gsub("(%a):([;/])", "%1\001%2")
517     for name in str:gmatch("([^;:]+)") do
518         if name ~= "" then
519             name = name:gsub("\001",":")
520             t[#t+1] = name
521         end
522     end
523     return t
524 end
525
```

`fpath.normalize sep` A function to change directory separators to canonical ones (`/`).

```
526
527 function fpath.normalize_sep(str)
528     return str:gsub("\\", "/")
529 end
530
```

`fpath.localize sep` A function changing directory separators into local ones (`/` on Unix, `\` on Windows).

```
531
532 function fpath.localize_sep(str)
```

```
533     if os.type == 'windows' or type == 'msdos' then
534         return str:gsub("/", "\\")
535     else
536         return str:gsub("\\", "/")
537     end
538 end
539
```

`lfs.is writable` Returns true if a file is writable. This function and the following ones are a bit too expensive, they should be made with `lfs.attributes`.

```
540
541 function lfs.is_writable(name)
542     local f = io.open(name, 'w')
543     if f then
544         f:close()
545         return true
546     else
547         return false
548     end
549 end
550
```

`lfs.is readable` Returns true if a file is readable.

```
551
552 function lfs.is_readable(name)
553     local f = io.open(name,'r')
554     if f then
555         f:close()
556         return true
557     else
558         return false
559     end
560 end
561
```

`math.round` Returns the closest integer.

```
562
563 if not math.round then
564     function math.round(x)
565         return math.floor(x + 0.5)
566     end
567 end
568
```

`math.div` Returns the quotient of the euclidian division of n by m.

```
569
570 if not math.div then
571     function math.div(n,m)
```

```
572         return floor(n/m)
573     end
574 end
575
```

`math.mod` Returns the remainder of the euclidian division of n by m.

```
576
577 if not math.mod then
578     function math.mod(n,m)
579         return n % m
580     end
581 end
582
```