The lualibs package

Élie Roux · elie.roux@telecom-bretagne.eu
Philipp Gesang · phg@phi-gamma.net

2016/04/06 v2.4

Abstract

Additional Lua functions taken from the `l-*` and `util-*` files of ConTEXt. For an introduction on this package (among others), please refer to the document `lualatex-doc.pdf`.

Contents

I Package Description

1 Overview

Lua is a very minimal language, and it does only have a minimal standard library. The aim of this package is to provide an extended standard library, to be used by various LuaTEX packages. The code is specific to LuaTEX and depends on LuaTEX functions and modules not available in regular lua.
The code is derived from ConTEXt libraries.

2 Usage

You can either load the **lualibs** module, which will in turn load one of two sets of libraries provided by this package: `require("lualibs")`, or explicitly load the modules you need, e.g.: `require("lualibs-table")`, please note that some modules depend on others.

2.1 *Loading Library Collections*

The libraries are split into a **basic** and an **extended** collection. Though neither initialization time nor memory consumption will be noticably impacted,[1] the **lualibs** package can skip loading of the latter on demand. The ***config*** table needs to be present prior to loading the package for this to work (in the future this may be achieved by an option of ***usepackage**) for LaTeX users). In the `lualibs` field, set `load_extended` to false:

```
\directlua{
  --- 1) create config table safely
  config                        = config or { }
  config.lualibs                = config.lualibs or { }

  --- 2) unset the load_extended entry
  config.lualibs.load_extended  = false

  --- 3) load the lualibs
  require "lualibs"
}
```

However, there is no guarantee that the **extended** set remains permanently excluded. Re-loading the package at a later point will cancel this option and possibly add the missing libraries.

2.2 *Loading libraries Individually*

In order to load the `table` module you would actually have to prepare it as follows:

```
require"lualibs-lua"
require"lualibs-lpeg"
require"lualibs-table"
```

If your code is run by the `texlua` intepreter, you will need to initialize ***kpse*** library so that `require()` can find files under TEXMF tree: `kpse.set_program_name("luatex")`.

[1] Note that in terms of code this is only a small fraction of what ConTeXt loads with every run.

Table 1: The basic set.

lualibs name	ConTEXt name	primary purpose
lualibs-lua.lua	l-lua.lua	compatibility
lualibs-package.lua	l-package.lua	Lua file loaders
lualibs-lpeg.lua	l-lpeg.lua	patterns
lualibs-function.lua	l-function.lua	defines a dummy function
lualibs-string.lua	l-string.lua	string manipulation
lualibs-table.lua	l-table.lua	serialization, conversion
lualibs-boolean.lua	l-boolean.lua	boolean converter
lualibs-number.lua	l-number.lua	bit operations
lualibs-math.lua	l-math.lua	math functions
lualibs-io.lua	l-io.lua	reading and writing files
lualibs-os.lua	l-os.lua	platform specific code
lualibs-file.lua	l-file.lua	filesystem operations
lualibs-gzip.lua	l-gzip.lua	wrapper for lgzip
lualibs-md5.lua	l-md5.lua	checksum functions
lualibs-dir.lua	l-dir.lua	directory handling
lualibs-unicode.lua	l-unicode.lua	utf and unicode
lualibs-url.lua	l-url.lua	url handling
lualibs-set.lua	l-set.lua	sets

3 Files

The lualibs bundle contains files from two ConTEXt Lua library categories: The generic auxiliary functions (original file prefix: `l-`) together form something close to a standard libary. Most of these are extensions of an existing namespace, like for instance `l-table.lua` which adds full-fledged serialization capabilities to the Lua table library. They were imported under the lualibs-prefix and are contained in the basic collection. (For a list see table 1.)

The extended category comprises a selection of files mostly from the utilities namespace (`util-`; cf. table 2). Their purpose is more specific and at times quite low-level. Additionally, the file `trac-inf.lua` has been included because it is essential to some of the code loaded subsequently.

4 Packaging

By default, lualibs will not load the libraries individually. Instead, it includes two *merged packages* that have been compiled from the original files. This is achieved by means of mtx-package, a script for bundling Lua code shipped with ConTEXt. This concatenates the code of several Lua files into a single file that is both easier to distribute and loading marginally faster. mtx-package ensures that the code from each file gets its own closure and strips newlines and comments, resulting in a smaller payload. Another package that relies on it heavily is the font loader as contained in luaotfload and luatex-fonts. Luaot-

Table 2: The extended set.

lualibs name	ConTEXt name	primary purpose
lualibs-util-str.lua	util-str.lua	extra `string` functions
lualibs-util-fil.lua	util-fil.lua	extra `file` functions
lualibs-util-tab.lua	util-tab.lua	extra `table` functions
lualibs-util-sto.lua	util-sto.lua	table allocation
lualibs-util-prs.lua	util-sto.lua	miscellaneous parsers
lualibs-util-dim.lua	util-dim.lua	conversion between dimensions
lualibs-trac-inf.lua	trac-inf.lua	timing, statistics
lualibs-util-lua.lua	util-lua.lua	operations on bytecode
lualibs-util-deb.lua	util-deb.lua	extra `debug` functionality
lualibs-util-tpl.lua	util-tpl.lua	templating
lualibs-util-sta.lua	util-sta.lua	stacker (e. g. for PDF)
lualibs-util-jsn.lua	util-jsn.lua	conversion to and from json

fload, a port of the ConTEXt fontloader for Plain and LATEX, has a hard dependency on the functionality provided by the **Lualibs** package. The packages should not be updated independently.

If ConTEXt is installed on the system, the merge files can be created by running:

```
mtxrun --script package --merge lualibs-basic.lua
mtxrun --script package --merge lualibs-extended.lua
```

Of course there is a make target for that:

```
make merge
```

will take care of assembling the packages from the files distributed with **lualibs**.

For this to work, the syntax of the Lua file needs to be well-formed: files that should be merged must be included via a function `loadmodule()`. It doesn't matter if the function actually does something; a dummy will suffice. Also, the argument to `loadmodule()` must be wrapped in parentheses. This rule is quite convenient, actually, since it allows excluding files from the merge while still using `loadmodule()` consistently.

```
...
loadmodule("my-lua-file.lua") -- <= will be merged
loadmodule('my-2nd-file.lua') -- <= will be merged
loadmodule "my-3rd-file.lua"  -- <= will be ignored
...
```

II `lualibs.lua`

```
lualibs = lualibs or { }

lualibs.module_info = {
  name          = "lualibs",
  version       = 2.4,
  date          = "2016-04-06",
  description   = "ConTeXt Lua standard libraries.",
  author        = "Hans Hagen, PRAGMA-ADE, Hasselt NL & Elie Roux & Philipp Gesang",
  copyright     = "PRAGMA ADE / ConTeXt Development Team",
  license       = "See ConTeXt's mreadme.pdf for the license",
}

```

The behavior of the lualibs can be configured to some extent.

- Based on the parameter `lualibs.prefer_merged`, the libraries can be loaded via the included merged packages or the individual files.
- Two classes of libraries are distinguished, mainly because of a similar distinction in ConTEXt, but also to make loading of the less fundamental functionality optional. While the "basic" collection is always loaded, the configuration setting `lualibs.load_extended` triggers inclusion of the extended collection.
- Verbosity can be increased via the `verbose` switch.

```

config           = config or { }
config.lualibs   = config.lualibs or { }

if config.lualibs.prefer_merged ~= nil then
  lualibs.prefer_merged = config.lualibs.prefer_merged
else
  lualibs.prefer_merged = true
end

if config.lualibs.load_extended ~= nil then
  lualibs.load_extended = config.lualibs.load_extended
else
  lualibs.load_extended = true
end

if config.lualibs.verbose ~= nil then
  config.lualibs.verbose = config.lualibs.verbose
else
  config.lualibs.verbose = false
end

```

The lualibs may be loaded in scripts. To account for the different environment, fallbacks for the luatexbase facilities are installed.

```

local dofile          = dofile
```

```
local kpsefind_file   = kpse.find_file
local stringformat    = string.format
local texiowrite_nl   = texio.write_nl

local find_file, error, warn, info
do
  local _error, _warn, _info
  if luatexbase and luatexbase.provides_module then
    _error, _warn, _info = luatexbase.provides_module(lualibs.module_info)
  else
    _error, _warn, _info = texiowrite_nl, texiowrite_nl, texiowrite_nl
  end

  if lualibs.verbose then
    error, warn, info = _error, _warn, _info
  else
    local dummylogger = function ( ) end
    error, warn, info = _error, dummylogger, dummylogger
  end
  lualibs.error, lualibs.warn, lualibs.info = error, warn, info
end

if luatexbase and luatexbase.find_file then
  find_file = luatexbase.find_file
else
  kpse.set_program_name"luatex"
  find_file = kpsefind_file
end

```

The lualibs load a merged package by default. In order to create one of these, the meta file that includes the libraries must satisfy certain assumptions `mtx-package` makes about the coding style. Most important is that the functions that indicates which files to include must go by the name `loadmodule()`. For this reason we define a `loadmodule()` function as a wrapper around `dofile()`.

```

local loadmodule = loadmodule or function (name, t)
  if not t then t = "library" end
  local filepath  = find_file(name, "lua")
  if not filepath or filepath == "" then
    warn(stringformat("Could not locate %s "%s".", t, name))
    return false
  end
  dofile(filepath)
  return true
end

lualibs.loadmodule = loadmodule

```

The separation of the “basic” from the “extended” sets coincides with the split into luat-bas.mkiv and luat-lib.mkiv.

```
80
81 if lualibs.basic_loaded        ~= true
82 or config.lualibs.force_reload == true
83 then
84   loadmodule”lualibs-basic.lua”
85   loadmodule”lualibs-compat.lua” --- restore stuff gone since v1.*
86 end
87
88 if  lualibs.load_extended       == true
89 and lualibs.extended_loaded     ~= true
90 or  config.lualibs.force_reload == true
91 then
92   loadmodule”lualibs-extended.lua”
93 end
94
95 --- This restores the default of loading everything should a package
96 --- have requested otherwise. Will be gone once there is a canonical
97 --- interface for parameterized loading of libraries.
98 config.lualibs.load_extended = true
99
100 -- vim:tw=71:sw=2:ts=2:expandtab
101
```

III `lualibs-basic.lua`

```
1 lualibs                 = lualibs or { }
2 local info              = lualibs.info
3 local loadmodule        = lualibs.loadmodule
4
5 local lualibs_basic_module = {
6   name          = ”lualibs-basic”,
7   version       = 2.4,
8   date          = ”2016-04-06”,
9   description   = ”ConTeXt Lua libraries -- basic collection.”,
10  author        = ”Hans Hagen, PRAGMA-ADE, Hasselt NL & Elie Roux & Philipp Gesang”,
11  copyright     = ”PRAGMA ADE / ConTeXt Development Team”,
12  license       = ”See ConTeXt’s mreadme.pdf for the license”,
13 }
14
15 local loaded = false --- track success of package loading
16
17 if lualibs.prefer_merged then
18   info”Loading merged package for collection “basic”.”
19   loaded = loadmodule(’lualibs-basic-merged.lua’)
20 else
21   info”Ignoring merged packages.”
```

```
  info"Falling back to individual libraries from collection "basic"."
end

```

`mtx-package` expects the files to be included by `loadmodule`. If run on this file, it will create `lualibs-basic-merged.lua` from all the files mentioned in the next block.

```

if loaded == false then
  loadmodule("lualibs-lua.lua")
  loadmodule("lualibs-package.lua")
  loadmodule("lualibs-lpeg.lua")
  loadmodule("lualibs-function.lua")
  loadmodule("lualibs-string.lua")
  loadmodule("lualibs-table.lua")
  loadmodule("lualibs-boolean.lua")
  loadmodule("lualibs-number.lua")
  loadmodule("lualibs-math.lua")
  loadmodule("lualibs-io.lua")
  loadmodule("lualibs-os.lua")
  loadmodule("lualibs-file.lua")
  loadmodule("lualibs-gzip.lua")
  loadmodule("lualibs-md5.lua")
  loadmodule("lualibs-dir.lua")
  loadmodule("lualibs-unicode.lua")
  loadmodule("lualibs-url.lua")
  loadmodule("lualibs-set.lua")
end

lualibs.basic_loaded = true
-- vim:tw=71:sw=2:ts=2:expandtab

```

IV `lualibs-extended.lua`

```
lualibs = lualibs or { }

```

Loading the *extended* set requires a tad more effort, but it's well invested.

Since we only want the functionality, we have to simulate parts of a running ConTEXt environment, above all logging, that some of the more involved libraries cannot be loaded without. Also, one utility file cannot be packaged because it returns a table which would preclude loading of later code. Thus, we remove it from the natural loading chain (it is not critical) and append it at the end.

```

local lualibs_extended_module = {
  name          = "lualibs-extended",
  version       = 2.4,
  date          = "2016-04-06",
  description   = "ConTeXt Lua libraries -- extended collection.",
```

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


local stringformat     = string.format
local loadmodule       = lualibs.loadmodule
local texiowrite       = texio.write
local texiowrite_nl    = texio.write_nl

```

Here we define some functions that fake the elaborate logging/tracking mechanism ConTeXt provides.

```

local error, logger, mklog
if luatexbase and luatexbase.provides_module then
  --- TODO test how those work out when running tex
  local __error,_,_,__logger =
    luatexbase.provides_module(lualibs_extended_module)
  error  = __error
  logger = __logger
  mklog = function ( ) return logger end
else
  mklog = function (t)
    local prefix = stringformat("[%s] ", t)
    return function (...)
      texiowrite_nl(prefix)
      texiowrite   (stringformat(...))
    end
  end
  error  = mklog"ERROR"
  logger = mklog"INFO"
end

local info = lualibs.info

```

We temporarily put our own global table in place and restore whatever we overloaded afterwards.

ConTeXt modules each have a custom logging mechanism that can be enabled for debugging. In order to fake the presence of this facility we need to define at least the function `logs.reporter`. For now it's sufficient to make it a reference to `mklog` as defined above.

```

local dummy_function = function ( ) end
local newline        = function ( ) texiowrite_nl"" end

local fake_logs = function (name)
  return {
```

```
    name     = name,
    enable   = dummy_function,
    disable  = dummy_function,
    reporter = mklog,
    newline  = newline
  }
end

local fake_trackers = function (name)
  return {
    name     = name,
    enable   = dummy_function,
    disable  = dummy_function,
    register = mklog,
    newline  = newline,
  }
end

local backup_store = { }

local fake_context = function ( )
  if logs     then backup_store.logs     = logs     end
  if trackers then backup_store.trackers = trackers end
  logs     = fake_logs"logs"
  trackers = fake_trackers"trackers"
end


```

Restore a backed up logger if appropriate.

```
local unfake_context = function ( )
  if backup_store then
    local bl, bt = backup_store.logs, backup_store.trackers
    if bl   then logs     = bl   end
    if bt   then trackers = bt   end
  end
end

fake_context()

local loaded = false
if lualibs.prefer_merged then
  info"Loading merged package for collection "extended"."
  loaded = loadmodule('lualibs-extended-merged.lua')
else
  info"Ignoring merged packages."
  info"Falling back to individual libraries from collection "extended"."
end

if loaded == false then
```

```
  loadmodule("lualibs-util-str.lua")--- string formatters (fast)
  loadmodule("lualibs-util-fil.lua")--- extra file helpers
  loadmodule("lualibs-util-tab.lua")--- extended table operations
  loadmodule("lualibs-util-sto.lua")--- storage (hash allocation)
  ----------("lualibs-util-pck.lua")---!packers; necessary?
  ----------("lualibs-util-seq.lua")---!sequencers (function chaining)
  ----------("lualibs-util-mrg.lua")---!only relevant in mtx-package
  loadmodule("lualibs-util-prs.lua")--- miscellaneous parsers; cool. cool cool cool
  ----------("lualibs-util-fmt.lua")---!column formatter (rarely used)
  loadmodule("lualibs-util-dim.lua")--- conversions between dimensions
  loadmodule("lualibs-util-jsn.lua")--- JSON parser

  ----------("lualibs-trac-set.lua")---!generalization of trackers
  ----------("lualibs-trac-log.lua")---!logging
  loadmodule("lualibs-trac-inf.lua")--- timing/statistics
  loadmodule("lualibs-util-lua.lua")--- operations on lua bytecode
  loadmodule("lualibs-util-deb.lua")--- extra debugging
  loadmodule("lualibs-util-tpl.lua")--- templating
  loadmodule("lualibs-util-sta.lua")--- stacker (for writing pdf)
end

unfake_context() --- TODO check if this works at runtime

lualibs.extended_loaded = true
-- vim:tw=71:sw=2:ts=2:expandtab

```