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

```
do
    local luatextra_module = {
        name          = "luaextra",
        version       = 0.91,
        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

```

```
    local type, pairs, getmetatable, setmetatable =
        type, pairs, getmetatable, setmetatable

    local function fastcopy(old) -- fast one
        if old then
            local new = { }
            for k,v in pairs(old) do
                if type(v) == "table" then
                    new[k] = fastcopy(v) -- was just table.copy
                else
                    new[k] = v
                end
            end
            local mt = getmetatable(old)
            if mt then
                setmetatable(new,mt)
            end
            return new
        else
            return { }
        end
    end

    table.fastcopy = fastcopy

end end

```

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

```

if not table.copy then do

    local type, pairs, getmetatable, setmetatable = type, pairs, getmetatable, setmetatable

    local function copy(t, tables) -- taken from lua wiki, slightly adapted
        tables = tables or { }
        local tcopy = {}
        if not tables[t] then
            tables[t] = tcopy
        end
        for i,v in pairs(t) do -- brrr, what happens with sparse indexed
            if type(i) == "table" then
                if tables[i] then
                    i = tables[i]
                else
                    i = copy(i, tables)
                end
            end
            if type(v) ~= "table" then
```

```
                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
table.serialize_inline    = true

local function key(k)
    if type(k) == "number" then -- or k:find("^%d+$") then
        return "["..k.."]"
    elseif noquotes and k:find("^%a[%a%d%_]*$") then
        return k
    else
        return '["'..k..'"]'
    end
end

local function simple_table(t)
    if #t > 0 then
        local n = 0
        for _,v in pairs(t) do
            n = n + 1
        end
        if n == #t then
            local tt = { }
            for i=1,#t do
                local v = t[i]
                local tv = type(v)
                if tv == "number" or tv == "boolean" then
                    tt[#tt+1] = tostring(v)
                elseif tv == "string" then
                    tt[#tt+1] = ("%q"):format(v)
                else
                    tt = nil
                    break
                end
            end
            return tt
        end
    end
    return nil
end

local function serialize(root,name,handle,depth,level,reduce,noquotes,indexed)
    handle = handle or print
    reduce = reduce or false
    if depth then
        depth = depth .. " "
        if indexed then
            handle(("%s{"):format(depth))
        else
            handle(("%s%s={"):format(depth,key(name)))
        end
    else
```

```
        depth = ""
        local tname = type(name)
        if tname == "string" then
            if name == "return" then
                handle("return {")
            else
                handle(name .. "={")
            end
        elseif tname == "number" then
            handle("[" .. name .. "]={")
        elseif tname == "boolean" then
            if name then
                handle("return {")
            else
                handle("{")
            end
        else
            handle("t={")
        end
    end
    if root and next(root) then
        local compact = table.serialize_compact
        local inline  = compact and table.serialize_inline
        local first, last = nil, 0 -- #root cannot be trusted here
        if compact then
          for k,v in ipairs(root) do -- NOT: for k=1,#root do (why)
                if not first then first = k end
                last = last + 1
            end
        end
        for _,k in pairs(table.sortedkeys(root)) do
            local v = root[k]
            local t = type(v)
            if compact and first and type(k) == "number" and k >= first and k <= last th
                if t == "number" then
                    handle(("%s %s,"):format(depth,v))
                elseif t == "string" then
                    if reduce and (v:find("^[%-%+]?[%d]-%.?[%d+]$") == 1) then
                        handle(("%s %s,"):format(depth,v))
                    else
                        handle(("%s %q,"):format(depth,v))
                    end
                elseif t == "table" then
                    if not next(v) then
                        handle(("%s {},"):format(depth))
                    elseif inline then
                        local st = simple_table(v)
                        if st then
                            handle(("%s { %s },"):format(depth,table.concat(st,", ")))
                        else
```

```
                                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)))
            --  handle(('%s %s=loadstring(%q),'):format(depth,key(k),string.dump(functi
            end
        end
        if level > 0 then
            handle(("%s},"):format(depth))
        else
            handle(("%s}"):format(depth))
        end
    else
        handle(("%s}"):format(depth))
    end
end

function table.serialize(root,name,reduce,noquotes)
    local t = { }
    local function flush(s)
        t[#t+1] = s
    end
    serialize(root, name, flush, nil, 0, reduce, noquotes)
    return table.concat(t,"\n")
end

function table.tostring(t, name)
    return table.serialize(t, name)
end

function table.tohandle(handle,root,name,reduce,noquotes)
    serialize(root, name, handle, nil, 0, reduce, noquotes)
end

-- sometimes tables are real use (zapfino extra pro is some 85M) in which
-- case a stepwise serialization is nice; actually, we could consider:
--
-- for line in table.serializer(root,name,reduce,noquotes) do
--    ...(line)
-- end
--
-- so this is on the todo list

table.tofile_maxtab = 2*1024

function table.tofile(filename,root,name,reduce,noquotes)
    local f = io.open(filename,'w')
    if f then
        local concat = table.concat
        local maxtab = table.tofile_maxtab
        if maxtab > 1 then
            local t = { }
            local function flush(s)
```

```
                    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
        end
    end
    return -1
end

```

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

```

function table.remove_value(t, v)
    local p = table.value_position(t,v)
    if p ~= -1 then
        table.remove(t, table.value_position(t,v))
    end
end

```

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

```

function table.remove_key(t, k)
    local p = table.key_position(t,k)
    if p ~= -1 then
        table.remove(t, table.key_position(t,k))
    end
end

```

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

```

function table.is_empty(t)
    return not t or not next(t)
end

```

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

```

fpath = { }

```

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

```

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

```

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

```

function fpath.addsuffix(filename, suffix)
    if not filename:find("%.[%a%d]+$") then
        return filename .. "." .. suffix
    else
        return filename
    end
end

```

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

```

function fpath.replacesuffix(filename, suffix)
    if not filename:find("%.[%a%d]+$") then
        return filename .. "." .. suffix
    else
        return (filename:gsub("%.[%a%d]+$","."..suffix))
    end
end

```

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

```

function fpath.dirname(name)
    return name:match("^(.+)[/\\].-$") or ""
end

```

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

```

function fpath.basename(fname)
    if not fname then
        return nil
    end
    return fname:match("^.+[/\\](.-)$") or fname
end

```

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

```

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

```

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

```

function fpath.suffix(name)
    return name:match("^.+%.([^/\\]-)$") or  ""
end

```

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

```

function fpath.join(...)
    local pth = table.concat({...},"/")
    pth = pth:gsub("\\","/")
    local a, b = pth:match("^(.*://)(.*)$")
    if a and b then
        return a .. b:gsub("//+","/")
    end
    a, b = pth:match("^(//)(.*)$")
    if a and b then
        return a .. b:gsub("//+","/")
    end
    return (pth:gsub("//+","/"))
end

```

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

```

function fpath.split(str)
    local t = { }
    str = str:gsub("\\", "/")
    str = str:gsub("(%a):([;/])", "%1\001%2")
    for name in str:gmatch("([^;:]+)") do
        if name ~= "" then
            name = name:gsub("\001",":")
            t[#t+1] = name
        end
    end
    return t
end

```

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

```

function fpath.normalize_sep(str)
    return str:gsub("\\", "/")
end

```

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

```

function fpath.localize_sep(str)
    if os.type == 'windows' or type == 'msdos' then
        return str:gsub("/", "\\")
    else
        return str:gsub("\\", "/")
    end
end

```

`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`.

```

function lfs.is_writable(name)
    local f = io.open(name, 'w')
    if f then
        f:close()
        return true
    else
        return false
    end
end

```

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

```

function lfs.is_readable(name)
    local f = io.open(name,'r')
    if f then
        f:close()
        return true
    else
        return false
    end
end

```

`math.round` Returns the closest integer.

```

if not math.round then
    function math.round(x)
        return math.floor(x + 0.5)
    end
end

```

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

```

```

```
if not math.div then
    function math.div(n,m)
        return floor(n/m)
    end
end

```

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

```

if not math.mod then
    function math.mod(n,m)
        return n % m
    end
end

```