

The luaextra package

Elie Roux

elie.roux@telecom-bretagne.eu

2009/04/15 v0.91

Abstract

Additional lua functions taken from the libs of ConT_EXt. 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 ConT_EXt libraries.

There are some differences with the ConT_EXt functions though, especially on names: for example the `file.*` functions 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 disappear. As LuaT_EX 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 intended string describing the table. It describes the table so that LuaT_EX 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",
```

```

6      description = "Lua additional functions.",
7      author      = "Hans Hagen, PRAGMA-ADE, Hasselt NL & Elie Roux",
8      copyright   = "PRAGMA ADE / ConTeXt Development Team",
9      license     = "See ConTeXt's mreadme.pdf for the license",
10   }
11
12   luatextra.provides_module(luatextra_module)
13 end

```

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

```

14
15 function string.stripspaces()
16     return (self:gsub("^%s*(.*)%s*$", "%1"))
17 end
18

```

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

```

19
20 function string.is_boolean(str)
21     if type(str) == "string" then
22         if str == "true" or str == "yes" or str == "on" or str == "t" then
23             return true
24         elseif str == "false" or str == "no" or str == "off" or str == "f" then
25             return false
26         end
27     end
28     return nil
29 end
30

```

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

```

31
32 function string.is_number(str)
33     return str:find("[%-+%]?[%d]-%.?[%d+]"$) == 1
34 end
35

```

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

```

36
37 lpeg.space    = lpeg.S(" \t\f\v")
38 lpeg.newline = lpeg.P("\r\n") + lpeg.P("\r") + lpeg.P("\n")
39

```

table.fastcopy A function copying a table fastly.

```

40
41 if not table.fastcopy then do
42

```

```

43 local type, pairs, getmetatable, setmetatable =
44     type, pairs, getmetatable, setmetatable
45
46 local function fastcopy(old) -- fast one
47     if old then
48         local new = { }
49         for k,v in pairs(old) do
50             if type(v) == "table" then
51                 new[k] = fastcopy(v) -- was just table.copy
52             else
53                 new[k] = v
54             end
55         end
56         local mt = getmetatable(old)
57         if mt then
58             setmetatable(new,mt)
59         end
60         return new
61     else
62         return { }
63     end
64 end
65
66 table.fastcopy = fastcopy
67
68 end end
69

```

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

```

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

```

```

90         tcopy[i] = v
91     elseif tables[v] then
92         tcopy[i] = tables[v]
93     else
94         tcopy[i] = copy(v, tables)
95     end
96 end
97 local mt = getmetatable(t)
98 if mt then
99     setmetatable(tcopy,mt)
100 end
101 return tcopy
102 end
103
104 table.copy = copy
105
106 end end
107

```

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

```

108
109 function table.sortedkeys(tab)
110     local srt, kind = { }, 0 -- 0=unknown 1=string, 2=number 3=mixed
111     for key,_ in pairs(tab) do
112         srt[#srt+1] = key
113         if kind == 3 then
114             -- no further check
115         else
116             local tkey = type(key)
117             if tkey == "string" then
118                 -- if kind == 2 then kind = 3 else kind = 1 end
119                 kind = (kind == 2 and 3) or 1
120             elseif tkey == "number" then
121                 -- if kind == 1 then kind = 3 else kind = 2 end
122                 kind = (kind == 1 and 3) or 2
123             else
124                 kind = 3
125             end
126         end
127     end
128     if kind == 0 or kind == 3 then
129         table.sort(srt,function(a,b) return (tostring(a) < tostring(b)) end)
130     else
131         table.sort(srt)
132     end
133     return srt
134 end
135
136 do
137     table.serialize_functions = true

```

```

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

```

```

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

```

```

238             serialize(v,k,handle,depth,level+1,reduce,noquotes,true)
239         end
240     else
241         serialize(v,k,handle,depth,level+1,reduce,noquotes,true)
242     end
243     elseif t == "boolean" then
244         handle((" %s %s,"):format(depth,tostring(v)))
245     elseif t == "function" then
246         if table.serialize_functions then
247             handle((" %s loadstring(%q),'):format(depth,string.dump(v)))
248         else
249             handle((" %s "function",'):format(depth))
250         end
251     else
252         handle((" %s %q,"):format(depth,tostring(v)))
253     end
254     elseif k == "__p__" then -- parent
255         if false then
256             handle((" %s __p__=nil,"):format(depth))
257         end
258     elseif t == "number" then
259         handle((" %s %s=%s,"):format(depth,key(k),v))
260     elseif t == "string" then
261         if reduce and (v:find("[^-+%]?[%d]-%.?[%d+]"$) == 1) then
262             handle((" %s %s=%s,"):format(depth,key(k),v))
263         else
264             handle((" %s %s=%q,"):format(depth,key(k),v))
265         end
266     elseif t == "table" then
267         if not next(v) then
268             handle((" %s %s={},"):format(depth,key(k)))
269         elseif inline then
270             local st = simple_table(v)
271             if st then
272                 handle((" %s %s={ %s },"):format(depth,key(k),table.concat(st," "))
273             else
274                 serialize(v,k,handle,depth,level+1,reduce,noquotes)
275             end
276         else
277             serialize(v,k,handle,depth,level+1,reduce,noquotes)
278         end
279     elseif t == "boolean" then
280         handle((" %s %s=%s,"):format(depth,key(k),tostring(v)))
281     elseif t == "function" then
282         if table.serialize_functions then
283             handle((" %s %s=loadstring(%q),'):format(depth,key(k),string.dump(v))
284         else
285             handle((" %s %s="function",'):format(depth,key(k))
286         end
287     else

```

```

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

```



```

338             t[#t+1] = s
339             if #t > maxtab then
340                 f:write(concat(t,"\n"),"\n") -- hm, write(sometable) should be nice
341                 t = { }
342             end
343         end
344         serialize(root, name, flush, nil, 0, reduce, noquotes)
345         f:write(concat(t,"\n"),"\n")
346     else
347         local function flush(s)
348             f:write(s,"\n")
349         end
350         serialize(root, name, flush, nil, 0, reduce, noquotes)
351     end
352     f:close()
353 end
354 end
355
356 end
357

```

`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.

```

358
359 function table.tohash(t)
360     local h = { }
361     for _, v in pairs(t) do -- no ipairs here
362         h[v] = true
363     end
364     return h
365 end
366

```

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

```

367
368 function table.fromhash(t)
369     local h = { }
370     for k, v in pairs(t) do -- no ipairs here
371         if v then h[#h+1] = k end
372     end
373     return h
374 end
375

```

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

```

376
377 function table.contains_value(t, val)
378     if t then
379         for k, v in pairs(t) do

```

```

380         if v==val then
381             return true
382         end
383     end
384 end
385 return false
386 end
387

```

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

```

388
389 function table.contains_key(t, key)
390     if t then
391         for k, v in pairs(t) do
392             if k==key then
393                 return true
394             end
395         end
396     end
397     return false
398 end
399

```

`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.

```

400
401 function table.value_position(t, val)
402     if t then
403         local i=1
404         for k, v in pairs(t) do
405             if v==val then
406                 return i
407             end
408             i=i+1
409         end
410     end
411     return 0
412 end
413

```

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

```

414
415 function table.key_position(t, key)
416     if t then
417         local i=1
418         for k,v in pairs(t) do
419             if k==key then
420                 return i
421             end

```

```

422         i = i+1
423     end
424 end
425 return -1
426 end
427

```

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

```

428
429 function table.remove_value(t, v)
430     local p = table.value_position(t,v)
431     if p ~= -1 then
432         table.remove(t, table.value_position(t,v))
433     end
434 end
435

```

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

```

436
437 function table.remove_key(t, k)
438     local p = table.key_position(t,k)
439     if p ~= -1 then
440         table.remove(t, table.key_position(t,k))
441     end
442 end
443

```

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

```

444
445 function table.is_empty(t)
446     return not t or not next(t)
447 end
448

```

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

```

449
450 fpath = { }
451

```

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

```

452
453 function fpath.removesuffix(filename)
454     return filename:gsub("%.[%a%d]+$", "")
455 end
456

```

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

```
457
458 function fpath.addsuffix(filename, suffix)
459     if not filename:find("%. [%a%d]+$") then
460         return filename .. "." .. suffix
461     else
462         return filename
463     end
464 end
465
```

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

```
466
467 function fpath.replacesuffix(filename, suffix)
468     if not filename:find("%. [%a%d]+$") then
469         return filename .. "." .. suffix
470     else
471         return (filename:gsub("%. [%a%d]+$", "." .. suffix))
472     end
473 end
474
```

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

```
475
476 function fpath.dirname(name)
477     return name:match("^(.+)[/\\].-$") or ""
478 end
479
```

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

```
480
481 function fpath.basename(fname)
482     if not fname then
483         return nil
484     end
485     return fname:match("^.+[/\\](.-)$") or fname
486 end
487
```

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

```
488
489 function fpath.nameonly(name)
490     return ((name:match("^.+[/\\](.-)$") or name):gsub("%. *$", ""))
491 end
492
```

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

```
493
494 function fpath.suffix(name)
495     return name:match("^%.([\^\]\-)$") or ""
496 end
497
```

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

```
498
499 function fpath.join(...)
500     local pth = table.concat({...},"/")
501     pth = pth:gsub("\\","/")
502     local a, b = pth:match("^(*://)(.*)$")
503     if a and b then
504         return a .. b:gsub("//+","/")
505     end
506     a, b = pth:match("^(/)(.*)$")
507     if a and b then
508         return a .. b:gsub("//+","/")
509     end
510     return (pth:gsub("//+","/"))
511 end
512
```

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

```
513
514 function fpath.split(str)
515     local t = { }
516     str = str:gsub("\\", "/")
517     str = str:gsub("(%a):([;/])", "%1\001%2")
518     for name in str:gmatch("([^;:]+)") do
519         if name ~= "" then
520             name = name:gsub("\001",":")
521             t[#t+1] = name
522         end
523     end
524     return t
525 end
526
```

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

```
527
528 function fpath.normalize_sep(str)
529     return str:gsub("\\", "/")
530 end
531
```

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

```

532
533 function fpath.localize_sep(str)
534     if os.type == 'windows' or type == 'msdos' then
535         return str:gsub("/", "\\")
536     else
537         return str:gsub("\\", "/")
538     end
539 end
540

```

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`.

```

541
542 function lfs.is_writable(name)
543     local f = io.open(name, 'w')
544     if f then
545         f:close()
546         return true
547     else
548         return false
549     end
550 end
551

```

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

```

552
553 function lfs.is_readable(name)
554     local f = io.open(name, 'r')
555     if f then
556         f:close()
557         return true
558     else
559         return false
560     end
561 end
562

```

math.round Returns the closest integer.

```

563
564 if not math.round then
565     function math.round(x)
566         return math.floor(x + 0.5)
567     end
568 end
569

```

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

```

570

```

```
571 if not math.div then
572     function math.div(n,m)
573         return floor(n/m)
574     end
575 end
576
```

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

```
577
578 if not math.mod then
579     function math.mod(n,m)
580         return n % m
581     end
582 end
583
```