local n, v = "serpent", 0.28 -- (C) 2012-15 Paul Kulchenko; MIT License

local c, d = "Paul Kulchenko", "Lua serializer and pretty printer"

local snum = {[tostring(1/0)]='1/0 --[[math.huge]]',[tostring(-1/0)]='-1/0 --[[-math.huge]]',[tostring(0/0)]='0/0'}

local badtype = {thread = true, userdata = true, cdata = true}

local keyword, globals, G = {}, {}, (\_G or \_ENV)

for \_,k in ipairs({'and', 'break', 'do', 'else', 'elseif', 'end', 'false',

'for', 'function', 'goto', 'if', 'in', 'local', 'nil', 'not', 'or', 'repeat',

'return', 'then', 'true', 'until', 'while'}) do keyword[k] = true end

for k,v in pairs(G) do globals[v] = k end -- build func to name mapping

for \_,g in ipairs({'coroutine', 'debug', 'io', 'math', 'string', 'table', 'os'}) do

for k,v in pairs(G[g] or {}) do globals[v] = g..'.'..k end end

local function s(t, opts)

local name, indent, fatal, maxnum = opts.name, opts.indent, opts.fatal, opts.maxnum

local sparse, custom, huge = opts.sparse, opts.custom, not opts.nohuge

local space, maxl = (opts.compact and '' or ' '), (opts.maxlevel or math.huge)

local iname, comm = '\_'..(name or ''), opts.comment and (tonumber(opts.comment) or math.huge)

local seen, sref, syms, symn = {}, {'local '..iname..'={}'}, {}, 0

local function gensym(val) return '\_'..(tostring(tostring(val)):gsub("[^%w]",""):gsub("(%d%w+)",

-- tostring(val) is needed because \_\_tostring may return a non-string value

function(s) if not syms[s] then symn = symn+1; syms[s] = symn end return tostring(syms[s]) end)) end

local function safestr(s) return type(s) == "number" and tostring(huge and snum[tostring(s)] or s)

or type(s) ~= "string" and tostring(s) -- escape NEWLINE/010 and EOF/026

or ("%q"):format(s):gsub("\010","n"):gsub("\026","\\026") end

local function comment(s,l) return comm and (l or 0) < comm and ' --[['..tostring(s)..']]' or '' end

local function globerr(s,l) return globals[s] and globals[s]..comment(s,l) or not fatal

and safestr(select(2, pcall(tostring, s))) or error("Can't serialize "..tostring(s)) end

local function safename(path, name) -- generates foo.bar, foo[3], or foo['b a r']

local n = name == nil and '' or name

local plain = type(n) == "string" and n:match("^[%l%u\_][%w\_]\*$") and not keyword[n]

local safe = plain and n or '['..safestr(n)..']'

return (path or '')..(plain and path and '.' or '')..safe, safe end

local alphanumsort = type(opts.sortkeys) == 'function' and opts.sortkeys or function(k, o, n) -- k=keys, o=originaltable, n=padding

local maxn, to = tonumber(n) or 12, {number = 'a', string = 'b'}

local function padnum(d) return ("%0"..tostring(maxn).."d"):format(tonumber(d)) end

table.sort(k, function(a,b)

-- sort numeric keys first: k[key] is not nil for numerical keys

return (k[a] ~= nil and 0 or to[type(a)] or 'z')..(tostring(a):gsub("%d+",padnum))

< (k[b] ~= nil and 0 or to[type(b)] or 'z')..(tostring(b):gsub("%d+",padnum)) end) end

local function val2str(t, name, indent, insref, path, plainindex, level)

local ttype, level, mt = type(t), (level or 0), getmetatable(t)

local spath, sname = safename(path, name)

local tag = plainindex and

((type(name) == "number") and '' or name..space..'='..space) or

(name ~= nil and sname..space..'='..space or '')

if seen[t] then -- already seen this element

sref[#sref+1] = spath..space..'='..space..seen[t]

return tag..'nil'..comment('ref', level) end

if type(mt) == 'table' and (mt.\_\_serialize or mt.\_\_tostring) then -- knows how to serialize itself

seen[t] = insref or spath

if mt.\_\_serialize then t = mt.\_\_serialize(t) else t = tostring(t) end

ttype = type(t) end -- new value falls through to be serialized

if ttype == "table" then

if level >= maxl then return tag..'{}'..comment('max', level) end

seen[t] = insref or spath

if next(t) == nil then return tag..'{}'..comment(t, level) end -- table empty

local maxn, o, out = math.min(#t, maxnum or #t), {}, {}

for key = 1, maxn do o[key] = key end

if not maxnum or #o < maxnum then

local n = #o -- n = n + 1; o[n] is much faster than o[#o+1] on large tables

for key in pairs(t) do if o[key] ~= key then n = n + 1; o[n] = key end end end

if maxnum and #o > maxnum then o[maxnum+1] = nil end

if opts.sortkeys and #o > maxn then alphanumsort(o, t, opts.sortkeys) end

local sparse = sparse and #o > maxn -- disable sparsness if only numeric keys (shorter output)

for n, key in ipairs(o) do

local value, ktype, plainindex = t[key], type(key), n <= maxn and not sparse

if opts.valignore and opts.valignore[value] -- skip ignored values; do nothing

or opts.keyallow and not opts.keyallow[key]

or opts.valtypeignore and opts.valtypeignore[type(value)] -- skipping ignored value types

or sparse and value == nil then -- skipping nils; do nothing

elseif ktype == 'table' or ktype == 'function' or badtype[ktype] then

if not seen[key] and not globals[key] then

sref[#sref+1] = 'placeholder'

local sname = safename(iname, gensym(key)) -- iname is table for local variables

sref[#sref] = val2str(key,sname,indent,sname,iname,true) end

sref[#sref+1] = 'placeholder'

local path = seen[t]..'['..tostring(seen[key] or globals[key] or gensym(key))..']'

sref[#sref] = path..space..'='..space..tostring(seen[value] or val2str(value,nil,indent,path))

else

out[#out+1] = val2str(value,key,indent,insref,seen[t],plainindex,level+1)

end

end

local prefix = string.rep(indent or '', level)

local head = indent and '{\n'..prefix..indent or '{'

local body = table.concat(out, ','..(indent and '\n'..prefix..indent or space))

local tail = indent and "\n"..prefix..'}' or '}'

return (custom and custom(tag,head,body,tail) or tag..head..body..tail)..comment(t, level)

elseif badtype[ttype] then

seen[t] = insref or spath

return tag..globerr(t, level)

elseif ttype == 'function' then

seen[t] = insref or spath

local ok, res = pcall(string.dump, t)

local func = ok and ((opts.nocode and "function() --[[..skipped..]] end" or

"((loadstring or load)("..safestr(res)..",'@serialized'))")..comment(t, level))

return tag..(func or globerr(t, level))

else return tag..safestr(t) end -- handle all other types

end

local sepr = indent and "\n" or ";"..space

local body = val2str(t, name, indent) -- this call also populates sref

local tail = #sref>1 and table.concat(sref, sepr)..sepr or ''

local warn = opts.comment and #sref>1 and space.."--[[incomplete output with shared/self-references skipped]]" or ''

return not name and body..warn or "do local "..body..sepr..tail.."return "..name..sepr.."end"

end

local function deserialize(data, opts)

local env = (opts and opts.safe == false) and G

or setmetatable({}, {

\_\_index = function(t,k) return t end,

\_\_call = function(t,...) error("cannot call functions") end

})

local f, res = (loadstring or load)('return '..data, nil, nil, env)

if not f then f, res = (loadstring or load)(data, nil, nil, env) end

if not f then return f, res end

if setfenv then setfenv(f, env) end

return pcall(f)

end

local function merge(a, b) if b then for k,v in pairs(b) do a[k] = v end end; return a; end

return { \_NAME = n, \_COPYRIGHT = c, \_DESCRIPTION = d, \_VERSION = v, serialize = s,

load = deserialize,

dump = function(a, opts) return s(a, merge({name = '\_', compact = true, sparse = true}, opts)) end,

line = function(a, opts) return s(a, merge({sortkeys = true, comment = true}, opts)) end,

block = function(a, opts) return s(a, merge({indent = ' ', sortkeys = true, comment = true}, opts)) end }