

The luatexbase-modutils package

Heiko Oberdiek (primary author of luatex)
Élie Roux, Manuel Pégourié-Gonnard, Philipp Gesang*

<https://github.com/lualatex/luatexbase>
luatex-dev@tug.org

v0.6 2013-05-11

Abstract

This package provides functions similar to L^AT_EX's `\usepackage` and `\ProvidesPackage` macros,¹ or more precisely the part of these macros that deals with identification and version checking (no attempt is done at implementing an option mechanism). It also provides functions for reporting errors and warnings in a standardised format.

Contents

| | | |
|----------|--------------------------------------------------------|-----------|
| 1 | Documentation | 2 |
| 1.1 | Scope of this package | 2 |
| 1.2 | T _E X macros | 2 |
| 1.3 | Lua functions | 2 |
| 1.4 | Templates | 3 |
| 2 | Implementation | 4 |
| 2.1 | T _E X package | 4 |
| 2.1.1 | Preliminaries | 4 |
| 2.2 | Auxiliary definitions | 6 |
| 2.2.1 | User macro | 6 |
| 2.3 | Lua module | 7 |
| 2.4 | Internal functions and data | 7 |
| 2.4.1 | Error, warning and info function for modules | 7 |
| 2.4.2 | module loading and providing functions | 8 |
| 2.4.3 | module availability and version checking | 9 |
| 3 | Test files | 10 |

*See “History” in [luatexbase.pdf](#) for details.

¹and their variants or synonyms such as `\documentclass` and `\RequirePackage` or `\ProvidesClass` and `\ProvidesFiles`

1 Documentation

1.1 Scope of this package

Lua’s standard function `require()` is similar to \TeX ’s `\input` primitive but is somehow more evolved in that it makes a few checks to avoid loading the same module twice. In the \TeX world, this needs to be taken care of by macro packages; in the \LaTeX world this is done by `\usepackage`.

But `\usepackage` also takes care of many other things. Most notably, it implements a complex option system, and does some identification and version checking. The present package doesn’t try to provide anything for options, but implements a system for identification and version checking similar to \LaTeX ’s system.

It is important to note that Lua’s uncton `module()` is deprecated in Lua 5.2 and should be avoided. For examples of good practices for creating modules, see section 1.4. Chapter 15 of *Programming in Lua, 3rd ed.* discusses various methods for managing packages.

1.2 \TeX macros

```
\RequireLuaModule{<name>}[<date>]
```

The macro `\RequireLuaModule` is an interface to the Lua function `require_module`; it take the same arguments with the same meaning. The second argument is optional.

1.3 Lua functions

```
luatexbase.require_module(<name> [, <required date>])
```

The function `luatexbase.require_module()` may be used as a replacement to `require()`. If only one argument is given, the only difference with `require()` is it checks that the module properly identifies itself (as explained below) with the same name.

The second argument is optional; if used, it must be a string² containing a date in `YYYY//MM/DD` format which specifies the minimum version of the module required.

```
luatexbase.provides_module(<info>)
```

This function is used by modules to identify themselves; the argument is a table containing information about the module. The required field `name` must contain the name of the module. It is recommended to provide a field `date` with the same format as above. Optional fields `version` (number or string) and `description` may be used if present. Other fields are ignored.

If a date was required, then a warning is issued if the required date is strictly newer than the declared date (or if no date was declared). A list of loaded modules and their associated information is kept, and used to check the date without reloading the module (since `require()` won’t reload it anyway) if a module is required several times.

```
luatexbase.module_error(<name>, <message>, ...)  
luatexbase.module_warning(<name>, <message>, ...)  
luatexbase.module_info(<name>, <message>, ...)  
luatexbase.module_log(<name>, <message>, ...)
```

²Previous versions of the package supported floating-point version numbers as well, but it caused confusion with authors trying to use version strings such as `0.3a` and probably isn’t worth the trouble.

These functions are similar to L^AT_EX's `\PackageError`, `\PackageWarning` and `\PackageInfo` in the way they format the output. No automatic line breaking is done, you may still use `\n` as usual for that, and the name of the package will be prepended to each output line (except for `log` which is intended for short messages in a non-verbose format). The first argument is the name of the current module; the remaining arguments are passed to `string.format()`.

Note that `module_error` raises an actual Lua error with `error()`, which currently means a call stack will be dumped. While this may not look pretty, at least it provides useful information for tracking the error down.

```
local err, warn, info, log = luatexbase.errwarinf(<name>)
local err, warn, info, log = luatexbase.provides_module(<name>)
```

Customised versions of the above commands maybe obtained by invoking `errwarinf()` and are also returned by `provides_module()`. They don't take the name of the module as their first argument any more, so that you don't need to repeat it all over the place. (Notice that `error` is the name of a standard Lua function, so you may want to avoid overwriting it, hence the use of `err` in the above example.)

```
local module_info = luatexbase.get_module_info(<name>)
local version = luatexbase.get_module_version(<name>)
local date = luatexbase.get_module_date(<name>)
local date_int = luatexbase.get_module_date_int(<name>)
local is_loaded = luatexbase.is_module_loaded(<name>)
```

These functions check for the availability or version of a module, and can even return a copy of the `module_info` table.

1.4 Templates

Now, here is a module header template showing all the recommended elements:

```
local err, warn, info, log = luatexbase.provides_module({
  -- required
  name      = 'mymodule',
  -- recommended
  date      = '1970/01/01',
  version   = 0.0,          -- or version = '0.0a',
  description = 'a Lua module template',
  -- optional and ignored
  author    = 'A. U. Thor',
  licence   = 'LPPL v1.3+',
})

mynamespace      = mynamespace or { }
local mynamespace = mynamespace
```

Alternatively, if you don't want to assume `luatexbase-modutils` is loaded, you may load your module with:

```
(luatexbase.require_module or require)('mymodule')
```

and begin your module's code with:

```
if luatexbase._provides_module then
  luatexbase.provides_module({
    -- required
    name      = 'mymodule',
    -- recommended
    date      = '1970/01/01',
    version    = 0.0,          -- or version = '0.0a',
    description = 'a Lua module template',
    -- optional and ignored
    author     = 'A. U. Thor',
    licence    = 'LPPL v1.3+',
  })
end

mynamespace      = mynamespace or { }
local mynamespace = mynamespace

local function err(msg)
  -- etc.
```

2 Implementation

2.1 T_EX package

```
1 <texpackage>
```

2.1.1 Preliminaries

Catcode defenses and reload protection.

```
2 \begingroup\catcode61\catcode48\catcode32=10\relax% = and space
3 \catcode123 1 % {
4 \catcode125 2 % }
5 \catcode 35 6 % #
6 \toks0\expandafter{\expandafter\endlinechar\the\endlinechar}%
7 \edef\x{\endlinechar13}%
8 \def\y#1 #2 {%
9   \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
10  \edef\x{\x \catcode#1 #2}}%
11 \y 13 5 % carriage return
12 \y 61 12 % =
13 \y 32 10 % space
14 \y 123 1 % {
15 \y 125 2 % }
16 \y 35 6 % #
17 \y 64 11 % @ (letter)
18 \y 10 12 % new line ^^J
19 \y 34 12 % "
20 \y 39 12 % '
21 \y 40 12 % (
22 \y 41 12 % )
23 \y 44 12 % ,
```

```

24 \y 45 12 % -
25 \y 46 12 % .
26 \y 47 12 % /
27 \y 58 12 % :
28 \y 91 12 % [
29 \y 93 12 % ]
30 \y 94 7 % ^
31 \y 95 8 % _
32 \y 96 12 % '
33 \toks0\expandafter{\the\toks0 \relax\noexpand\endinput}%
34 \edef\y#1{\noexpand\expandafter\endgroup%
35 \noexpand\ifx#1\relax \edef#1{\the\toks0}\x\relax%
36 \noexpand\else \noexpand\expandafter\noexpand\endinput%
37 \noexpand\fi}%
38 \expandafter\y\cename luatexbase@modutils@sty@endinput\endcsname%

Package declaration.

39 \begingroup
40 \expandafter\ifx\cename ProvidesPackage\endcsname\relax
41 \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
42 \else
43 \let\x\ProvidesPackage
44 \fi
45 \expandafter\endgroup
46 \x{luatexbase-modutils}[2013/05/11 v0.6 Module utilities for LuaTeX]

Make sure LuaTeX is used.

47 \begingroup\expandafter\expandafter\expandafter\endgroup
48 \expandafter\ifx\cename RequirePackage\endcsname\relax
49 \input ifluatex.sty
50 \else
51 \RequirePackage{ifluatex}
52 \fi
53 \ifluatex\else
54 \begingroup
55 \expandafter\ifx\cename PackageError\endcsname\relax
56 \def\x#1#2#3{\begingroup \newlinechar10
57 \errhelp{#3}\errmessage{Package #1 error: #2}\endgroup}
58 \else
59 \let\x\PackageError
60 \fi
61 \expandafter\endgroup
62 \x{luatexbase-modutils}{LuaTeX is required for this package. Aborting.}{%
63 This package can only be used with the LuaTeX engine^^J%
64 (command 'lualatex' or 'luatex').^^J%
65 Package loading has been stopped to prevent additional errors.}
66 \expandafter\luatexbase@modutils@sty@endinput%
67 \fi

Load luatexbase-loader (hence luatexbase-compat), require the supporting Lua module and
make sure luaescapestring is available.

68 \expandafter\ifx\cename RequirePackage\endcsname\relax
69 \input luatexbase-loader.sty
70 \else

```

```

71 \RequirePackage{luatexbase-loader}
72 \fi
73 \luatexbase@directlua{require('luatexbase.modutils')}
74 \luatexbase@ensure@primitive{luaescapestring}

```

2.2 Auxiliary definitions

We need a version of `\@ifnextchar`. The definitions for the not- \LaTeX case are stolen from `ltxcmds` verbatim, only the prefix is changed.

```

75 \ifdefined\kernel@ifnextchar
76   \let\lltxb@ifnextchar\kernel@ifnextchar
77 \else
78   \chardef\lltxb@zero0
79   \chardef\lltxb@two2
80   \long\def\lltxb@ifnextchar#1#2#3{%
81     \begingroup
82     \let\lltxb@CharToken= #1\relax
83     \toks\lltxb@zero{#2}%
84     \toks\lltxb@two{#3}%
85     \futurelet\lltxb@LetToken\lltxb@ifnextchar@
86   }
87   \def\lltxb@ifnextchar@{%
88     \ifx\lltxb@LetToken\lltxb@CharToken
89       \expandafter\endgroup\the\toks\expandafter\lltxb@zero
90     \else
91       \ifx\lltxb@LetToken\lltxb@SpaceToken
92         \expandafter\expandafter\expandafter\lltxb@@ifnextchar
93       \else
94         \expandafter\endgroup\the\toks
95         \expandafter\expandafter\expandafter\lltxb@two
96       \fi
97     \fi
98   }
99   \begingroup
100   \def\x#1{\endgroup
101     \def\lltxb@@ifnextchar#1{%
102       \futurelet\lltxb@LetToken\lltxb@ifnextchar@
103     }%
104   }%
105   \x{ }
106   \begingroup
107   \def\x#1{\endgroup
108     \let\lltxb@SpaceToken= #1%
109   }%
110   \x{ }
111 \fi

```

2.2.1 User macro

Interface to the Lua function for module loading. Avoid passing a second argument to the function if empty (most probably not specified).

```

112 \def\RequireLuaModule#1{%

```

```

113 \lltxb@ifnextchar[{\lltxb@requirelua{#1}}{\lltxb@requirelua{#1}[]}]
114 \def\lltxb@requirelua#1[#2]{%
115   \luatexbase@directlua{luatexbase.require_module(
116     "\luatexluaescapestring{#1}"
117     \expandafter\ifx\expandafter\detokenize{#2}\else
118     , "\luatexluaescapestring{#2}"
119     \fi)}}
120 \luatexbase@modutils@sty@endinput%
121 \end{texpackage}

```

2.3 Lua module

```

122 (*luamodule)
123 luatexbase      = luatexbase or { }
124 local luatexbase = luatexbase
125 local string_gsub = string.gsub

```

2.4 Internal functions and data

Tables holding informations about the modules loaded and the versions required. Keys are module names and values are the info tables as passed to `provides_module()`.

```

126 local modules = modules or {}

    Convert a date in YYYY/MM/DD format into a number.

127 local function date_to_int(date)
128   if date == '' then return -1 end
129   local numbers = string_gsub(date, "(%d+)/(%d+)/(%d+)", "%1%2%3")
130   return tonumber(numbers)
131 end

```

2.4.1 Error, warning and info function for modules

Here are the reporting functions for the modules. An internal function is used for error messages, so that the calling level (last argument of `error()` remains constant using either `module_error()` or a custom version as returned by `errwarinf()`.

```

132 local function msg_format(msg_type, mod_name, ...)
133   local cont = '('..mod_name..'') .. ('Module: '..msg_type):gsub('.', ' ')
134   return 'Module '..mod_name..' '..msg_type..'':
135     .. string.format(...):gsub('\n', '\n'..cont) .. '\n'
136 end
137 local function module_error_int(mod, ...)
138   error(msg_format('error', mod, ...), 3)
139 end
140 local function module_error(mod, ...)
141   module_error_int(mod, ...)
142 end
143 luatexbase.module_error = module_error

```

Split the lines explicitly in order not to depend on the value of `\newlinechar`.

```

144 local function module_warning(mod, ...)
145   for _, line in ipairs(msg_format('warning', mod, ...):explode('\n')) do
146     texio.write_nl(line)
147   end

```

```

148 end
149 luatexbase.module_warning = module_warning
150 local function module_info(mod, ...)
151   for _, line in ipairs(msg_format('info', mod, ...):explode('\n')) do
152     texio.write_nl(line)
153   end
154 end
155 luatexbase.module_info = module_info

```

No line splitting or advanced formatting here.

```

156 local function module_log(mod, msg, ...)
157   texio.write_nl('log', mod..'': '..msg:format(...)')
158 end
159 luatexbase.module_log = module_log

```

Produce custom versions of the reporting functions.

```

160 local function errwarinf(name)
161   return function(...) module_error_int(name, ...) end,
162     function(...) module_warning(name, ...) end,
163     function(...) module_info(name, ...) end,
164     function(...) module_log(name, ...) end
165 end
166 luatexbase.errwarinf = errwarinf

```

For our own convenience, local functions for warning and errors in the present module.

```

167 local err, warn = errwarinf('luatexbase.modutils')

```

2.4.2 module loading and providing functions

Load a module with mandatory name checking and optional version checking.

```

168 local function require_module(name, req_date)
169   require(name)
170   local info = modules[name]
171   if not info then
172     warn("module '%s' was not properly identified", name)
173   elseif req_date and info.date then
174     if date_to_int(info.date) < date_to_int(req_date) then
175       warn("module '%s' required in version '%s'\n"
176         .. "but found in version '%s'", name, req_date, info.date)
177     end
178   end
179 end
180 luatexbase.require_module = require_module

```

Provide identification information for a module. As a bonus, custom reporting functions are returned. No need to do any check here, everything done in `require_module()`.

```

181 local function provides_module(info)
182   if not (info and info.name) then
183     err('provides_module: missing information')
184   end
185   texio.write_nl('log', string.format("Lua module: %s %s %s %s\n",
186     info.name, info.date or '', info.version or '', info.description or ''))
187   modules[info.name] = info

```



```

188     return errwarinf(info.name)
189 end
190 luatexbase.provides_module = provides_module

```

2.4.3 module availability and version checking

A simple table copy function.

```

191 local fastcopy
192 fastcopy = table.fastcopy or function(old)
193     if old then
194         local new = { }
195         for k,v in next, old do
196             if type(v) == "table" then
197                 new[k] = fastcopy(v)
198             else
199                 new[k] = v
200             end
201         end
202         local mt = getmetatable(old)
203         if mt then
204             setmetatable(new,mt)
205         end
206         return new
207     else
208         return { }
209     end
210 end

```

Gives the table of the infos on a module, as given in `provides_module`.

```

211 local function get_module_info(name)
212     local module_table = modules[name]
213     if not module_table then
214         return nil
215     else
216         return fastcopy(module_table)
217     end
218 end
219 luatexbase.get_module_info = get_module_info

```

Gives the version of a module, nil if the module is not loaded and empty string if the module did not set its date.

```

220 function get_module_version(name)
221     local module_table = modules[name]
222     if not module_table then
223         return nil
224     else
225         return module_table.version
226     end
227 end
228 luatexbase.get_module_version = get_module_version

```

Gives the date string of a module, nil if the module is not loaded and empty string if the modules did not set its date.

```

229 function get_module_date(name)
230     local module_table = modules[name]
231     if not module_table then
232         return nil
233     else
234         return module_table.date
235     end
236 end
237 luatexbase.get_module_date = get_module_date

```

Gives the date number of a module, for date comparison, nil if the module is not loaded and -1 if the module did not set its date. The number is formatted as `yyyymmdd`.

```

238 function get_module_date_int(name)
239     local module_table = modules[name]
240     if not module_table then
241         return nil
242     else
243         return module_table.date and date_to_int(module_table.date)
244     end
245 end
246 luatexbase.get_module_date_int = get_module_date_int

```

Returns true if the module is loaded, false otherwise.

```

247 function is_module_loaded(name)
248     if modules[name] then
249         return true
250     else
251         return false
252     end
253 end
254 luatexbase.is_module_loaded = is_module_loaded

```

We provide the module, for version checking.

```

255 provides_module({
256     name      = 'luatexbase-modutils',
257     date      = '2013/05/11',
258     version   = 0.6,
259     description = 'Module utilities for LuaTeX',
260 })
261 </luamodule>

```

3 Test files

A dummy lua file for tests.

```

262 <*testdummy>
263 local err, warn, info, log = luatexbase.provides_module {
264     name      = 'test-modutils',
265     date      = '2000/01/01',
266     version   = 1,
267     description = 'dummy test package',
268 }
269 luatexbase.provides_module {

```

```

270 name      = 'test-modutils2',
271 date       = '',
272 version    = 1,
273 description = 'dummy test package',
274 }
275 info('It works!\nOh, rly?\nYeah rly!')
276 log("I'm a one-line info.")
277 info("1 = "..luatexbase.get_module_version('test-modutils'))
278 if is_module_loaded('test-modutils') then
279   info("ok!")
280 else
281   err("problem!")
282 end
283 info("2000/01/01 = "..luatexbase.get_module_info('test-modutils').date)
284 info("20000101 = "..luatexbase.get_module_date_int('test-modutils'))
285 info("-1 = "..luatexbase.get_module_date_int('test-modutils2'))
286 </testdummy>

```

We just check that the package loads properly, under both LaTeX and Plain TeX, is able to load and identify the above dummy module.

```

287 <testplain>\input luatexbase-modutils.sty
288 <testlatex>\RequirePackage{luatexbase-modutils}
289 <*testplain,testlatex>
290 \RequireLuaModule{test-modutils}
291 \RequireLuaModule{test-modutils}[1970/01/01]
292 </testplain,testlatex>
293 <testplain>\bye
294 <testlatex>\stop

```