The **luatexbase-modutils** package

Manuel Pégourié-Gonnard & Élie Roux
Support: lualatex-dev@tug.org

v0.2 2010-05-12

Abstract

This package provides functions similar to LaTeX'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

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`.

---

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

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 notice that Lua's standard function `module()` is completely orthogonal with the present package. It has nothing to do with identification and deals only with namespaces: more precisely, it modifies the current environment. So, you should continue to use it normally regardless of whether you chose to use this package's features for identification.

It is recommended to always use `module()` or any other method that ensure the global name space remains clean. For example, you may heavily use the `local` keyword and explicitly qualify the name of every non-local symbol. Chapter 15 of Programming in Lua, 1st 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 string[2] 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⟩, ...)`

---

[2]Previous 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 LaTeX'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.)

1.4 Templates

Let me emphasize again that, while `luatexbase.require_module()` is meant to be used as a replacement for `require()`, the function `luatexbase.provides_module()` *is not* a replacement for `module()`: they just don't do the same thing (declaring information vs changing the current name space).

Now, here is how you module may begin:

```
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+',
})

module('mynamespace', package.seeall)
-- or any other method (see chapter 15 of PIL for examples)
```

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

module('mynamespace', package.seeall)
-- or any other method (see chapter 15 of PIL for examples)

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

2 Implementation

2.1 TEX package

1 ⟨∗texpackage⟩

2.1.1 Preliminaries

Reload protection, especially for Plain TEX.

```
2                 \csname lltxb@modutils@loaded\endcsname
3 \expandafter\let\csname lltxb@modutils@loaded\endcsname\endinput
```

Catcode defenses.

```
4 \begingroup
5   \catcode123 1 % {
6   \catcode125 2 % }
7   \catcode 35 6 % #
8   \toks0{}%
9   \def\x{}%
10  \def\y#1 #2 {%
11    \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
12    \edef\x{\x \catcode#1 #2}}%
13  \y 123 1  % {
14  \y 125 2  % }
15  \y  35 6  % #
16  \y  10 12 % ^^J
17  \y  34 12 % "
18  \y  36 3  % $ $
19  \y  39 12 % '
20  \y  40 12 % (
21  \y  41 12 % )
22  \y  42 12 % *
23  \y  43 12 % +
24  \y  44 12 % ,
25  \y  45 12 % -
```

```
26  \y  46 12 % .
27  \y  47 12 % /
28  \y  60 12 % <
29  \y  61 12 % =
30  \y  64 11 % @ (letter)
31  \y  62 12 % >
32  \y  95 12 % _ (other)
33  \y  96 12 % `
34  \edef\y#1{\endgroup\edef#1{\the\toks0\relax}\x}%
35 \expandafter\y\csname lltxb@modutils@AtEnd\endcsname
```

Package declaration.

```
36 \begingroup
37   \expandafter\ifx\csname ProvidesPackage\endcsname\relax
38     \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
39   \else
40     \let\x\ProvidesPackage
41   \fi
42 \expandafter\endgroup
43 \x{luatexbase-modutils}[2010/10/10 v0.3 Module utilities for LuaTeX]
```

Make sure LuaTeX is used.

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

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

```
66 \ifdefined\RequirePackage
67   \RequirePackage{luatexbase-loader}
68 \else
69   \input luatexbase-loader.sty
70 \fi
71 \luatexbase@directlua{require('luatexbase.modutils')}
72 \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.

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

```
110 \def\RequireLuaModule#1{%
111   \lltxb@ifnextchar[{\lltxb@requirelua{#1}}{\lltxb@requirelua{#1}[]}}
112 \def\lltxb@requirelua#1[#2]{%
113   \luatexbase@directlua{luatexbase.require_module(
114     "\luatexluaescapestring{#1}"
115     \expandafter\ifx\expandafter\/\detokenize{#2}\/\else
116       , "\luatexluaescapestring{#2}"
```

```
117    \fi)}}
118 \lltxb@modutils@AtEnd
119 ⟨/texpackage⟩
```

2.3 Lua module

```
120 ⟨∗luamodule⟩
121 module("luatexbase", package.seeall)
```

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()`.

```
122 local modules = modules or {}
```

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

```
123 local function date_to_int(date)
124     numbers = string.gsub(date, "(%d+)/(%d+)/(%d+)", "%1%2%3")
125     return tonumber(numbers)
126 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()`.

```
127 local function msg_format(msg_type, mod_name, ...)
128   local cont = '('..mod_name..')' .. ('Module: '..msg_type):gsub('.', ' ')
129   return 'Module '..mod_name..' '..msg_type..': '
130     .. string.format(...):gsub('\n', '\n'..cont) .. '\n'
131 end
132 local function module_error_int(mod, ...)
133   error(msg_format('error', mod, ...), 3)
134 end
135 function module_error(mod, ...)
136   module_error_int(mod, ...)
137 end
```

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

```
138 function module_warning(mod, ...)
139   for _, line in ipairs(msg_format('warning', mod, ...):explode('\n')) do
140     texio.write_nl(line)
141   end
142 end
143 function module_info(mod, ...)
144   for _, line in ipairs(msg_format('info', mod, ...):explode('\n')) do
145     texio.write_nl(line)
146   end
147 end
```

No line splitting or advanced formating here.

```
148 function module_log(mod, msg, ...)
149   texio.write_nl('log', mod..': '..msg:format(...))
150 end
```

Produce custom versions of the reporting functions.

```
151 function errwarinf(name)
152   return function(...) module_error_int(name, ...) end,
153     function(...) module_warning(name, ...) end,
154     function(...) module_info(name, ...) end,
155     function(...) module_log(name, ...) end
156 end
```

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

```
157 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.

```
158 function require_module(name, req_date)
159     require(name)
160     local info = modules[name]
161     if not info then
162         warn("module '%s' was not properly identified", name)
163     elseif version then
164         if not (info.date and date_to_int(info.date) > date_to_int(req_date))
165         then
166             warn("module '%s' required in version '%s'\n"
167             .. "but found in version '%s'", name, req_date, info.date)
168         end
169     end
170 end
```

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()`.

```
171 function provides_module(info)
172     if not (info and info.name) then
173         err('provides_module: missing information')
174     end
175     texio.write_nl('log', string.format("Lua module: %s %s %s %s\n",
176     info.name, info.date or '', info.version or '', info.description or ''))
177     modules[info.name] = info
178     return errwarinf(info.name)
179 end
```

180 ⟨/luamodule⟩

3 Test files

A dummy lua file for tests.

```
181 ⟨*testdummy⟩
182 local err, warn, info, log = luatexbase.provides_module {
183   name        = 'test-modutils',
184   date        = '2000/01/01',
185   version     = 1,
186   description = 'dummy test package',
187 }
```

```
188 info('It works!\nOh, rly?\nYeah rly!')
189 log("I'm a one-line info.")
190 ⟨/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.

```
191 ⟨testplain⟩\input luatexbase-modutils.sty
192 ⟨testlatex⟩\RequirePackage{luatexbase-modutils}
193 ⟨∗testplain, testlatex⟩
194 \RequireLuaModule{test-modutils}
195 \RequireLuaModule{test-modutils}[1970/01/01]
196 ⟨/testplain, testlatex⟩
197 ⟨testplain⟩\bye
198 ⟨testlatex⟩\stop
```